pyfiberamp Documentation

Release 0.1.0

Joona Rissanen

Oct 23, 2019

Getting started:

CHAPTER 1

Introduction to PyFiberAmp

PyFiberAmp is a rate equation simulation library for rare-earth-doped fiber amplifiers and fiber lasers partly based on the Giles model[1].

With PyFiberAmp you can simulate:

- Both core-pumped and double-clad fiber amplifiers
- Simple continuous-wave, gain-switched and Q-switched fiber lasers
- Unlimited number of pump, signal and ASE channels
- Limited number of Raman channels
- Arbitrarily time-dependent beams from continuous-wave to nanosecond pulses
- Radially varying dopant concentration and excitation
- Automatically calculated Bessel, Gaussian and top-hat mode shapes

Additional benefits include:

- Built-in plotting commands: easy visualization of results
- Python interface: convenient for post-processing the data
- C++, Numba and Pythran backends: fast time-dynamic simulations
- Open source: see what's happening under the hood
- Free of charge: install on as many computers as you like

Documentation is still in progress and available on Read the Docs. For practical examples, see the examples folder above. If you have a question, comment or feature request, please open a new issue on GitHub or contact me at pyfiberamp@gmail.com. If you find PyFiberAmp useful in your own project, I would also very much like to hear about it.

[1] C.R. Giles and E. Desurvire, "Modeling erbium-doped fiber amplifiers," in Journal of Lightwave Technology, vol. 9, no. 2, pp. 271-283, Feb 1991. doi: 10.1109/50.65886

1.1 A visual example

Few-nanosecond pulses propagating in an Yb-doped fiber amplifier are distorted because of gain saturation. The Gaussian pulse with its exponential leading edge retains its shape better than the square or saw-tooth pulses.

1.2 Download

PyFiberAmp is not yet on PyPI. You can either download the code as a zip-file or clone the repository with

```
git clone git://github.com/Jomiri/pyfiberamp.git
```

and then install the library by executing

```
python setup.py install
```

in the (unzipped) download directory.

1.3 System requirements

PyFiberAmp depends on the standard scientific Python packages: Numpy, SciPy and Matplotlib and has been tested on Windows 7 and Windows 10. It should work on other operating systems as well provided that Python and the required packages are installed. The Anaconda distribution contains everything you'll need out of the box.

Even though all of PyFiberAmp's functionality is available in interpreted Python code, the use of one of the compiled backends (C++, Numba or Pythran) is recommended for computationally intensive time-dynamic simulations. The hand-written C++ extension is fastest but has also the strictest system requirements: Windows 7 or 10, Python 3.6 and a fairly modern CPU with AVX2 instruction support. The Pythran backend probably only works on Linux and requires that pythran is installed before installing PyFiberAmp. The Numba backend should work on all operating systems provided that Numba is available. Please open a new issue if you encounter problems with a backend that should work but does not.

1.4 Example

The simple example below demonstrates a core-pumped Yb-doped fiber amplifier. All units are in SI.

```
from pyfiberamp.steady_state import SteadyStateSimulation
from pyfiberamp.fibers import YbDopedFiber

yb_number_density = 2e25  # m^-3
core_radius = 3e-6  # m
length = 2.5  # m
core_na = 0.12

fiber = YbDopedFiber(length=length,
                     core_radius=core_radius,
                     ion_number_density=yb_number_density,
                     background_loss=0,
                     core_na=core_na)
```

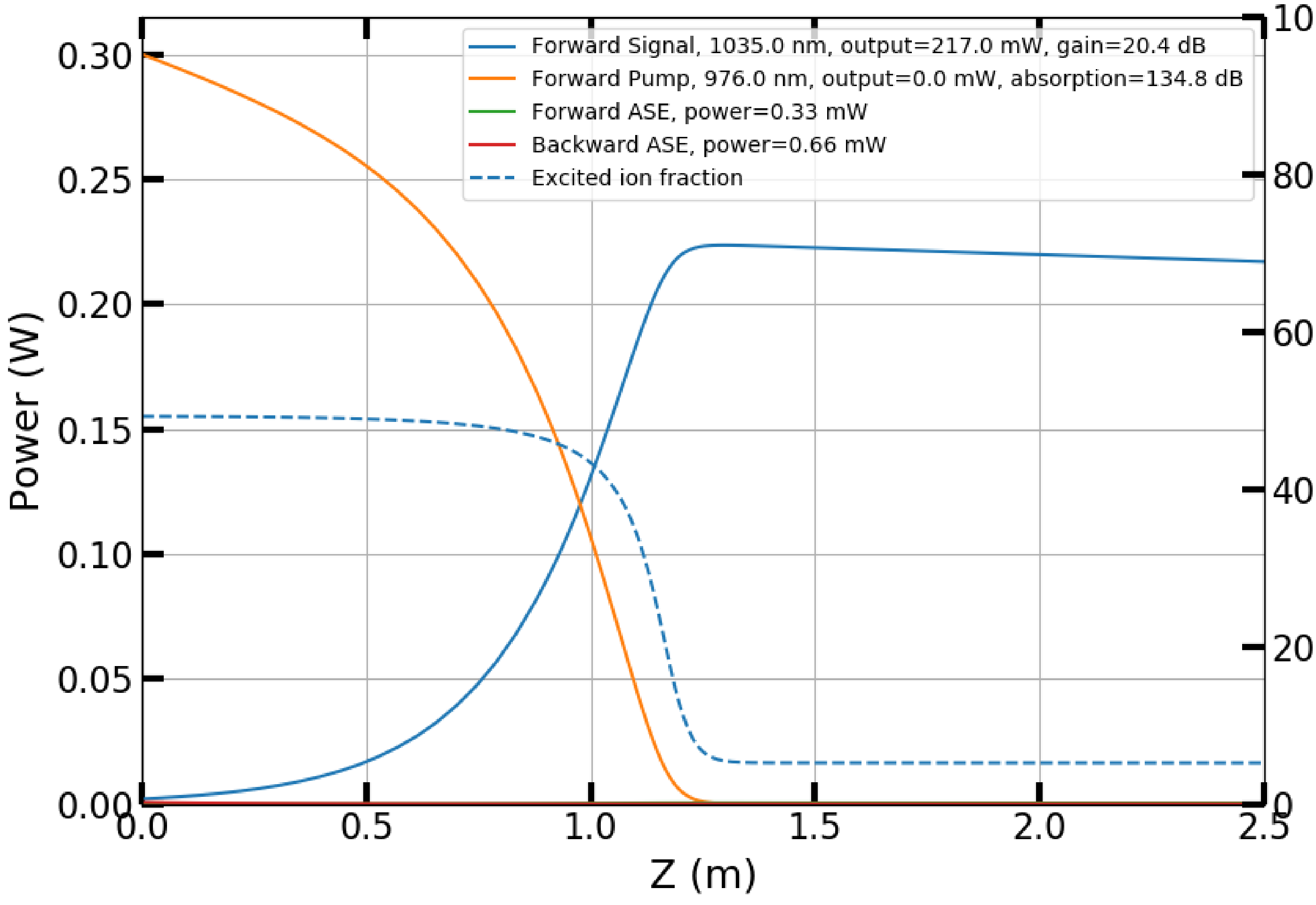
(continues on next page)

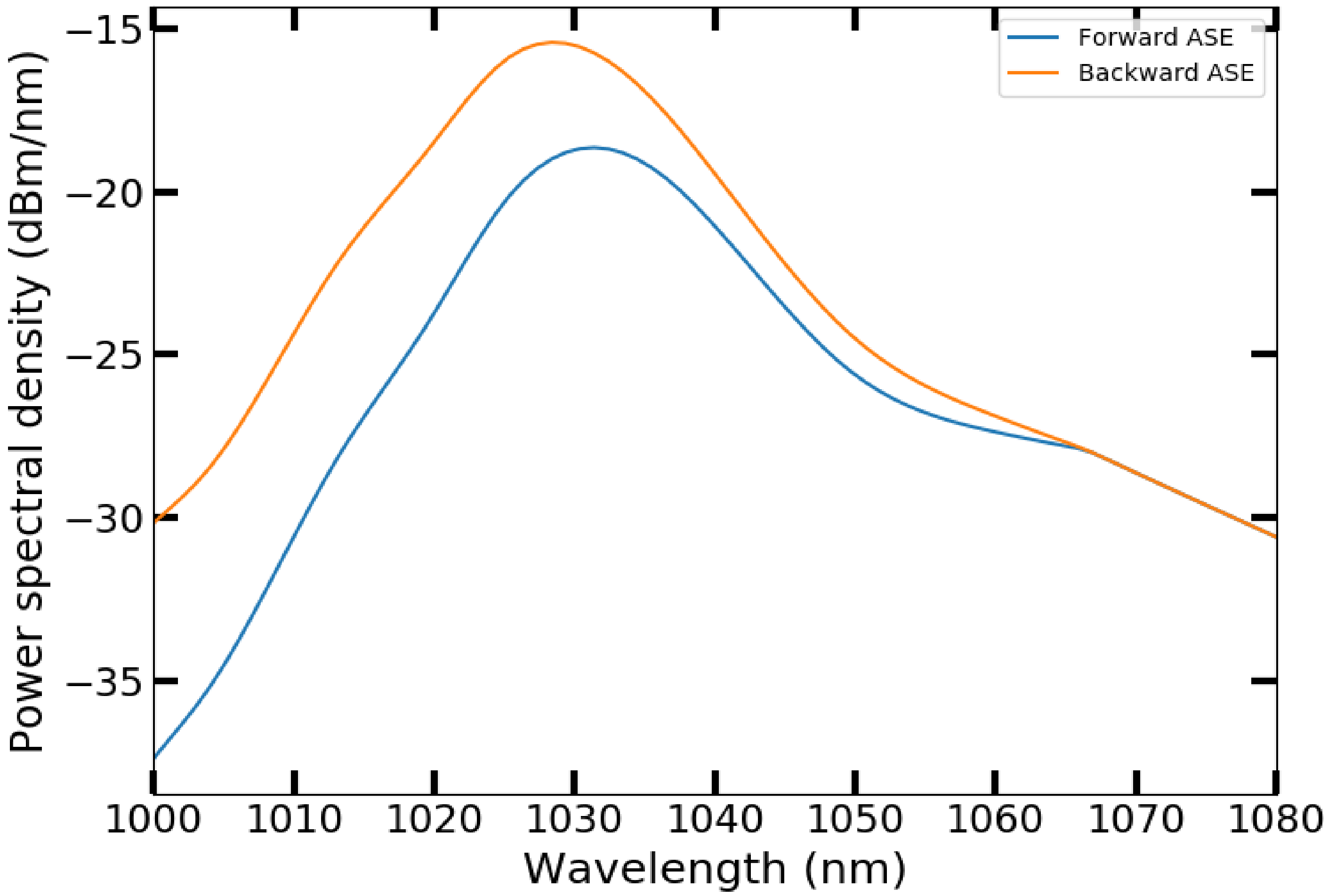
(continued from previous page)

```
simulation = SteadyStateSimulation()
simulation.fiber = fiber
simulation.add_cw_signal(wl=1035e-9, power=2e-3)
simulation.add_forward_pump(wl=976e-9, power=300e-3)
simulation.add_ase(wl_start=1000e-9, wl_end=1080e-9, n_bins=80)

result = simulation.run(tol=1e-5)
result.plot_amplifier_result()
```

The script calculates and plots the power evolution in the amplifier and the amplified spontaneous emission (ASE) spectra. The co-propagating pump is absorbed in the first ~1.2 m of the fiber while the signal experiences gain. When the pump has been depleted, the signal starts to be reabsorbed. ASE is stronger against the pumping direction.





For more usage examples, please see the Jupyter notebooks in the examples folder. More examples will be added in the future.

1.5 Fiber data

PyFiberAmp comes with spectroscopic data (effective absorption and emission cross sections) for Yb-doped germanosilicate fibers[3] and supports importing spectra for other dopants and glass compositions.

1.6 Theory basics

For a quick review on the theory, see the pyfiberamp theory.pdf file. Theory on the time-dynamic simulations is not yet included. A more complete description can be found in the references.

3

R. Paschotta, J. Nilsson, A. C. Tropper and D. C. Hanna, "Ytterbium-doped fiber amplifiers," in IEEE Journal of Quantum Electronics, vol. 33, no. 7, pp. 1049-1056, Jul 1997. doi: 10.1109/3.594865

1.7 License

PyFiberAmp is licensed under the MIT license. The C++ extension depends on the pybind11 and Armadillo projects. See the license file for their respective licenses.

1.8 References

CHAPTER 2

Simulation types

2.1 Steady state simulation

class SteadyStateSimulation

SteadyStateSimulation is the main class used for running steady state Giles model simulations without Raman scattering. Only one ion population is supported. The class defines the fiber, boundary conditions and optical channels used in the simulation.

__init__()

Initialize self. See help(type(self)) for accurate signature.

add_cw_signal(*wl*, *power*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label=''*)

Adds a new forward propagating single-frequency CW signal to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **power** (*float*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of [‘bessel’, ‘gaussian’, ‘tophat’] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel

add_forward_pump(*wl*, *power*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label=''*)

Adds a new forward propagating single-frequency pump to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **power** (*float*) – Input power of the signal at the beginning of the fiber

- **`wl_bandwidth`** (*`float`*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **`mode_shape_parameters`** (*`dict`*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **`label`** (*`str`*) – Optional label for the channel

`add_backward_pump` (*wl*, *power*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label=''*)
Adds a new backward propagating single-frequency pump to the simulation.

Parameters

- **`wl`** (*`float`*) – Wavelength of the signal
- **`power`** (*`float`*) – Input power of the signal at the beginning of the fiber
- **`wl_bandwidth`** (*`float`*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **`mode_shape_parameters`** (*`dict`*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **`label`** (*`str`*) – Optional label for the channel

`add_ase` (*wl_start*, *wl_end*, *n_bins*)
Adds amplified spontaneous emission (ASE) channels. Using more channels improves accuracy, but incurs a heavier computational cost to the simulation.

Parameters

- **`wl_start`** (*`float`*) – The shorted wavelength of the ASE band
- **`wl_end`** (*`float`*) – The longest wavelength of the ASE band
- **`n_bins`** (*`positive int`*) – The number of simulated ASE channels.

`run` (*tol=0.001*)
Runs the simulation, i.e. calculates the steady state of the defined fiber amplifier. ASE or raman simulations might require higher tolerance than the default value. It is best to decrease the tolerance until the result no longer changes.

Parameters **`tol`** (*`float`*) – Target error tolerance of the solver.

`set_guess_parameters` (*guess_parameters*)
Overrides the default initial guess parameters.

Parameters **`guess_parameters`** (*`Instance of GuessParameters class`*) – Parameters used to create the initial guess array

Example

from pyfiberamp import GuessParameters, GainShapes params = GuessParameters() params.signal.set_gain_shape(GainShapes.LINEAR) params.pump.set_gain_db(-20) simulation.set_guess_parameters(params)

`set_guess_array` (*array*, *force_node_number=None*)
Use an existing array as the initial guess. Typically this array is the result of a previous simulation with sligthly different parameters. Note that the number of simulated beams/channels must be the same.

Parameters

- **array** (*numpy array*) – The initial guess array
- **force_node_number** (*int, optional*) – The new number of columns in the resized array.

set_number_of_nodes (*N*)
Override the default number of nodes used by the solver. The solver will increase the number of nodes if necessary.

Parameters **N** (*int*) – New starting number of nodes used by the solver.

2.2 Steady state simulation with Raman scattering

class SteadyStateSimulationWithRaman
SteadyStateSimulationWithRaman is the main class for running Giles model simulations with Raman scattering. Only one ion population is supported. The class defines the fiber, boundary conditions and optical channels used in the simulation.

__init__ ()
Initialize self. See help(type(self)) for accurate signature.

add_pulsed_signal (*wl*, *power*, *f_rep*, *fwhm_duration*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label=''*)
Adds a new forward propagating single-frequency pulsed signal to the simulation. A pulsed signal has a higher peak power resulting in stronger nonlinear effects, in particular spontaneous and stimulated Raman scattering. The pulse shape is assumed to be Gaussian.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **power** (*float*) – Input power of the signal at the beginning of the fiber
- **f_rep** (*float*) – Repetition frequency of the signal
- **fwhm_duration** (*float*) – Full-width at half-maximum duration of the Gaussian pulses
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel

add_raman (*input_power=1e-14*, *backward_raman_allowed=True*, *raman_gain=1e-13*)
Adds Raman channels to the simulation.

Parameters

- **backward_raman_allowed** (*bool, default True*) – Determines if only the forward propagating Raman beam is simulated.
- **input_power** (*float, default ~0 W*) – Input power of the Raman beam(s)
- **raman_gain** (*float, default 1e-13 m/W*) – Raman gain value to be used in the simulation.

add_ase (*wl_start*, *wl_end*, *n_bins*)
Adds amplified spontaneous emission (ASE) channels. Using more channels improves accuracy, but incurs a heavier computational cost to the simulation.

Parameters

- **wl_start** (*float*) – The shorted wavelength of the ASE band
- **wl_end** (*float*) – The longest wavelength of the ASE band
- **n_bins** (*positive int*) – The number of simulated ASE channels.

add_backward_pump (*wl*, *power*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label="*)
Adds a new backward propagating single-frequency pump to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **power** (*float*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel

add_cw_signal (*wl*, *power*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label="*)
Adds a new forward propagating single-frequency CW signal to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **power** (*float*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel

add_forward_pump (*wl*, *power*, *wl_bandwidth=0*, *mode_shape_parameters=None*, *label="*)
Adds a new forward propagating single-frequency pump to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **power** (*float*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations

- **`label`** (*`str`*) – Optional label for the channel

`run`(*tol=0.001*)
Runs the simulation, i.e. calculates the steady state of the defined fiber amplifier. ASE or raman simulations might require higher tolerance than the default value. It is best to decrease the tolerance until the result no longer changes.

Parameters **`tol`** (*`float`*) – Target error tolerance of the solver.

`set_guess_array`(*array*, *force_node_number=None*)
Use an existing array as the initial guess. Typically this array is the result of a previous simulation with sligthly different parameters. Note that the number of simulated beams/channels must be the same.

Parameters

- **`array`** (*`numpy array`*) – The initial guess array
- **`force_node_number`** (*`int, optional`*) – The new number of columns in the resized array.

`set_guess_parameters`(*guess_parameters*)
Overrides the default initial guess parameters.

Parameters **`guess_parameters`** (*`Instance of GuessParameters class`*) – Parameters used to create the initial guess array

Example

from pyfiberamp import GuessParameters, GainShapes params = GuessParameters() params.signal.set_gain_shape(GainShapes.LINEAR) params.pump.set_gain_db(-20) simulation.set_guess_parameters(params)

`set_number_of_nodes`(*N*)
Override the default number of nodes used by the solver. The solver will increase the number of nodes if necessary.

Parameters **`N`** (*`int`*) – New starting number of nodes used by the solver.

2.3 Dynamic simulation

class DynamicSimulation(*max_time_steps*)
DynamicSimulation is the interface class used for running fiber amplifier simulations with arbitrarily varying input powers. It also supports reflective boundary conditions and thus modeling of simple CW, gain-switched or Q-switched fiber lasers. With constant input powers, the result converges to the steady state simulation result. Setting multiple ion populations is also supported. The class defines the fiber, boundary conditions and optical channels used in the simulation.

`__init__`(*max_time_steps*)
Initialize self. See help(type(self)) for accurate signature.

`use_python_backend`()
Sets the simulation to use the slow Python finite difference solver. Using one of the faster solvers instead is highly recommended.

`use_cpp_backend`()
Sets the simulation to use the C++ backend if available.

`use_pythran_backend`()
Sets the simulation to use the pythran backend if available.

use_numba_backend()

Sets the simulation to use the numba backend if available.

get_time_coordinates(*fiber*, *z_nodes*, *dt='auto'*)

Returns the time coordinates used in the simulation. Useful for setting time-varying input powers.

Parameters

- **fiber** (*Subclass of FiberBase*) – The fiber used in the simulation
- **z_nodes** (*int*) – Number of spatial nodes used in the simulation.
- **dt** (*float*) – Time step size. The 'auto' option uses realistic time step calculated from the Courant condition based on the speed of light in glass and the spatial step size. Larger (and physically unrealistic) time steps can be used to drastically speed up the convergence of steady state simulations.

Returns Time coordinate array

Return type numpy float array

add_forward_signal(*wl*, *input_power*, *wl_bandwidth=0.0*, *mode_shape_parameters=None*, *label="*, *reflection_target="*, *reflectance=0*)

Adds a new forward-propagating signal to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **input_power** (*float or numpy array*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel (required to receive reflected power from another channel)
- **reflection_target** (*str*) – Label of the channel receiving reflection from this channel
- **reflectance** – Reflectance R [0,1] from this channel to the target channel

add_backward_signal(*wl*, *input_power*, *wl_bandwidth=0.0*, *mode_shape_parameters=None*, *label="*, *reflection_target="*, *reflectance=0*)

Adds a new backward-propagating signal to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **input_power** (*float or numpy array*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* ->

float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations

- **label** (*str*) – Optional label for the channel (required to receive reflected power from another channel)
- **reflection_target** (*str*) – Label of the channel receiving reflection from this channel
- **reflectance** – Reflectance R [0,1] from this channel to the target channel

add_forward_pump(*wl*, *input_power*, *wl_bandwidth=0.0*, *mode_shape_parameters=None*, *label=''*, *reflection_target=''*, *reflectance=0*)

Adds a new forward-propagating pump to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **input_power** (*float or numpy array*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel (required to receive reflected power from another channel)
- **reflection_target** (*str*) – Label of the channel receiving reflection from this channel
- **reflectance** – Reflectance R [0,1] from this channel to the target channel

add_backward_pump(*wl*, *input_power*, *wl_bandwidth=0.0*, *mode_shape_parameters=None*, *label=''*, *reflection_target=''*, *reflectance=0*)

Adds a new backward-propagating pump to the simulation.

Parameters

- **wl** (*float*) – Wavelength of the signal
- **input_power** (*float or numpy array*) – Input power of the signal at the beginning of the fiber
- **wl_bandwidth** (*float*) – Wavelength bandwidth of the channel. Finite bandwidth means including ASE.
- **mode_shape_parameters** (*dict*) – Defines the mode field shape. Allowed key-value pairs: *functional_form* -> one of ['bessel', 'gaussian', 'tophat'] *mode_diameter* -> float *overlaps* -> list of pre-calculated overlaps between the channel and the ion populations
- **label** (*str*) – Optional label for the channel (required to receive reflected power from another channel)
- **reflection_target** (*str*) – Label of the channel receiving reflection from this channel
- **reflectance** – Reflectance R [0,1] from this channel to the target channel

add_ase (*wl_start*, *wl_end*, *n_bins*)
Adds amplified spontaneous emission (ASE) channels. Using more channels improves accuracy, but incurs a heavier computational cost to the simulation.

Parameters

- **wl_start** (*float*) – The shorted wavelength of the ASE band
- **wl_end** (*float*) – The longest wavelength of the ASE band
- **n_bins** (*positive int*) – The number of simulated ASE channels.

run (*z_nodes*, *dt='auto'*, *P=None*, *N2=None*, *stop_at_steady_state=False*, *steady_state_tolerance=0.0001*, *convergence_checking_interval=10000*)
Runs the simulation.

Parameters

- **z_nodes** (*int*) – Number of spatial nodes used in the simulation.
- **dt** (*float or str*) – Time step size. The 'auto' option uses realistic time step calculated from the Courant condition based on the speed of light in glass and the spatial step size. Larger (and physically unrealistic) time steps can be used to drastically speed up the convergence of steady state simulations.
- **P** (*numpy float array*) – Pre-existing powers in the fiber, useful when chaining multiple simulations.
- **N2** (*numpy float array*) – Pre-existing upper state excitation in the fiber, useful when chaining multiple simulations.
- **stop_at_steady_state** (*bool*) – If this flag parameter is set to True, the simulation stops when the excitation reaches a steady state (does not work if the excitation fluctuates at a specific frequency).
- **steady_state_tolerance** (*float*) – Sets the relative change in excitation that is used to detect the steady state.
- **convergence_checking_interval** (*positive int*) – If aiming for steady state, the simulation checks convergence always after this number of iterations and prints the average excitation. In truly dynamic simulations, only prints the excitation.

CHAPTER 3

Fiber types

3.1 Passive fiber

class PassiveFiber(*length=0*, *core_radius=0*, *background_loss=0*, *core_na=0*)
PassiveFiber describes a step-index single-mode fiber with no dopant ions. It extends the FiberBase class by stating that there is no emission or absorption by ions. The only possible gain comes of stimulated Raman scattering.

__init__(*length=0*, *core_radius=0*, *background_loss=0*, *core_na=0*)

Parameters

- **length** (*float*) – Fiber length
- **core_radius** (*float*) – Core radius
- **background_loss** (*float*) – Linear loss of the core (1/m, NOT in dB/m)
- **core_na** (*float*) – Numerical aperture of the core

get_channel_emission_cross_section(*freq*, *frequency_bandwidth*)
Passive fiber has no gain.

get_channel_absorption_cross_section(*freq*, *frequency_bandwidth*)
Passive fiber has no absorption by dopant ions.

core_area()
Returns the core area of the fiber defined as pi*r**2, where r is the core radius.

Returns Core area

Return type float

nonlinear_effective_area(*freq*)
Returns the nonlinear effective area of the fundamental fiber mode with the given frequency. The method used is determined by the attribute self.effective_area_type.

Parameters **freq** (*float or numpy float array*) – The frequency of the optical signal (Hz).

Returns The nonlinear effective area

Return type Same as argument type.

3.2 Active fiber

class ActiveFiber(*length=0*, *core_radius=0*, *background_loss=0*, *core_na=0*, *spectroscopy=None*, *ion_number_density=0*)

ActiveFiber describes a step-index single-mode fiber with active dopant ions. Currently, only uniform doping in the whole core area is supported. This class extends the FiberBase class by adding spectroscopic data: gain and emission spectra, upper state lifetime and doping concentration.

classmethod from_cross_section_files(*length*, *absorption_cs_file=None*, *emission_cs_file=None*, *core_radius=0*, *upper_state_lifetime=0*, *ion_number_density=0*, *background_loss=0*, *core_na=0*)

Parameters

- **length** (`float`) – Fiber length
- **absorption_cs_file** (`str`) – Name of the file containing absorption cross-section data
- **emission_cs_file** (`str`) – Name of the file containing emission cross-section data
- **core_radius** (`float`) – Core radius
- **upper_state_lifetime** (`float`) – Lifetime of the excited state
- **ion_number_density** (`float`) – Number density of the dopant ions (1/m^3)
- **background_loss** (`float`) – Linear loss of the core (1/m, NOT in dB/m)
- **core_na** (`float`) – Numerical aperture of the core

__init__(*length=0*, *core_radius=0*, *background_loss=0*, *core_na=0*, *spectroscopy=None*, *ion_number_density=0*)

Parameters

- **length** (`float`) – Fiber length
- **core_radius** (`float`) – Core radius
- **background_loss** (`float`) – Linear loss of the core (1/m, NOT in dB/m)
- **core_na** (`float`) – Numerical aperture of the core
- **spectroscopy** (`Spectroscopy`) – The spectroscopic properties of the fiber.
- **ion_number_density** (`float`) – Number density of the dopant ions (1/m^3)

saturation_parameter()

Returns the constant saturation parameter zeta defined in the Giles model.

core_area()

Returns the core area of the fiber defined as pi*r**2, where r is the core radius.

Returns Core area

Return type float

nonlinear_effective_area (*freq*)

Returns the nonlinear effective area of the fundamental fiber mode with the given frequency. The method used is determined by the attribute self.effective_area_type.

Parameters **freq** (*float or numpy float array*) – The frequency of the optical signal (Hz).

Returns The nonlinear effective area

Return type Same as argument type.

3.3 Double-clad fiber

class DoubleCladFiber (*length=0, core_radius=0, background_loss=0, core_na=0, spectroscopy=None, ion_number_density=0, ratio_of_core_and_cladding_diameters=0*)

DoubleCladFiber extends ActiveFiber and describes a double-clad active fiber with single-mode step-index core. Flat-top pump distribution in the pump cladding is assumed as well as constant overlap between the pump modes and the core. All pump beams propagate in the pump cladding.

classmethod from_cross_section_files (*length=0, absorption_cs_file=None, emission_cs_file=None, core_radius=0, upper_state_lifetime=0, ion_number_density=0, background_loss=0, core_na=0, ratio_of_core_and_cladding_diameters=0*)

Parameters

- **length** (*float*) – Fiber length
- **absorption_cs_file** (*str*) – Name of the file containing absorption cross-section data
- **emission_cs_file** (*str*) – Name of the file containing emission cross-section data
- **core_radius** (*float*) – Core radius
- **upper_state_lifetime** (*float*) – Lifetime of the excited state
- **ion_number_density** (*float*) – Number density of the dopant ions (1/m^3)
- **background_loss** (*float*) – Linear loss of the core (1/m, NOT in dB/m)
- **core_na** (*float*) – Numerical aperture of the core
- **ratio_of_core_and_cladding_diameters** (*float*) – Core diameter divided by cladding diameter

__init__ (*length=0, core_radius=0, background_loss=0, core_na=0, spectroscopy=None, ion_number_density=0, ratio_of_core_and_cladding_diameters=0*)

Parameters

- **length** (*float*) – Fiber length
- **core_radius** (*float*) – Core radius
- **background_loss** (*float*) – Linear loss of the core (1/m, NOT in dB/m)
- **core_na** (*float*) – Numerical aperture of the core
- **spectroscopy** (Spectroscopy) – The spectroscopic properties of the fiber.
- **ion_number_density** (*float*) – Number density of the dopant ions (1/m^3)

- **`ratio_of_core_and_cladding_diameters`** (*`float`*) – Core diameter divided by cladding diameter

`pump_to_core_overlap`()
Returns the overlap between the core and the pump beams, which equals to the ratio of core and cladding area.

`pump_cladding_radius`()
Returns the radius of the fiber's pump cladding.

`core_area`()
Returns the core area of the fiber defined as pi*r**2, where r is the core radius.

Returns Core area

Return type float

`nonlinear_effective_area`(*freq*)
Returns the nonlinear effective area of the fundamental fiber mode with the given frequency. The method used is determined by the attribute self.effective_area_type.

Parameters **`freq`** (*`float or numpy float array`*) – The frequency of the optical signal (Hz).

Returns The nonlinear effective area

Return type Same as argument type.

`saturation_parameter`()
Returns the constant saturation parameter zeta defined in the Giles model.

3.4 Yb-doped fiber

`class YbDopedFiber`(*length=0, core_radius=0, ion_number_density=0, background_loss=0, core_na=0*)
YbDopedFiber is a convenience class for Yb-doped single-mode fiber that uses the default spectroscopic data for Yb-ions.

`__init__`(*length=0, core_radius=0, ion_number_density=0, background_loss=0, core_na=0*)

Parameters

- **`length`** (*`float`*) – Fiber length
- **`core_radius`** (*`float`*) – Core radius
- **`ion_number_density`** (*`float`*) – Number density of the dopant ions (1/m^3)
- **`background_loss`** (*`float`*) – Linear loss of the core (1/m, NOT in dB/m)
- **`core_na`** (*`float`*) – Numerical aperture of the core

`core_area`()
Returns the core area of the fiber defined as pi*r**2, where r is the core radius.

Returns Core area

Return type float

`classmethod from_cross_section_files`(*length, absorption_cs_file=None, emission_cs_file=None, core_radius=0, upper_state_lifetime=0, ion_number_density=0, background_loss=0, core_na=0*)

Parameters

- **`length`** (*float*) – Fiber length
- **`absorption_cs_file`** (*str*) – Name of the file containing absorption cross-section data
- **`emission_cs_file`** (*str*) – Name of the file containing emission cross-section data
- **`core_radius`** (*float*) – Core radius
- **`upper_state_lifetime`** (*float*) – Lifetime of the excited state
- **`ion_number_density`** (*float*) – Number density of the dopant ions (1/m^3)
- **`background_loss`** (*float*) – Linear loss of the core (1/m, NOT in dB/m)
- **`core_na`** (*float*) – Numerical aperture of the core

`nonlinear_effective_area`(*freq*)

Returns the nonlinear effective area of the fundamental fiber mode with the given frequency. The method used is determined by the attribute self.effective_area_type.

Parameters **`freq`** (*float or numpy float array*) – The frequency of the optical signal (Hz).

Returns The nonlinear effective area

Return type Same as argument type.

`saturation_parameter`()

Returns the constant saturation parameter zeta defined in the Giles model.

3.5 Yb-doped double-clad fiber

class YbDopedDoubleCladFiber(*length*, *core_radius*, *ion_number_density*, *background_loss*, *core_na*, *ratio_of_core_and_cladding_diameters*)

YbDopedDoubleCladFiber is a convenience class for Yb-doped double-clad fiber that uses the default spectroscopic data for Yb-ions.

`__init__`(*length*, *core_radius*, *ion_number_density*, *background_loss*, *core_na*, *ratio_of_core_and_cladding_diameters*)

Parameters

- **`length`** (*float*) – Fiber length
- **`core_radius`** (*float*) – Core radius
- **`ion_number_density`** (*float*) – Number density of the dopant ions (1/m^3)
- **`background_loss`** (*float*) – Linear loss of the core (1/m, NOT in dB/m)
- **`core_na`** (*float*) – Numerical aperture of the core
- **`ratio_of_core_and_cladding_diameters`** (*float*) – Core diameter divided by cladding diameter

`core_area`()

Returns the core area of the fiber defined as pi*r**2, where r is the core radius.

Returns Core area

Return type float

classmethod from_cross_section_files(*length=0*, *absorption_cs_file=None*, *emission_cs_file=None*, *core_radius=0*, *upper_state_lifetime=0*, *ion_number_density=0*, *background_loss=0*, *core_na=0*, *ratio_of_core_and_cladding_diameters=0*)

Parameters

- **length** (*float*) – Fiber length
- **absorption_cs_file** (*str*) – Name of the file containing absorption cross-section data
- **emission_cs_file** (*str*) – Name of the file containing emission cross-section data
- **core_radius** (*float*) – Core radius
- **upper_state_lifetime** (*float*) – Lifetime of the excited state
- **ion_number_density** (*float*) – Number density of the dopant ions (1/m^3)
- **background_loss** (*float*) – Linear loss of the core (1/m, NOT in dB/m)
- **core_na** (*float*) – Numerical aperture of the core
- **ratio_of_core_and_cladding_diameters** (*float*) – Core diameter divided by cladding diameter

nonlinear_effective_area(*freq*)

Returns the nonlinear effective area of the fundamental fiber mode with the given frequency. The method used is determined by the attribute self.effective_area_type.

Parameters **freq** (*float or numpy float array*) – The frequency of the optical signal (Hz).

Returns The nonlinear effective area

Return type Same as argument type.

pump_cladding_radius()

Returns the radius of the fiber's pump cladding.

pump_to_core_overlap()

Returns the overlap between the core and the pump beams, which equals to the ratio of core and cladding area.

saturation_parameter()

Returns the constant saturation parameter zeta defined in the Giles model.

CHAPTER 4

Initial guess

class GuessParameters

GuessParameters defines the guessed gain and functional form of each channel in the simulation. See also docs for `ChannelGuessParameters`

class ChannelGuessParameters

ChannelGuessParameters defines the guessed gain and the functional form of the power evolution for each type of channel (signal, pump, ASE, and Raman). The gain can be defined directly or as the output power. The gain guess is stored as a function used to calculate the output power.

get_gain_shape()

Getter for the guessed shape of the function.

Returns The guessed functional form

Return type Member of the GainShapes Enum

get_output_power(*input_power*)

Getter for the guessed output power.

Returns The guessed output power

Return type float

set_gain_db(*gain_db*)

Set new guessed gain value. Overrides default and previously set gain and output power guesses.

Parameters **gain_db** (*float*) – Guessed total gain in dB

set_gain_shape(*gain_shape*)

Set new guessed shape of the power evolution. Overrides default or previously set values.

Parameters **gain_shape** (*Member of GainShapes Enum*) – New guess for the functional form of power evolution

set_output_power(*output_power*)

Set new guessed output power value. Overrides default and previously set gain and output power guesses.

Parameters **output_power** (*float*) – Guessed output power in W

class GainShapes

This Enum defines the possible functional forms used the construct the initial guess.

CHAPTER 5

Boundary conditions

class BasicBoundaryConditions(*channels*)

This class implements the most basic possible boundary conditions in the Giles model: all input powers should be those given to the model. Backward progapating beams have their inputs at the end.

CHAPTER 6

Simulation result

Finalizing the interface and the documentation is in progress!

CHAPTER 7

Helper functions

`load_spectrum` (*file_name*)
Loads a spectrum file with two columns of floats as numpy array. The first column is wavelength in nanometers; the second column is some spectroscopic property (mostly cross section) in SI units.

`load_two_column_file` (*file_name*)
Loads a file with two columns of floats as a numpy array.

`wl_bw_to_freq_bw` (*wl_bw*, *center_wl*)
Transforms a spectral bandwidth in wavelength centered at wavelength center_wl into a spectral bandwidth in frequency.

Parameters

- **`wl_bw`** – Wavelength bandwidth
- **`center_wl`** (*`float or numpy array of floats`*) – Central wavelength of the spectrum

Returns Frequency bandwidth

Return type float or numpy array

`wl_to_freq` (*wl*)
Transforms (vacuum) wavelength to frequency.

`freq_to_wl` (*f*)
Transforms frequency to (vacuum) wavelength.

`decibel_to_exp` (*x*)
Transforms a logarithmic quantity from dB/m to 1/m.

`exp_to_decibel` (*x*)
Transforms a logarithmic quantity from 1/m to dB/m.

`to_db` (*x*)
Transforms a quantity to decibels.

`to_dbm` (*power*)
Transforms a power in Watts to dBm.

`fundamental_mode_mfd_marcuse`(*wl*, *r*, *na*)

Calculates the mode field diameter of the fundamental mode with vacuum wavelength wl using Marcuse's equation.

Parameters

- **`wl`** (*`float`*) – Wavelength of the mode
- **`r`** (*`float`*) – Core radius
- **`na`** (*`float`*) – Core numerical aperture

Returns Mode field diameter of the fundamental mode

Return type float

`fundamental_mode_mfd_petermann_2`(*wl*, *r*, *na*)

Calculates the mode field diameter of the fundamental mode with vacuum wavelength wl using the Petermann II equation.

Parameters

- **`wl`** (*`float`*) – Wavelength of the mode
- **`r`** (*`float`*) – Core radius
- **`na`** (*`float`*) – Core numerical aperture

Returns Mode field diameter of the fundamental mode

Return type float

`fundamental_mode_radius_petermann_2`(*wl*, *r*, *na*)

Calculates the fundamental mode radius with vacuum wavelength wl using the Petermann II equation.

Parameters

- **`wl`** (*`float`*) – Wavelength of the mode
- **`r`** (*`float`*) – Core radius
- **`na`** (*`float`*) – Core numerical aperture

Returns Mode field radius of the fundamental mode

Return type float

`fiber_v_parameter`(*wl*, *r*, *na*)

Calculates the V-parameter or normalized frequency of a fiber mode with vacuum wavelength wl.

Parameters

- **`wl`** (*`float`*) – Wavelength of the mode
- **`r`** (*`float`*) – Core radius
- **`na`** (*`float`*) – Core numerical aperture

Returns V-parameter of the mode

Return type float

`zeta_from_fiber_parameters`(*core_radius*, *upper_state_lifetime*, *ion_number_density*)

Calculates the Giles modes saturation parameter zeta.

Parameters

- **`core_radius`** (*`float`*) – Core radius of the fiber

- **`upper_state_lifetime`** (*`float`*) – Lifetime of the excited state
- **`ion_number_density`** (*`float`*) – Number density of the dopant ions (1/m^3)

Returns Saturation parameter zeta

Return type float

`gaussian_peak_power`(*average_power*, *f_rep*, *fwhm_duration*)
Calculates the peak power of a Gaussian pulse.

Parameters

- **`average_power`** (*`float`*) – Average power of the pulse signal
- **`f_rep`** (*`float`*) – Repetition rate of the pulsed signal
- **`fwhm_duration`** (*`float`*) – FWHM duration of the Gaussian pulses

Returns Peak power of the pulses

Return type float

`resample_array`(*arr*, *N*)
Changes the width of an array to N columns by using linear interpolation to each row. :param arr: Array to be resized :type arr: 2D numpy array :param N: Number of columns in the resized array :type N: int :returns: The resized array with N colums. :rtype: 2D numpy array

`linspace_2d`(*start_vec*, *end_vec*, *length*)
Creates a numpy array with given start and end vectors as first and last columns and a total number of columns specified by "length". The middle columns are linearly interpolated.

Parameters

- **`start_vec`** (*`1D numpy array`*) – First column of the generated array
- **`end_vec`** (*`1D numpy array`*) – Last column of the generated array
- **`length`** – Total number of columns in the generated array

Returns Array interpolated between the start and end vectors

Return type 2D numpy array

`expspace_2d`(*start_vec*, *end_vec*, *length*)
Creates a numpy array with given start and end vectors as first and last columns and a total number of columns specified by "length". The middle columns are calculated by assuming exponential increase (or decrease).

Parameters

- **`start_vec`** (*`1D numpy array`*) – First column of the generated array
- **`end_vec`** (*`1D numpy array`*) – Last column of the generated array
- **`length`** – Total number of columns in the generated array

Returns Array interpolated between the start and end vectors

Return type 2D numpy array

`check_signal_reprate`(*f_rep*)
Emits a warning if the repetition rate of the signal is too low to be accurately modelled due to pulse-to-pulse gain variations.

Parameters **`f_rep`** (*`float`*) – Repetition frequency

`dynamic_time_coordinates`(*max_time_steps*, *z_nodes*, *fiber_length*, *dt='auto'*)
Returns the time coordinates used in the simulation. Useful for setting time-varying input powers.

Parameters

- **`max_time_steps`** – Number of time steps in the simulation
- **`fiber`** (*`Subclass of FiberBase`*) – The fiber used in the simulation
- **`z_nodes`** (*`int`*) – Number of spatial nodes used in the simulation.
- **`dt`** (*`float`*) – Time step size. The 'auto' option uses realistic time step calculated from the Courant condition based on the speed of light in glass and the spatial step size. Larger (and physically unrealistic) time steps can be used to drastically speed up the convergence of steady state simulations.

Returns Time coordinate array

Return type numpy float array

`averaged_value_of_finite_bandwidth_spectrum`(*center_frequency*, *frequency_bandwidth*, *spectrum_func*)

Function used to calculate the average gain or absorption cross section of a finite bandwidth channel.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index