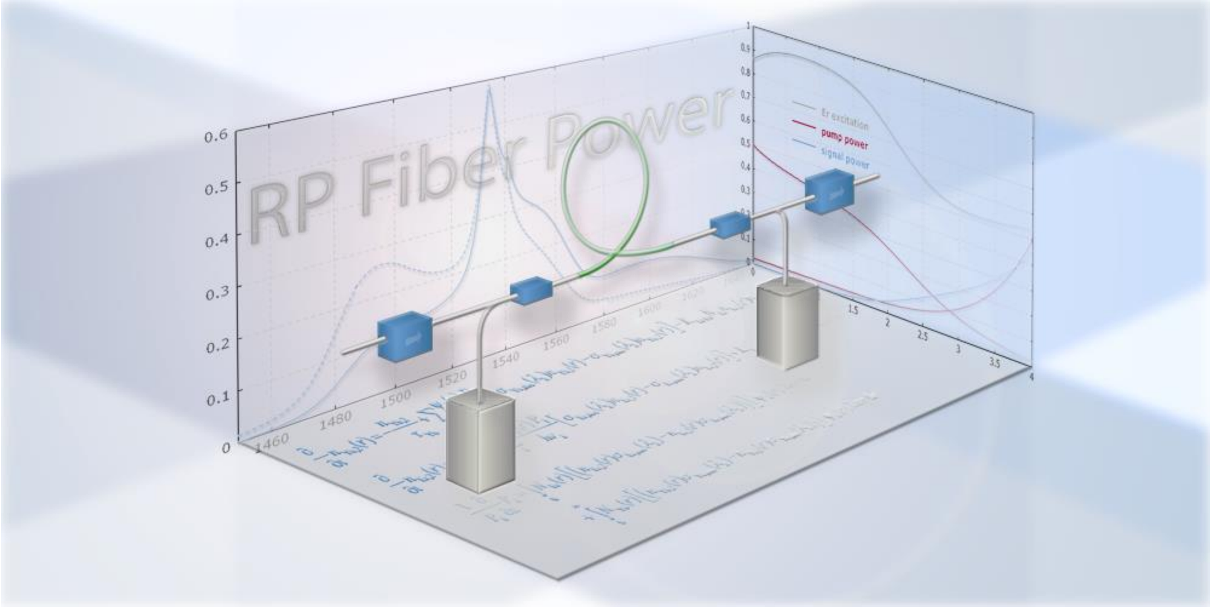
RP Photonics CONSULTING GmbH
www.rp-photonics.com

RP Fiber Power V5.0

a simulation and design software from
RP Photonics Consulting GmbH
www.rp-photonics.com



Version of this manual: 2014-07-09

RP Photonics Consulting GmbH, Dr. Rüdiger Paschotta, Waldstr. 17, 78073 Bad Dürrheim, Germany
www.rp-photonics.com, e-mail paschotta@rp-photonics.com, Tel. +49 7726 389 22 60, Fax +49 7726 389 22 62

Contents

1 General Information 5

1.1 What is "RP Fiber Power"? 5
1.2 Copyright Information and Disclaimer 5
1.3 Versions of RP Fiber Power 6
1.3.1 Passive Edition 6
1.3.2 Version History 6
1.3.3 Upgrades and Updates 6
1.4 Installation of RP Fiber Power 6
1.5 Technical Support 7
1.6 Documentation 7
1.7 Files belonging to RP Fiber Power 8

2 Principles of the Model 8

2.1 Basic Aspects of the Fiber; Numerical Step Sizes 9
2.1.1 Longitudinal Dimension 9
2.1.2 Transverse Dimensions: Rings and Azimuthal Segments 9
2.1.3 Transverse Dimensions: Rectangular Grid 10
2.2 Laser-active Ions 10
2.3 Optical Absorption and Amplification 12
2.4 Optical Channels 13
2.5 Reflections and Laser Resonators 14
2.6 Dynamic Simulations 15
2.7 The Mode Solver 16
2.8 Beam Propagation 16
2.8.1 General Assumptions 17
2.8.2 Used Algorithm 17
2.8.3 Beam Propagation Devices 18
2.8.4 The Numerical Grid 18
2.8.5 The Optical Channels 19
2.8.6 Laser Amplification 20
2.8.7 Kerr Nonlinearity 20
2.9 Ultrashort Pulse Propagation 20
2.9.1 Initial State of the Fiber 20
2.9.2 Numerical Representation of an Ultrashort Pulse 21
2.9.3 Physical Effects Acting on the Pulses 22
2.9.4 The Current Pulse 22
2.9.5 Pulses Propagation through a Fiber 23
2.9.6 Pulse Properties 23
2.10 General Assumptions for Power Calculations 23

3 Spectroscopic Data 25

4 The User Interface 26

4.1 General Principles 26
4.2 Command Line Parameters 26
4.3 The Main Form 27
4.4 Working with Interactive Forms 28
4.4.1 Calculating Fiber Modes 29
4.4.2 Beam Propagation 30
4.4.3 Defining the Parameters of an Active Fiber 31
4.4.4 Defining the Optical Channels 32
4.4.5 Additional Definitions 34
4.4.6 Defining the Graphical Output 35
4.4.7 Ultrashort pulses 36
4.5 The Menus 37
4.5.1 The File Menu 38
4.5.2 The Edit Menu 38

4.5.3 The View Menu ... 39
4.5.4 The Execute Menu ... 39
4.5.5 The Options Menu ... 39
4.5.6 The Help Menu ... 40
4.6 The Script Editors ... 40
4.7 The Log Area ... 41
4.8 The Output Area ... 41
4.9 The Graphics Windows ... 41
4.10 The Beam Profile Viewer ... 42
4.11 The Pulse Display Window ... 45

5 The Script Language of RP Fiber Power ... 48

5.1 General Remarks ... 48
5.2 General Rules for Scripts ... 48
5.3 Overview on the Definition of a Model ... 49
5.4 Definition of Fiber Parameters ... 50
5.5 Definition of Spectroscopic Data ... 51
5.5.1 Simple Level System ... 52
5.5.2 Extended Level System ... 53
5.5.3 Rules for Spectroscopic Data Files ... 54
5.6 Input and ASE Channels ... 55
5.6.1 Input Channels ... 56
5.6.2 ASE Channels ... 56
5.6.3 Reflections at the Fiber Ends ... 57
5.7 Localized Internal and External Losses ... 58
5.8 Retrieving Calculated Results ... 58
5.9 Modifying Model Parameters ... 59
5.10 Switching Between Different Devices ... 59
5.11 Calculation of the Noise Figure ... 60
5.12 Using the Mode Solver ... 60
5.13 Simulating Beam Propagation ... 63
5.14 Dynamic Simulations ... 70
5.15 Simulating Ultrashort Pulse Propagation ... 72
5.16 Graphical Output of Results ... 79
5.17 Display of Numerical Results ... 81
5.18 Output of Results to a File ... 82
5.19 Overview on the Arithmetics ... 82
5.19.1 General Properties ... 82
5.19.2 Arrays ... 82
5.19.3 Composite Expressions ... 82
5.19.4 Functions for Arithmetics ... 83
5.19.5 Numerical Integration ... 85
5.19.6 Numerical Root Finding ... 85
5.19.7 Fast Fourier Transforms (FFT) ... 85
5.19.8 Control Structures ... 86
5.19.9 Functions for the Handling of Strings ... 86
5.19.10 Formatting of Numerical and String Values ... 87
5.19.11 Functions for Graphics ... 88
5.19.12 Functions for Handling Files ... 89
5.19.13 Various Other Functions ... 90

6 Reference for Model-specific Functions ... 90

6.1.1 Functions for Defining or Modifying the Fiber Model ... 90
6.1.2 Functions for Obtaining Calculated Results ... 92
6.1.3 Functions for the Mode Solver ... 92
6.1.4 Functions for Numerical Beam Propagation ... 94
6.1.5 Functions Used Specifically for Dynamic Simulations ... 98
6.1.6 Functions Used for Ultrashort Pulse Simulations ... 98

7 How to Achieve Certain Things ... 105

7.1 How to Get Started with RP Fiber Power ... 105
7.2 How to Import Spectroscopic Data ... 105
7.3 How to Choose Appropriate Numerical Parameters ... 106
7.4 How to Export Calculated Data in Text Form ... 107
7.5 How to Trace Errors in a Script ... 108
7.6 How to Model Double-clad Fiber Devices ... 109
7.7 How to Model Mode Locking of a Fiber Laser ... 109

8 Demo Files ... 110

8.1 Calculating Fiber Modes ... 110
8.2 Modes of a Germanosilicate Fiber ... 111
8.3 Launching Light into a Multimode Fiber (Mode-based Simulation) ... 111
8.4 Launching Light into a Single-mode Fiber (with Beam Propagation) ... 111
8.5 Bent Large Mode Area Fiber ... 112
8.6 Tapered Fiber ... 112
8.7 Fiber Coupler ... 112
8.8 Pump Absorption in a Double-clad Fiber ... 113
8.9 Nonlinear Self-focusing ... 113
8.10 Simple Ytterbium-doped Amplifier ... 113
8.11 Ytterbium-doped Amplifier with Calculated Mode Profiles ... 114
8.12 Multimode Fiber Amplifier ... 114
8.13 Cladding-pumped Fiber Amplifier ... 114
8.14 Cladding-pumped Fiber Amplifier, with Calculated Cladding Modes ... 115
8.15 Fiber Amplifier with Dip in Doping Profile ... 115
8.16 Two-stage Fiber Amplifier Models ... 115
8.17 Fiber Amplifier with ASE ... 116
8.18 Fiber Amplifier with ASE (with html Output) ... 117
8.19 Ytterbium-doped Fiber Laser ... 117
8.20 Ytterbium-doped Fiber Laser, Automatic Wavelength Calculation ... 117
8.21 Erbium-doped Fiber Amplifier with ASE ... 117
8.22 Erbium-doped Fiber Amplifier with Quenching ... 118
8.23 Erbium-doped Fiber Amplifier for Multiple Signals ... 118
8.24 Erbium-ytterbium-doped Fiber Laser ... 118
8.25 Thulium-doped Upconversion Fiber Laser ... 118
8.26 Dynamic Amplifier Simulation ... 119
8.27 ASE Source with Automatic Spectral Optimization ... 119
8.28 Actively Q-switched Fiber Laser ... 119
8.29 Passively Q-switched Fiber Laser ... 120
8.30 Actively Q-switched Nd:YAG Laser ... 120
8.31 Actively Q-switched Nd:YAG Laser, with Beam Propagation ... 121
8.32 Passively Q-switched Nd:YAG Laser ... 121
8.33 Higher-order Soliton Pulses ... 122
8.34 Soliton Self-Frequency Shift ... 122
8.35 Parabolic Pulses in an Ytterbium-doped Fiber Amplifier ... 123
8.36 Stimulated Raman Scattering in Yb-doped Fiber Amplifier ... 123
8.37 Nonlinear Loop Mirror ... 124
8.38 Regenerative Amplifier ... 124
8.39 Mode-locked Fiber Lasers ... 125
8.40 Chirped-pulse Amplifier System ... 126
8.41 Soliton Self-frequency Shift ... 126
8.42 Supercontinuum Generation ... 126
8.43 Transmission of Telecom Signal ... 127
8.44 Fiber Amplifier with ASE, Output to html File ... 127

1 General Information

1.1 What is “RP Fiber Power”?

RP Fiber Power is a simulation and design software for passive fibers, fiber amplifiers, fiber lasers, and ASE sources. It has the following basic features:

- It can calculate various properties of all guided modes (LP modes) of weakly guiding fibers with radially symmetric refractive index profiles. The calculated properties include the transverse intensity distribution, the effective mode area, the effective refractive index and the group index, and the group velocity dispersion.
- It can model the behavior of laser-active ions with complicated user-defined level schemes, and it can include effects like spontaneous and stimulated emission, nonradiative transitions, and energy transfer processes.
- It can handle an arbitrary number of pump and signal inputs and also deals with amplified spontaneous emission (ASE). The full radial dependence of doping profiles and optical intensities is included.
- Noise properties can be calculated from the ASE powers.
- The software can calculate self-consistent steady-state solutions for the optical powers and electronic excitations, and can also simulate the temporal evolution. Such dynamic simulations can be done both for amplifiers (e.g., for single-pass pulse amplification) and for lasers (e.g. when being Q-switched).
- The propagation of ultrashort pulses can be simulated. Wavelength-dependent amplification is taken into account, also nonlinear and dispersive effects.
- The user interface (section 3) is both handy for a quick start and very flexible. The software is controlled via a powerful script language, and simple input forms allow to automatically generate the required scripts in most cases.

The **passive edition** of this software can calculate mode properties and simulate beam propagation, but lacks all features related to optical power calculations. Note that this documentation describes *all features,* even if you only have the passive edition of the software. The same holds if you obtained an update to an earlier version of the software.

The program **RP Fiber Power** was derived from **JPLOT**, from which it inherited all general features: much of the user interface, the input commands, and the arithmetics. In addition to **JPLOT** commands and its general arithmetic features, **RP Fiber Power** understands various functions for the definition of fiber parameters and pump, signal or ASE channels, and functions for retrieving calculated results, which are described in detail in this document. This manual covers only those **JPLOT** features which are typically relevant for use in **RP Fiber Power**, but does not give a complete description of **JPLOT**’s script language. The context-sensitive online help system, however, does cover all these details.

1.2 Copyright Information and Disclaimer

The program **RP Fiber Power** underlies the copyright of Dr. Rüdiger Paschotta, RP Photonics Consulting GmbH, who is the only author. Non-exclusive licenses for using this software are sold by RP Photonics Consulting GmbH. Please check and observe the license conditions when using this software.

RP Photonics Consulting GmbH or the author cannot accept any legal responsibility for adverse effects caused by using this program. As any other non-trivial program, it may contain bugs which might lead to wrong results in some cases. Bug reports are welcome, as well as other suggestions for further improvements.

1.3 Versions of RP Fiber Power

1.3.1 Passive Edition

Customers buying RP Fiber Power will obtain the latest software version – usually with all features included. However, it is possible also to obtain the **passive edition**, which is suitable for users of passive fibers only. It contains the mode solver and can simulate ultrashort pulse propagation, but it lacks the features related to laser amplification in active fibers.

1.3.2 Version History

So far, the following versions of RP Fiber Power have been develped:

- Version 1 was the original version. It had script editors only, i.e., the users had to develop scripts.
- Version 2 introduced the interactive forms. Except for sophisticated simulations, one could now work simply by filling out forms, and the script code was generated automatically. Also, dynamic simulations were introduced, i.e., power simulations with time-dependent pump and signal powers.
- Version 3 introduced the mode solver, calculating the properties of fiber modes from a given refractive index profile.
- Version 4 introduced ultrashort pulse propagation. Also, the user interface was revised: the originally separate interactive forms have been integrated into the main form, and the script editors are now selected via tabs. One can switch between the forms mode and the script editor mode.
- Version 5 provided means to simulate beam propagation in fibers and other waveguides.

1.3.3 Upgrades and Updates

Customers having a user license for an older version can buy an upgrade. They will then obtain the latest version of the software. The upgrade price is approximately the difference of the license prices.

Within one version, updates can be obtained, which bring some improvements and bug fixes. Such updates are free.

If a customer obtains an update but does not want to upgrade to the latest version, he obtains a version which looks like the latest version but does not contain certain new features. For example, the software may then have interactive fields for giving inputs to ultrashort pulse propagation, but the actual code for doing such simulations is not contained.

Note that the documentation (manual and online help) always describes all features, even if you only have the passive edition or an update on an older version of the software.

1.4 Installation of RP Fiber Power

The installation of this software works as follows:

- If you obtained the software in a ZIP file, unpack it to obtain the installation program **RP Fiber Power setup.exe**.

- Make sure that you have sufficient user rights under Windows to install programs, e.g. by logging in as the system administrator. (Once the software is installed, you may work as a user with limited rights, even as a guest user.)
- Start the installation program and go through the usual dialogs. You may e.g. modify the folder where the software is installed, but this is normally not necessary.

Now the software is installed. Your Windows system now knows that files with the filename extension **`.fpw`** can be opened with **RP Fiber Power**. This means that you can double-click on any **`.fpw`** file in the Windows Explorer to start the software. Alternatively, you can use the Windows Start menu. However, it is preferable to store your scripts in project-specific folders, and to start the software with a double click on such a file. Note that the software stores information on open files, form settings etc. in the corresponding work folder, and it is convenient to keep these settings separate for different projects by using different folders.

When you first start the software, it needs to be activated. The software displays a site code and a machine code which you have to send via e-mail to **support@rp-photonics.com**. You will then receive an activation code, which activates the software for use on that computer. You have to do this procedure for every computer on which you use the software, and of course need a license for every computer. (Licenses for use on additional computers within the same institution are far cheaper than the first license.)

In addition to the installation program, the installation package also contains the software manual in PDF format and a folder named **`Demo`** with some demo files. You may copy this folder to your working folder and execute some of these files to get acquainted with the software.

The system requirements of this software are moderate:

- One requires a personal computer running under Microsoft Windows. The Windows version should not be older than Windows 2000.
- The hardware requirements concerning main memory and disk space are moderate – not more than few tens of megabytes of harddisk space are required.

1.5 Technical Support

According to your license contract, some number of hours of technical support are included in the license fee. Before you call support, please coordinate that within your institution.

You may send an e-mail with the description of your problem to support@rp-photonics.com (see also: **Help | Technical support**). In many cases, it may be helpful to send a script file which does not work as expected. RP Photonics may then simply do some corrections in that file, or send you a few lines of additional script code.

You may also obtain support via telephone at the number +41 44 201 02 60. Note the central European time zone, but American customers have a fair chance to reach RP Photonics also at relatively late times.

You may also have a look at the RP Photonics website (http://www.rp-photonics.com/), in particular at the **Encyclopedia of Laser Physics and Technology** (http://www.rp-photonics.com/encyclopedia.html).

1.6 Documentation

The documentation of **RP Fiber Power** consists of two parts:

- a PDF document (which you are reading now),
- the context-sensitive online help system,

For using this software, you will mainly require sections 2 and 5 of this document. The former describes the basic handling, while the latter explains in detail the calculations which are specific to fiber amplifier and laser simulations. See also the how..to section 6.

Also have a look at the demo files (section 7.7). They are well documented, so it worth studying them in some detail. Also, for setting up a new model it is often the easiest way to start with a copy of a demo file.

1.7 Files belonging to RP Fiber Power

The following files belong to the software:

- **`RP_Fiber_Power.exe`** is the executable program file.
- **`RP_Fiber_Power.ico`** contains the program icon.
- **`pcgint.dll`** and **`pcgw32.dll`** belong to the software activation system.
- **`RP Fiber Power.pdf`** contains this documentation.
- **`RP Fiber Power.chm`** contains the context-sensitive help texts.
- **`Units.inc`** contains the definitions for various units, and is often used as an include file. For example, the variable **`nm`** is defined as 1e–9. This allows you to specify a wavelength as **`1550 nm`**, for example, instead of **`1550e-9`**.
- **`License.txt`** informs you about the legal rights which you have as a user.

You also obtain the folder **`Demo`** (placed in the program folder), containing some files (described in section 7.7) which demonstrate the most important features of the program. It is recommended to copy files from there to your user files folder and leave the originals unmodified. When **RP Fiber Power** is installed, double-click on any of those files to execute them.

The following files are generated by **RP Fiber Power** in your user folder:

- **`*.fpw`** are the **RP Fiber Power** scripts, which you edit to control the calculations and the outputs.
- **`*.fpi`** files contain the settings of the interactive forms, as saved from those. One can load them into the interactive forms and execute them to obtain a script and the calculated results. On the other hand, it *not* possible to get the form settings from a script file; therefore, it is advisable to save the contents of the interactive forms to an **`.fpi`** file if you may want to continue with these settings at some later time, after having modified the settings in the forms.
- **`RP_Fiber_Power-W.dsk`** is generated in each folder where **RP Fiber Power** is called. It stores information about open editor files and also the contents of the interactive forms.

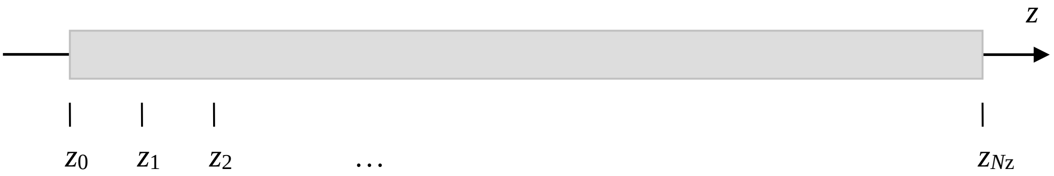
2 Principles of the Model

This section explains the basic concepts of the underlying physical model, and how the data are organized.

2.1 Basic Aspects of the Fiber; Numerical Step Sizes

2.1.1 Longitudinal Dimension

The model always considers a single active fiber with the length L. The position along the fiber is characterized with a coordinate z, varying from 0 to L. The user defines some number of steps N_z, and numerically the fiber is represented by N_z sections of length $\Delta L = L / N_z$. The software calculates the optical powers and the electronic level populations for the position $z_j = j \cdot \Delta L$, where the index j varies from 0 to N_z.



When deciding for the number of numerical steps in z direction, the following should be considered:

- For obtaining a high accuracy, the step size has to be so small that each of the involved optical channels experiences a gain or loss of below 0.5 dB in each step. For example, for a fiber amplifier with a signal gain of 30 dB, one should at least 60 steps, or better 100 steps for good accuracy. (Some reserve is recommended, since the gain may be larger in some sections of the fiber.)
- To be sure, check whether the results change substantially when you double the number of steps.
- The required computation time will rise with the number of steps. In most cases, computation time is not a critical aspect. However, it may be worth trying with a smaller number of steps in cases where a high transverse resolution (see below) and/or a large number of optical channels is required.

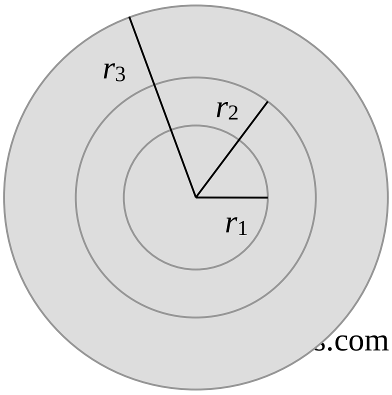
Any z-dependent calculated quantities will be linearly interpolated between the grid points. If these step-wise linear characteristics become apparent in some generated diagrams, this is an indication that the chosen number of steps may be too small.

A general assumption of the model is that the doping concentrations are constant in z direction.

Note that a different longitudinal step size may be chosen for simulating ultrashort pulse propagation (see section 2.9.4).

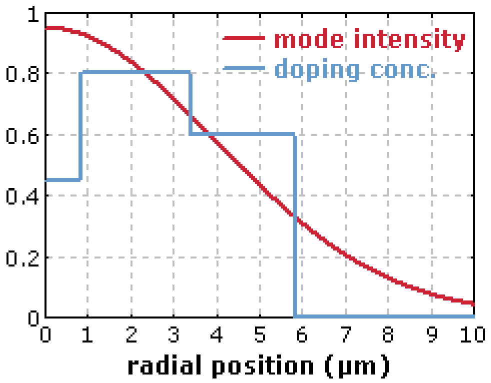
2.1.2 Transverse Dimensions: Rings and Azimuthal Segments

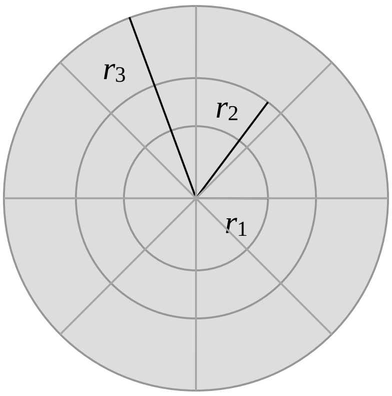
The model can also include the transverse dependencies of doping profiles and optical intensities. Here, the user can define one or several rings with increasing radii. The rings are numbered with an index k, ranging from 1 to the maximum value N_r. The figure on the right side shows an example with 3 rings.



The first “ring” with radius r_1 is actually a circle. The second ring ranges from r_1 to r_2, and the third one from r_2 to r_3. Together with each ring, the user defines the doping concentration of a first type of laser-active ion. It is possible later on to define doping concentrations for other ions in all rings.

Within each ring, a constant doping concentration is assumed. Also, average values for the optical intensities are used. These are calculated from the given intensity profiles, using multiple values spread over a ring. To obtain a finer radial resolution, one may further subdivide some rings, even if some of the rings may have identical doping concentrations.

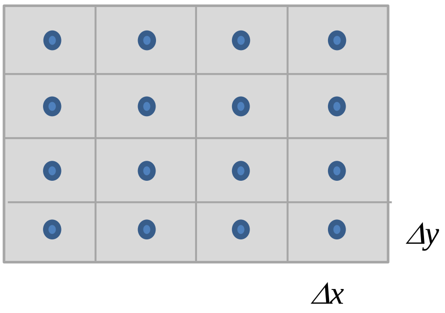




The diagram above shows an example for a doping profile, again with three “rings”, and the mode intensity distribution for some optical channel (see section 2.4).

It is also possible to introduce an azimuthal dependence of the field intensities. (This can be relevant for higher-order fiber modes.) The user can define some number of segments per ring. This may be chosen to be 4 or 8, for example, for modeling the amplification of LP$_{10}$ and LP$_{11}$ modes. The figure above (right side) shows a division with 8 azimuthal segments per ring.

2.1.3 Transverse Dimensions: Rectangular Grid



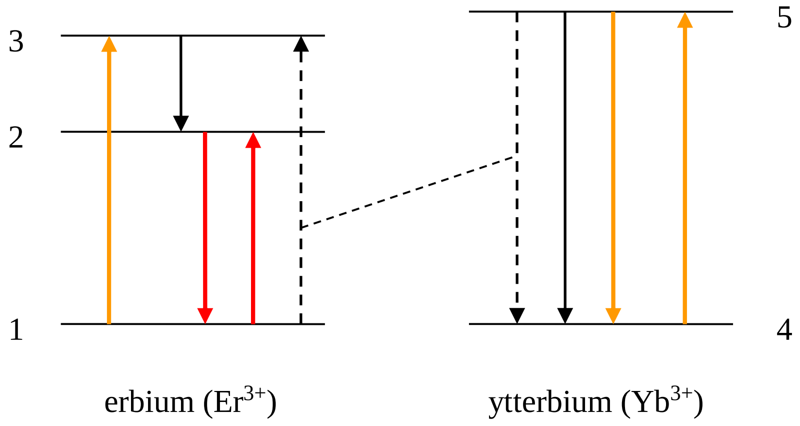
For some applications (for example, in planar waveguides with no radial symmetry), it is preferable to use a rectangular grid. This is also possible with the software. In that case, the grid spans some range of *x* and *y* values, where the *x* and *y* ranges can be different and can have different step sizes. Each x/y pair represents a rectangular segment and sits in its center.

2.2 Laser-active Ions

The model can consider a single type of laser-active ions, or several such ion types.

Any ion is assumed to have a number of energy levels. These levels actually represent whole Stark level manifolds. It would not be sensible to distinguish single Stark levels within such manifolds, because phonons of the crystal lattice induce very fast transitions between the sublevels, and it is very hard to obtain spectroscopic data for single sublevels. Instead, wavelength-dependent effective transition cross sections are used, which take into account a weighted average of transitions between all sublevels.

The electronic levels of all ion types are numbered with a single scheme. As an example, for a fiber doped with both erbium and ytterbium, we consider three energy levels of erbium and two levels of ytterbium. Here, we have levels 1 to 3 for erbium and 4 to 5 for ytterbium:



Various transitions are also shown in the figure above. Stimulated transitions (induced by light) are indicated with thicker arrows, while spontaneous transitions are marked with thinner arrows. In more detail, the transitions are the following:

- Some pump wave with a wavelength of e.g. 980 nm can excite erbium ions from level 1 to 3. Stimulated emission in the opposite direction is neglected (but could be implemented if required).
- There is a fast non-radiative transition from level 3 to 2. This is not related to optical influences or to emission of light, but rather associated with multi-phonon emission.
- There is the laser (or amplifier) transition from level 2 to 1 with emission in the 1.5-μm spectral region. Also, there light can be reabsorbed, exciting ions from level 1 to 2.
- Ytterbium ions can also be pumped from level 4 to 5 or down again via stimulated or spontaneous emission.
- Finally, there is the possibility of an energy transfer, where an ytterbium ion goes from level 5 to 4, while the energy is used to pump an erbium ion from level 1 to 3.

Apart from the ground states, only metastable levels are considered in the model. For example, if the non-radiative transfer from level 3 to 2 in erbium were assumed to be infinitely fast, level 3 would be eliminated. The pumping of erbium around 980 nm would then directly lead into level 2, and the same applies to the energy transfer process.

The fractional level populations are denoted as n_1 to n_5, and of course depend on the position in the fiber. The sum of all level populations of a certain ion type is always 1. In the example above, the ground state of the system corresponds to $n_1 = n_4 = 1$, $n_2 = n_3 = n_5 = 0$.

For the calculation of the populations of electronic levels and also of optical absorption or amplification, the following user-defined parameters are used:

- For spontaneous transitions (radiative or non-radiative), where the transition rate is proportional to the population of the starting level, the user specifies the transition rate per ion in the starting level. For example, for the upper laser level, this is the inverse upper-state lifetime.

- For optically induced transitions (absorption or stimulated emission), wavelength-dependent effective transition cross sections are used. The software then automatically determines which of the optical channels (see below) are relevant for some transition.
- There can be quenching processes, where the transition rate from some starting level to some end level is proportional to the square or the cube of the population of the starting level. The user defines the proportionality constant.
- For energy transfers, the user specifies two starting levels (which may be identical) and two end levels, and a proportionality factor.

In many cases, it is sufficient to use a simplified gain model (see section 5.5.1) with the following features:

- Apart from the ground state, there is only a single metastable excited level, having a certain upper-state lifetime.
- Any photon absorption brings ions into that excited level, and spontaneous or stimulated emission brings them back to the ground state.

That model can often be applied even if there are multiple excited levels, provided that only one of them is metastable, and the other ones are very short-lived. For example, pumping may occur into a short-lived level 3, from which ions are quickly transferred into the metastable level 2 by multi-phonon emission. This is the case, for example, for pumping erbium in the 980-nm spectral region. It is only that additional complications such as upconversion can not be considered within that simplified kind of model.

2.3 Optical Absorption and Amplification

From the data discussed in the previous subsection, the software can calculate the optical absorption or gain at a certain position for a given wavelength λ. Taking the erbium-ytterbium system discussed above as an example, and assuming constant doping concentration and mode intensity in the transverse direction, the local gain coefficient in the 1.5-µm region is

$$g(z,\lambda) = N_{\mathrm{Er}}\left(n_2(z)\sigma_{21}(\lambda) - n_1(z)\sigma_{12}(\lambda)\right), \tag{1}$$

and the total amplification factor for the fiber length L is $\exp\left(\int_0^L g(z,\lambda)\,\mathrm{d}z\right)$. If the transverse dimensions are taken into account, the gain coefficient has contributions from different positions:

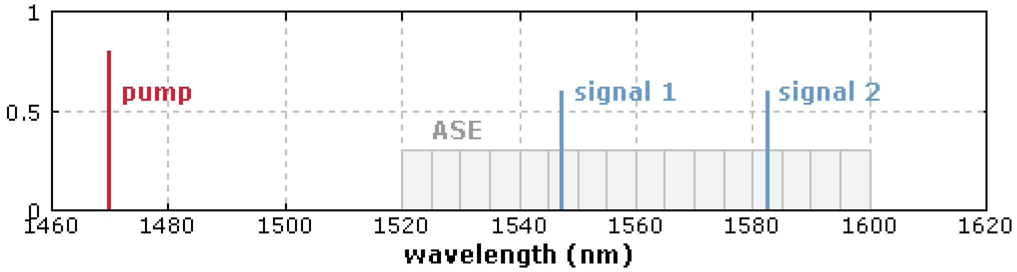
$$g(z,\lambda) = \int_{r=0}^{\infty}\int_{\varphi=0}^{2\pi} N_{\mathrm{Er}}(r)\left[n_2(z,r)\sigma_{21}(\lambda) - n_1(z,r)\sigma_{12}(\lambda)\right]\psi(r,\varphi,\lambda)\; r\,\mathrm{d}r\,\mathrm{d}\varphi\,, \tag{2}$$

where $\psi(r,\varphi,\lambda)$ defines the transverse intensity distribution of the mode considered, which is normalized so that the integral of $\psi(r,\varphi,\lambda)$ in the transverse dimension is 1. In the numerical realization, the integral is replaced with a sum of contributions from the different rings.

If there are several combinations of levels interacting with a certain optical wavelength, all contributions to the gain or loss are added.

2.4 Optical Channels

The software allows the user to define some number of optical "channels", representing the light propagation in the fiber. The figure below illustrates the optical channels in an erbium-doped fiber amplifier: a pump channel, two signal channels, and 16 ASE channels:



An optical channel has the following properties:

- It has some wavelength λ, and in case of an ASE channel (for calculating amplified spontaneous emission), it also has a wavelength bandwidth $\Delta\lambda$, and in addition the user specifies the number of propagation modes. The latter can be 2 for a single-mode fiber, when both polarization directions are accounted for.
- It has a propagation direction, which is forward ($z = 0 \to L$) or backward ($z = L \to 0$).
- There can be some background loss, specified in dB/m, resulting e.g. from absorption by impurity or from scattering at the core–cladding interface.
- There can be reflectivities for any channel at both fiber ends. Such reflectivities are particularly used for modeling fiber lasers. This is explained in section 2.5.
- There can be additional parasitic losses at both ends, inside and/or outside the reflector.
- There is a transverse intensity distribution, characterized with a function $\psi(r)$ or $\psi(r,\varphi)$ (which is automatically normalized). For single-mode fibers, $\psi(r)$ can normally be well approximated with a Gaussian mode function:

$$\psi(r) = \exp\left(-\left(\frac{r}{w}\right)^2\right) \quad (3)$$

with the mode radius w. For a step-index fiber with core radius a and numerical aperture NA, the mode radius can be estimated with the Marcuse formula:

$$\frac{w}{a} \approx 0.65 + \frac{1.619}{V^{3/2}} + \frac{2.879}{V^6} \quad (4)$$

where the V number is

$$V = \frac{2\pi}{\lambda} a \text{ NA} = \frac{2\pi}{\lambda} a \sqrt{n_{\text{core}}{}^2 - n_{\text{cladding}}{}^2} . \quad (5)$$

(Note that one may use the Marcuse formula in a script in order to automatically calculate the beam radius from given step-index fiber parameters.)

For double-clad fibers, the multimode pump wave is usually approximated as a top-hat function, the radius of which is identical with the radius of the pump cladding.

Note that the algorithm of the software is based on the assumption that the shape of the transverse intensity distribution of each channel remains constant during propagation. This assumption is usually very well fulfilled for single-mode fibers, ore more generally if an optical channel represents just one mode. For multimode fiber supporting only few guided modes, one optical channel for each mode can be defined, so that the assumption remains valid; however, mode coupling cannot be included. For multimode propagation with a large number of modes, a single channel with a top-hat intensity profile can be a reasonable approximation. For the pump wave in a double-clad fiber (see section 7.6), this approximation may be violated if there is only weak mode mixing: the pump intensity distribution may acquire a “hole” in the region of the core, where pump light is absorbed. This effect is often suppressed to some extent, e.g. by using an off-centered fiber core or a D-shaped pump cladding.

- Finally, each channel can have some input power, which is injected from outside the fiber. This can e.g. be a pump beam or an input signal beam. If there are any reflections at the fiber ends, these inputs are of course attenuated and mixed with the power from a channel with the opposite propagation direction.

For amplified spontaneous emission, one normally uses some array of wavelength channels, e.g. ranging from 1500 to 1600 nm in steps of 5 nm. A finer channel spacing improves the spectral resolution of the results, but it also increases the computation time.

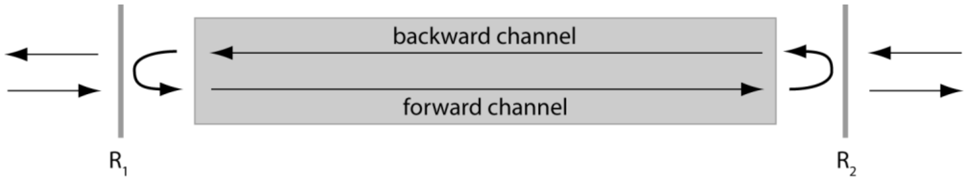
Note that all optical channels must be defined by the user. The software does not automatically create any channels. This means, e.g., that some ASE problem may be overlooked if the user fails to define ASE channel at all wavelengths with substantial optical amplification.

2.5 Reflections and Laser Resonators

In a fiber amplifier, a signal or pump wave usually travels through the fiber in only one direction, but it is also possible to a reflection at one end, so that such a wave can make a double pass. If such reflections for a signal occur on both ends, laser action may occur, where the signal can be maintained in such a laser resonator even without an external input.

RP Fiber Power allows one to simulate two different configurations:

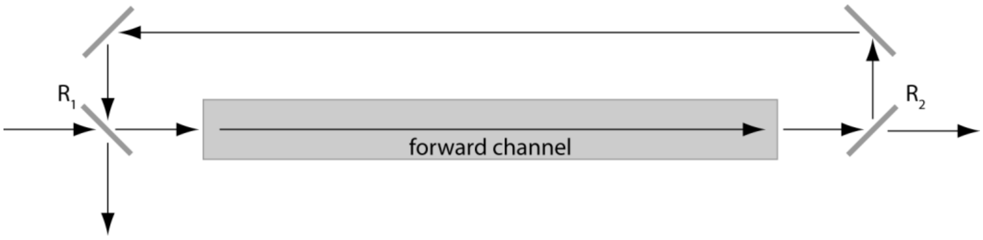
- The default configuration is a linear one, where the reflectivities couple an optical channel (see section 2.4) to another channel with the opposite propagation direction:



For the forward-propagating channel, the reflectivity R_1 provides an input from the corresponding backward-propagating channel, and the reflectivity R_2 couples power from its end to the input of the corresponding backward-propagating channel.

If there is a specified input power for the channel, this is also attenuated by the reflection, i.e., its power is multiplied with $1-R_1$.

- In a ring configuration, the power at the end of a channel is multiplied with $R_1 \cdot R_2$ and provides an input for the *same* channel.



There can be counterpropagating channels in a ring configuration, but these are *not* coupled to each other by the reflectivities, but rather stay independent.

Note that coherent effects are not taken into account, as the software only calculates optical powers, but not optical phases. For example, the beam going downward from the mirror with reflectivity R_1 in the previous figure contains contributions from a reflected input (coming from the left side) and from the transmitted internal beam. The software simply adds the powers of both contributions, not considering possible interference effects for narrowband beams.

For ultrashort pulse simulations (section 5.15), the end reflectivities are ignored.

2.6 Dynamic Simulations

RP Fiber Power can also simulate the temporal evolution of an active fiber system. In that case, time-dependent input powers for one or several optical channels are defined, and some time interval and a temporal step size are specified. It is also possible to introduce a time dependence of the resonator losses. The software then simulates the temporal evolution of the output powers of all channels and of the excitation densities. For example, these features may be used to address the following kinds of situations:

- When short and intense pulses are amplified in a fiber amplifier, the gain saturates during amplification of the pulse. As a result, the pulse shape may be substantially distorted. The software can be used calculate such distortions and to investigate their dependence on various parameters. Of course, it is also possible calculate how much of the stored energy can be extracted by a pulse.

- A fiber laser may be actively Q-switched by incorporating some kind of optical element with time-dependent power attenuation. Comparing with the situation in a solid-state bulk laser, the pulse evolution in a fiber laser can be more complicated due to the much higher round-trip gain. This is the case particularly when using a fast modulator, where the switching time is well below the round-trip time. **RP Fiber Power** can fully simulate such pulse evolution, even in situations where conventional Q-switching software would not be accurate.

Dynamic simulations always start with some initial state of the system, which can either be the steady state calculated for some input powers or the final state of a previous dynamic simulation.

There are two different modes of dynamic simulations:

- The most precise, but also the most time-consuming mode takes into account the propagation times. Here, the dynamical variables are the excitation densities along the fiber and the optical powers at all locations. The temporal step size is determined by the time that light requires to propagate through one numerical fiber section (see section 2.1). This kind of model is suitable for simulating the performance of Q-switched lasers, or for spiking phenomena as occur when the pump power of a laser is switched on. However, it leads to long computation times when investigating phenomena on a time scale far longer than the round-trip time.

- A more simplified mode allows for a much higher speed, e.g. for simulating pulse amplification in a fiber amplifier. Here, the time delays associated with light propagation within the fiber are neglected. This is no problem when the signal light passes an amplifier just once (or possibly twice), because it does not matter that different parts of the fiber see the pulse at different times. The temporal step size can then be much larger (specified by the user), so that one does not need excessive computation time for investigating phenomena on a time scale far longer than the round-trip time. The optical powers for some moment of time can simply be calculated from the input powers and the excitation of the fiber.

For simulating Q-switched lasers, it can be appropriate to combine both methods. The precise method has to be used for some time interval after opening the Q-switch. However, for a period of time where only the gain medium is pumped, the simplified method may be employed, using a much longer temporal step size. In that way, it is possible to simulate the evolution of many subsequent pulses while keeping the computation time very low.

Note that some dynamical aspects of fiber lasers, such as certain spiking phenomena observed during continuous-wave operation, appear to be critically dependent on certain nonlinear effects, and are not yet well understood. Also, mode beating effects and the like, leading to modulations on the time scale of the round-trip time, are not included. Such effects can not be modeled with this software. Any software simulating such effects would generally require a lot more input data which would be very difficult to provide.

Section 5.13 explains how to implement dynamic simulations using the script language.

Note that there are separate features for simulation the propagation of ultrashort pulses, as explained in the following section. The dynamic simulations are based on the assumption that the pulse bandwidth is small, and cannot take into account nonlinearities. On the other hand, they can be used to simulate the interaction of multiple optical channels, which is not possible with the ultrashort pulse propagation feature.

2.7 The Mode Solver

The software also contains a mode solver, which can calculate the properties of all guided modes (no cladding modes and leaky modes) of the fiber, based on a given refractive index profile. The index profile needs to be real and radially symmetric, but an arbitrary radial dependence can be given. The mode solver provides functions to access the mode properties, in particular their intensity profiles. These can be used for the optical channels.

The mode solver works only up to a certain maximum number of modes, which depends on the type of refractive index profile. For typical index profiles, more than 1000 modes are possible.

See section 5.12 for details on the use of the mode solver.

2.8 Beam Propagation

Since version 5, **RP Fiber Power** provides means to simulate beam propagation in fibers – i.e., not only propagation of optical powers, but the propagation of full complex amplitude profiles under the influence of arbitrary (but low-contrast) refractive index profiles, laser gain, Kerr effect, etc.

In the following, the principles of the used model are explained. For the details, such as the functions to be used to set up and utilize a model, see section 5.13.

2.8.1 General Assumptions

As the simulation of beam propagation is computationally quite demanding, it is subject to several restrictions:

- All involved waves must propagate essentially in one direction (called the *z* direction). Their divergence angles must be well below 1 rad, as the paraxial approximation is used. Back-reflections leading to counterpropagating waves (at index discontinuities, for example) cannot be modeled.
- The waves interact only in cases with laser gain, e.g. via gain saturation.
- Only one polarization direction is considered, i.e., a scalar (rather than vectorial) model is used.
- The number of grid points (see below) must be limited due to memory constraints. However, one may do sub-steps without saving the local amplitudes, and can in that way do the modeling for very large grids.
- The computation time may be rather long for very large grids, as required e.g. for a long fiber with high numerical aperture.
- The beam propagation simulations cannot be directly combined with laser and amplifier models with counterpropagating waves, and with ultrashort pulse propagation. However, one may of course exchange data – for example, use beam profiles from beam propagation simulations as inputs for fiber amplifier models.

2.8.2 Used Algorithm

There are various quite different algorithms for simulating beam propagation. **RP Fiber Power** uses a split-step-Fourier algorithm. Here, the optical fields are propagated forward in *z* direction as follows:

- First, the field is transformed to the spatial Fourier domain. Here, phase factors are applied which take into account diffraction effects. (A wave vector according to a mean refractive index is assumed.) Then, the fields are transformed back to the spatial domain.
- Thereafter, phase factors are applied in the spatial domain, which take into account the refractive index inhomogeneities (e.g., the index increase of a fiber core).
- Finally, laser gain (or absorption) is applied in cases with laser-active ions, and nonlinear changes are applied if the fiber is nonlinear. All this is also done in the spatial domain.

The accuracy of this method is good provided that the *z* steps are small enough, as in reality the effects of diffraction and spatial phase changes are continuously distributed in space. For fibers and other waveguides with small refractive index contrast, the *z* steps can be made relatively large – in many cases, much larger than the wavelength. More details on the required step size are given below.

The used algorithm naturally implies periodic boundary conditions: a wave hitting the edge of the numerical grid can reenter the grid on the opposite side. However, that behavior can be changed (see below).

2.8.3 Beam Propagation Devices

The simulation of beam propagation in **RP Fiber Power** is done using “beam propagation devices”, which are kept separate from the fiber models for propagation of optical powers only: they have their own numerical grids, optical channels, spectroscopic data, etc.

One can define multiple beam propagation devices in one script. One may, for example, use them for different versions of a device, for multiple devices used in series, or for simulations with different resolutions for sanity checking.

Each device has its own numerical grid. Field distributions can be transferred from one device to another one – if necessary, using interpolation.

2.8.4 The Numerical Grid

Each beam propagation device has a numerical grid of rectangular type: it spans some ranges in *x* and *y* directions, which are both centered around the origin of the coordinate system: *x* runs in some range from $-x_{\max}$ to $+x_{\max}$, and *y* from $-y_{\max}$ to $+y_{\max}$. The limits for *x* and *y* can be chosen independently, and the step sizes are determined as $\delta x = 2x_{\max} / N_x$ and $\delta y = 2y_{\max} / N_y$, with some integer numbers N_x and N_y, which have to be powers of 2.

Also, the grid spans a certain *z* range, where *z* = 0 corresponds to the beginning of the waveguide (where initial fields are injected) and the waves propagate in positive *z* direction to some value $z_{\max}$ with a step size $\delta z = z_{\max} / N_z$.

The **transverse grid resolution** (i.e., in *x* and *y* direction) has to be high enough to properly sample the transverse variations of the fields. How fast these variations can be for guided modes, depends on the numerical aperture of the fiber. For a step-index fiber, for example, the maximum beam angle against the fiber axis (within the fiber) is of the order of $\theta_{\max} = \mathrm{NA} / n_{\mathrm{core}}$ (where NA is the numerical aperture of the fiber). That corresponds to a maximum transverse wave vector component of $\approx k \sin\theta_{\max} \approx k\, \theta_{\max} = k\, \mathrm{NA} / n = (2\pi / \lambda)\mathrm{NA}$ where λ is the vacuum wavelength. From that we can conclude that a transverse resolution of the order of $\lambda / (2\pi \mathrm{NA})$ should be sufficient, as far as only guided modes are of interest (but not if cladding modes also need to be modeled accurately). This means that the guided mode of a single-mode fiber, for example, can often be sampled with a transverse step size which is only a few times smaller than its beam radius, even if that is much larger than the wavelength.

Size of the grid and behavior at boundaries: In principle, it is fully sufficient to make the grid so large that it just covers the region where non-negligible field amplitudes can occur. In practice, a grid extension to a radius of about twice the core radius (or even somewhat less) is normally sufficient.

Note, however, that a field hitting a boundary of the grid will reenter the grid on the opposite side. This behavior can be disturbing, as it is physically not realistic. It can be suppressed by introducing an artificial loss, which rapidly but smoothly increases toward the grid boundaries. For obtaining a sufficiently smooth transition, it may be necessary to make the grid somewhat larger.

It is also possible to make the grid boundaries reflecting on the *x* and/or *y* boundaries, or to define a reflecting boundary of arbitrary shape in the *x*-*y* plane; see section 5.13 for details.

The **longitudinal resolution** has to be so fine that within a single *z* step there are no strong influences of any effect (diffraction, refractive index variations, losses, etc.) on the fields. In other words, none of the mentioned effects alone should lead to substantial changes of the field profiles in one *z* step. For an ordinary step-index single-mode fiber with a mode radius of 5 µm at 1 µm wavelength, for example, the Rayleigh length of the mode in the fiber would be of the order of 120 µm. As the beam profile changes substantially within one Rayleigh length, the step size should be of the order of 20 µm or below in this case. For fibers with smaller modes and/or higher numerical aperture, a smaller step size may be required. In some cases, it may have to be of the order of a wavelength – which implies many propagation steps even just for propagating light over 1 mm of fiber.

Generally, it is recommend to test whether the results of a simulation change substantially when the resolution in transverse or longitudinal direction is doubled, for example.

Beam propagation can in principle be done not only for fibers and waveguides, but also for large optical systems such as free-space optical systems with lenses. However, the transverse resolution needs to be high if significant beam angles are involved, and that will then lead to very large grids. In addition, very fine *z* steps may be required due to large index contrasts. In this area, there can be some practical limitations for beam propagation simulations.

2.8.5 The Optical Channels

Each beam propagation device can have multiple "optical channels", having different optical wavelengths. (Interferences and beating effects are not considered, even if the wavelengths are close.) One defines a refractive index profile for each optical channel, where the refractive index can depend on *x* and *y*, and even on *z*. A user-defined refractive index function is evaluated for all grid points.

The used index profiles can be modified by a radius of curvature of the fiber in *x* and *y* direction, which may depend on *z*. A curvature in *x* direction, for example, is treated as a correction to all index profiles which is proportional to the *x* coordinate:

$$n_{\text{eff}}(x, y) = n(x, y) \cdot \left(1 + \frac{x}{R_x}\right)$$

Note that this way of treating the curvature ignores the effects of stress on the glass. For silica fibers, it has been shown (A. B. Sharma et al., Appl. Opt. 23, 3297 (1984)) that the effective bend radius is ≈1.28 times the geometric bend radius: the stress effects partially compensate the geometrical effects. One should thus use accordingly increased bend radii in the software.

For each optical channel, one also defines a two-dimensional complex amplitude profile as the initial field, which is then propagated along the fiber. The numerical propagation is automatically done only up to the *z* coordinate for which amplitudes have been requested via some functions. (Therefore, one may quickly get results for small *z* values, before further amplitudes have been calculated.) The calculated amplitudes are stored, such they can be quickly recalled later on. Also, they can be interpolated in the *x*-*y* plane. One can generate plots of various kinds which can depend on amplitudes at any positions within the numerical grid.

Optical fields in different optical channels of the same beam propagation device interact with each other only if there is laser amplification or a Kerr nonlinearity (see below).

2.8.6 Laser Amplification

The fiber of a beam propagation device can contain laser-active ions. A simplified gain model is used, where there can be only one type of laser-active ion (e.g. Er or Yb), having only a single metastable level (the upper laser level). (Quasi-three-level behavior is allowed.) The relevant spectroscopic data are

- the upper-state lifetime
- absorption and emission cross sections as functions of the wavelength

One can define an arbitrary two-dimensional profile of the doping concentration. A *z* dependence of the doping concentration is so far not allowed.

The local laser gain or absorption is normally calculated from the local optical intensities in the steady state. Alternatively, one can do dynamic simulations, where some time interval is defined by the user. In that case, the propagation simulates the temporal evolution of the upper-state population, which moves towards the steady state.

The optical channels interact with each other because they can influence the upper-state populations of the laser-active ions, which in turn influence the gain or absorption for all channels.

In some cases, one wants to do beam propagation only for a signal wave, but not for a pump wave. The pump intensity distribution can then be defined as a user-defined function. The advantages of that approach are that the computation speed is increased, and that backward pumping can be simulated. Of course, mutual saturation effects cannot be calculated that way; when determining the pump intensity distribution, one does not yet know how that pump intensity is influenced by the signal intensity.

2.8.7 Kerr Nonlinearity

Beam propagation can take into account the Kerr nonlinearity of the medium. For that, the user can define the nonlinear index of the material, which can be a function of *x* and *y*. For example, the core of a fiber may be more nonlinear than then cladding.

The Kerr nonlinearity leads to self-phase modulation (SPM) within an optical channel, but also to cross-phase modulation (XPM) between different channels of the same beam propagation device. The strength of cross-phase modulation can be reduced by the user to take into account the influence of different polarization directions.

2.9 Ultrashort Pulse Propagation

Since version 4, the software also has features for simulating the evolution of the temporal and spectral profiles of ultrashort pulses propagating through a fiber amplifier. Section 5.15 explains how to implement such simulations using the script language.

2.9.1 Initial State of the Fiber

The details of pulse propagation usually depend on the initial state of the fiber, i.e., on the excitation densities of the laser-active ions. (An exception is a case where the fiber has no laser-active doping.) That initial state is usually obtained with a continuous-wave simulation.

For example, consider the amplification of a continuous pulse train with a high repetition rate of 1 MHz. Here, each single pulse has an energy which is far below the saturation energy of the fiber, so that gain saturation by a single pulse is negligible. However, many pulses in combination do lead to gain saturation. The steady state of the fiber is then accurately approximated by a simulation where a continuous-wave signal is injected, the power of which is equal to the average power of the pulse train. In case that the pulses have a large optical bandwidth, one may need to do a simulation with multiple signal channels, overall approximating the spectrum of the pulse train. A remaining problem might be that the evolution of the pulse spectrum during its passage through the fiber deviates from that for the taken continuous-wave signals due to nonlinear effects. However, this should happen only in very special cases.

The ultrashort pulse propagation will generally modify the excitation densities in the fiber. This means that subsequent pulse propagation simulations will start with a modified initial state of the fiber.

2.9.2 Numerical Representation of an Ultrashort Pulse

As the fiber modes determine the transverse shapes of the field distribution, it is sufficient to consider only the variation of pulse properties along the propagation axis (*z* direction).

The state of a pulse at a certain z position can be stored either in the form of an array of complex amplitudes *A(t)* in the **time domain**, or as an array of complex amplitudes *A(f)* in the **frequency domain**, or both. While the time samples are located symmetrically around $t = 0$, the center of the frequency trace is at a reference frequency f_0 which is determined from the center wavelength of the corresponding optical channel. The number of samples, which is always a power of 2, is determined in the script via a function call.

Note that in the scientific literature there are different conventions for signs and amplitude normalizations. In the following, the conventions used in **RP Fiber Power** are explained. Time and frequency domain are related by the following equations for the complex amplitudes:

$$A(t) = \int_{-T/2}^{+T/2} A(f) \cdot \exp(-i2\pi\, f\, t)\, \mathrm{d}f$$

$$A(f) = \int_{-F/2}^{+F/2} A(t) \cdot \exp(+i2\pi\, f\, t)\, \mathrm{d}t$$

where F is the width of the covered frequency interval, determined as the inverse temporal resolution.

The complex amplitude *A(t)* is normalized such that the optical power is

$$P(t) = |A(t)|^2 .$$

The “electric field” is defined as

$$E(t) = \mathrm{Re}\big(A(t)\exp(-i2\pi f_0 t)\big),$$

which does not have the units of an electric field but rather the units $\sqrt{\mathrm{W}}$. Consequently, the units of $A(f)$ are $\sqrt{\mathrm{J/Hz}}$.

The initial pulse is usually centered around *t* = 0, but the pulse may drift away from that position, for example as an effect of chromatic dispersion. If the pulse reaches the end of the used time trace, it will normally reappear on the other end. In some cases, it is desirable to prevent this wrapping, which can also occur for broad pulse pedestals. Therefore, the software provides means to attenuate the ends of the time trace. Also, it is possible to have the pulse centered to $t = 0$ after each step.

Instead of a single pulse, the "start pulse" may also consist of multiple pulses. Although the "pulse" is considered to belong to a single optical channel, it spans some range of optical frequencies (see above), which means that different sub-pulses may have different wavelengths, associated with different time evolution of the optical phases.

2.9.3 Physical Effects Acting on the Pulses

Various effects occurring in a fiber can modify the temporal or spectral amplitude profile of a pulse:

- The amplification (or possibly absorption) generally leads to wavelength-dependent amplitude changes. There is also the effect of gain saturation: an intense pulse may substantially modify the population densities of the laser-active ions. Amplifier noise and phase changes due to Kramers−Kronig effects are not taken into account.
- Background losses are treated without a wavelength dependence.
- Chromatic dispersion leads to wavelength-dependent phase changes.
- Nonlinear effects are modeled in the time domain. In the simplest case, we only have an instantaneous nonlinearity (Kerr effect) and ignore self-steepening. This leads to self-phase modulation (SPM), i.e., time-dependent phase changes in proportion to the temporally changing optical power:

$$\Delta\varphi(t) = \gamma \left|A(t)\right|^2 \tag{6}$$

- It is also possible to introduce a delayed nonlinear response, where the change of the complex amplitude at one time can also depend on the optical power at all previous times:

$$\Delta A(t) = i\gamma \left(1 + \frac{i}{\omega_0}\frac{\partial}{\partial t}\right)\left(A(t)\int_0^T R(\tau)\left|A(t-\tau)\right|^2 d\tau\right) \tag{7}$$

  where

$$R(\tau) = (1 - f_\mathrm{R})\,\delta(\tau) + f_\mathrm{R}\,h(\tau), \tag{8}$$

  $\delta(\tau)$ is the Dirac delta function (representing an instantaneous response), $h(\tau)$ is a given Raman response function (normalized such that its time integral is 1), and f_R specifies the relative strength of the delayed nonlinear response. The term with $\frac{i}{\omega_0}\frac{\partial}{\partial t}$ describes the effect of self-steepening and may be omitted in some cases in order to increase the speed of the computations. Self-steepening can also be modeled while ignoring the delayed part of the nonlinear response.

2.9.4 The Current Pulse

The software stores a current pulse, which is relevant in various ways:

- There are various functions for defining a start pulse – for example, a Gaussian pulse with given parameters, or a pulse with arbitrary temporal or spectral profile. These functions set the mentioned current pulse.

- Further functions let the current pulse propagate through some optical element – for example, a piece of fiber, but there are also various other types of elements which can be simulated, such as spectral filters and saturable absorbers. After application of such a function, the current pulse will generally be modified. One can use subsequent function calls to simulate the effect of any sequence of optical elements.
- The software offers various functions delivering properties of the current pulse, such as the pulse energy, peak power, duration, bandwidth, etc. See section 6.1.6 for details.

For simulating pulse formation in a mode-locked fiber laser, one will usually do the following:

- Define a start pulse, which is a first rough estimate of the pulse expected for the steady state at some location in the laser resonator – typically, just before the output coupler mirror.
- Define a function, which applies all the effects experienced by the pulse in a complete resonator round trip. Call that function many times, and possibly save the pulse after each round trip in order to later recall the evolution of pulse parameters. For getting the output pulse after any number of round trips, one may recall the stored internal pulse and apply the action of the output coupler mirror in transmission.

See also section 7.7.

2.9.5 Pulses Propagation through a Fiber

A fiber is just one of many possible optical elements, through which a pulse can be propagated with the software. One may also define multiple pieces of fiber and propagate the pulse through any of these.

A pulse may be injected into the fiber either at $z = 0$ (left fiber end) or at $z = L$ (right fiber end), depending on the propagation direction of the corresponding optical channel.

After propagating a pulse through some fiber, it is also possible to retrieve the pulse at positions within the fiber. The pulses in the fiber are stored on some given equidistant grid, the step size of which is the same as that mentioned in section 2.1.1. The z position used when retrieving a pulse is rounded to the closest grid position; no interpolation of pulse properties is done in z direction.

If the chosen grid is too coarse for an accurate calculation of pulse propagation, the software automatically uses intermediate steps, but it does not store the resulting intermediate pulses.

Note that fiber end reflections as discussed in section 2.5 are *not* taken into account for the ultrashort pulse propagation, because one may want to take into account the effects of other optical elements on the left or right side of the fiber.

2.9.6 Pulse Properties

Once a pulse at a certain position in the fiber has been calculated, the software offers various functions for calculating properties of that pulse such as the pulse energy, peak power, duration, bandwidth, etc.

2.10 General Assumptions for Power Calculations

The general assumptions underlying the physical model of **RP Fiber Power** are:

- The doping concentrations must be constant in the longitudinal direction. (They can, however, depend on the radial and azimuthal coordinates.)

- All dopant ions of a certain type essentially behave identically, i.e., the gain medium exhibits homogeneous broadening.

  This assumption is well fulfilled for many ytterbium-doped and erbium-doped fibers, for example, but not e.g. for some neodymium-doped silica fibers. For various reasons, it would be difficult to model such cases. Even if the software could do that, it would be very difficult to obtain the corresponding spectroscopic data.

- Spatial hole burning as another possible origin of inhomogeneous broadening is also neglected.

  In typical cases, the emission bandwidth of fiber lasers is such that spatial hole burning effects are very weak and can be safely neglected.

- It is assumed that the optical channels preserve the given intensity shape during propagation in the fiber.

  This assumption is usually very well fulfilled if a channel corresponds to a certain mode of the fiber. However, if a channel consists of multiple modes, its intensity profile may actually change as the power distribution over the modes could change; then, the assumption may be not well fulfilled. This can be a problem with cladding-pumped fibers, when the pump channel is assumed to have a simple top-hat intensity profile, consisting of many modes. See also section 7.6 on the modeling of double-clad devices.

  Note also that mode coupling effects may occur under certain circumstances, such as strong bending of a fiber. Such effects cannot be simulated with the software.

- The intensities corresponding to different channels are simply added up, implicitly assuming that there is no coherence between them.

  This assumption is well fulfilled for broadband signals. However, there could be cases where this assumption is not valid – for example, when a single-frequency signal is injected into a multimode fiber such that multiple modes are excited. Their mutual coherence would lead to interference effects, the modeling of which would of course require the knowledge of the relative phases of the modes.

- Nonlinear optical effects in the fiber (e.g., stimulated Raman and Brillouin scattering) are neglected (except for ultrashort pulse simulations).

  This is valid for most fiber lasers and amplifiers operated with continuous-wave inputs. For pulsed devices, there may be nonlinear effects which can change the performance. The software cannot take these into account in the dynamic calculations (section 2.8), but it allows to check whether this nonlinear regime is entered – for example, by calculating the Raman gain from the optical powers.

- The population distributions within each Stark level manifold of the laser-active ions are always in thermal equilibrium.

  This assumption is usually very well fulfilled for steady-state situations, but in some dynamic cases (section 2.8) (amplification of very short and intense pulses) it can be wrong.

- For the mode solver, it is assumed that the fiber is weakly guiding, i.e., the refractive index contrast is not excessive. Also, the index profile needs to be radially symmetric and real.

These assumptions are valid for basically all doped glass fibers, except for photonic crystal fibers. The refractive index actually becomes complex for fibers with strong gain or absorption, and this could not be handled by the software. However, in almost all practical cases the gain or loss in a fiber is too weak to make this relevant.

- In ultrashort pulse simulations, only a single fiber mode is considered. Nonlinear coupling effects involving higher-order modes or a counterpropagating wave (as can occur for Brillouin scattering and Raman scattering) cannot be modeled.

3 Spectroscopic Data

The software requires some data on the fibers. In this section, it is only described where these data are stored and which data are provided with the software, whereas section 5.5 explains how spectroscopic data can be defined. See also section 7.2 with some hints for importing spectroscopic data.

It is advisable to always store commands and data for the definition of spectroscopy data in `include` files (see section 5.1), because this allows to conveniently use them in other scripts, and also allows to conveniently switch between the data for different fiber types. In principle, however, you may make such definitions within any script.

Normally, such `include` files have a name beginning with “Yb-“, for example (indicating ytterbium as the laser-active ion), the extension **.inc**, and are stored in a folder named “Spectroscopic data”, which is a subfolder of the folder containing the RP FiberPower program files (for example, “**C:\Program Files\RP Software\RP Fiber Power\Spectroscopic data**”). You may, however, want to use a copy of that folder at a different location (for example, some user folder). A reason for this can be that Windows 7 does allows write access to subfolders of “Program Files” only to administrators, so that you may have difficulties adding data files. See **Options | Change location** of fiber data folder (section 4.5.5).

The following files are provided with the software:

- **Yb-germanosilicate.inc**: data for an ytterbium-doped germanosilicate fiber, recorded by Dr. Paschotta and published in R. Paschotta et al., “Ytterbium-doped fiber amplifiers”, IEEE J. Quantum Electron. 33 (7), 1049 (1997).
- **ErYb-phosphate.inc**: data for an erbium- and ytterbium-doped phosphate glass, taken from G. C. Valley, “Modeling cladding-pumped Er/Yb fiber amplifiers”, Optical Fiber Technol. 7, 21 (2001); the data were partly taken from S. Konkanen et al., Proc. SPIE 2996, 32 (1997).
- **Er-fluorophosphate L11**: data for an erbium-doped fluorophosphates glass, taken from W. J. Miniscalco and R. S. Quimby, “General procedure for the analysis of Er^{3+} cross sections”, Opt. Lett. 16 (4), 258 (1991).
- **Er-fluorozirconate F88**: data for an erbium-doped fluorozirconate glass, taken from W. J. Miniscalco, "Erbium-doped glasses for fiber amplifiers at 1500 nm", J. Lightwave Technol. 9 (2), 234 (1991).
- **Er-silicate L22**: data for an erbium-doped silicate glass, taken from W. J. Miniscalco and R. S. Quimby, “General procedure for the analysis of Er^{3+} cross sections”, Opt. Lett. 16 (4), 258 (1991).

(Note that the first two letters of the filename always indicate the type of laser-active ions.)

In addition, RP Photonics can provide data for various commercially available fibers. See the web page http://www.rp-photonics.com/fiberpower_data.html for details.

4 The User Interface

This section describes the basic handling of the program. For the syntax of the script files, see section 5.

4.1 General Principles

RP Fiber Power can be controlled in two different ways:

- Basically, all calculations and all output of calculated quantities are defined by script files with the filename extension **`.fpw`**. Such a script file contains certain commands, e.g. for defining the physical properties of an active fiber and the details of graphical output. Direct writing of script files is an extremely powerful and flexible approach, allowing one to set up most sophisticated models and calculation, but it requires some familiarity with the script language.

- In not too complicated cases, it is also possible (and often easier) to fill all required data into interactive forms (see section 4.4). The software can then create a script file based on the entered data, and execute that script file. For more sophisticated modeling, one may then further refine the script file.

For these two methods, **RP Fiber Power** has two different input modes (see section 4.3): the **script editor mode** and the **forms mode**.

RP Fiber Power is usually started by double-clicking an **`.fpw`** file in the Windows Explorer. It then presents an editor in which that script file can be edited. See section 4.6 for more details on editing scripts.

It is also possible to start **RP Fiber Power** via a Windows shortcut, where the project folder is passed to the program as a command line parameter. Similarly, the software can be started via the Windows programs menu. The Windows user data folder is then always used as the working folder (active folder).

Once a script file is loaded into the active editor, it can be "executed" by pressing the F8 key, which leads to menu item **Execute | Graphics** (see section 4.5.3). This means that the script file is read, and if this is successfully interpreted, graphics windows according to the definitions in the file will be made (see section 4.5.3). With F9, one can execute a script while suppressing the generation of graphics windows.

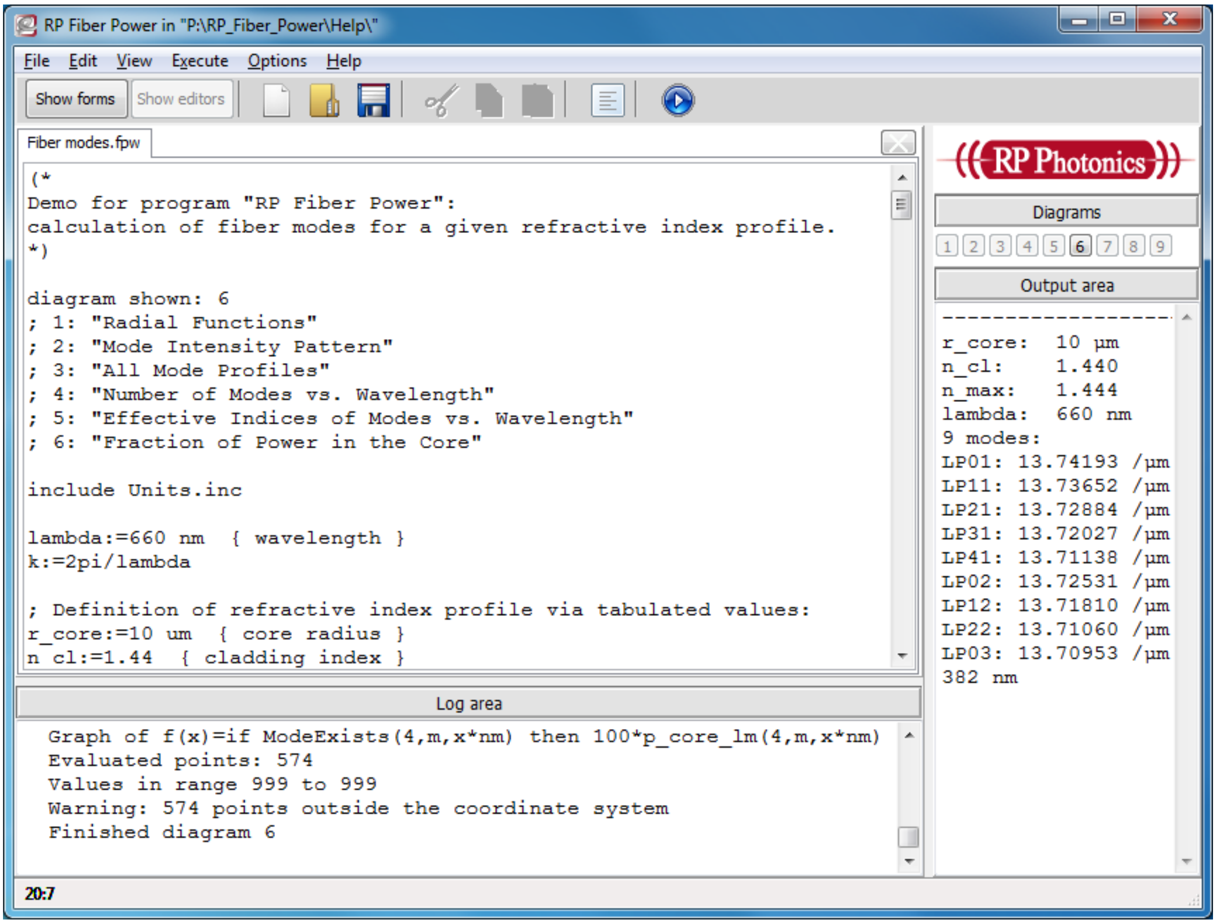
4.2 Command Line Parameters

When one double-clicks on a script file (extension **`.fpw`**) in Windows Explorer, the name of that script is passed to the software as a command line parameter, and this lets the software open an editor into which that script is loaded. Similarly, form settings can be loaded from an **`.fpi`** file.

If the command line parameter is a folder, the software starts in that folder. Settings are read from the desktop file **`RP_Fiber_Power-W.dsk`**, if that is present. The same happens when the desktop file is passed as a command line parameter. Therefore, you may drop the icon of a desktop file on a shortcut for RP Fiber Power in order to start the software in its folder.

4.3 The Main Form

Below you see a screen shot from the program’s main form in the **script editor mode**, taken after a demo script has been loaded and executed:



```
(*
Demo for program "RP Fiber Power":
calculation of fiber modes for a given refractive index profile.
*)

diagram shown: 6
; 1: "Radial Functions"
; 2: "Mode Intensity Pattern"
; 3: "All Mode Profiles"
; 4: "Number of Modes vs. Wavelength"
; 5: "Effective Indices of Modes vs. Wavelength"
; 6: "Fraction of Power in the Core"

include Units.inc

lambda:=660 nm  { wavelength }
k:=2pi/lambda

; Definition of refractive index profile via tabulated values:
r_core:=10 um  { core radius }
n_cl:=1.44  { cladding index }
```

```
-------------------
r_core:   10 µm
n_cl:     1.440
n_max:    1.444
lambda:   660 nm
9 modes:
LP01: 13.74193 /µm
LP11: 13.73652 /µm
LP21: 13.72884 /µm
LP31: 13.72027 /µm
LP41: 13.71138 /µm
LP02: 13.72531 /µm
LP12: 13.71810 /µm
LP22: 13.71060 /µm
LP03: 13.70953 /µm
382 nm
```

```
  Graph of f(x)=if ModeExists(4,m,x*nm) then 100*p_core_lm(4,m,x*nm)
  Evaluated points: 574
  Values in range 999 to 999
  Warning: 574 points outside the coordinate system
  Finished diagram 6
```

Notice the following elements:

- The **title line** displays the program line and the folder in which the program has been started. That should always be the folder corresponding to the simulation project.
- Below the title line, there is the **main menu** (see section 4.5).
- Below the menu, there is the **toolbar** with various icons for quickly finding some functions. For example, the buttons on the left side can be used to switch between the script editor mode and the forms mode. Also, one can execute a script using the blue “Play” button. On the left side, there is the button for opening the interactive forms (see the next section).
- Below the toolbar, there is the **tab control with the editors**. Here, you can edit your scripts (and other text files). If you are in the interactive forms mode instead of the script editor mode, you see the **interactive forms** instead of the script editors.
- At the lower left you find the **log area** (section 4.7), where potentially useful information is displayed when a script is executed.
- On the right side, you find several items:
    - You can click on the **RP Photonics logo** in order to get to the RP Photonics website.

- The **diagrams area** shows you buttons for the first 9 graphics windows. You can click on such a button to recall one of these graphics windows. These buttons are active only once a script has been executed and created these graphics.
- The **output area** (section 4.8) can contain text output created by scripts.

- At the very bottom, there is the **status line**, which displays the cursor position in the editor or what is currently done during script execution.

You can resize the log area and the output area by grabbing and moving the splitters next to them. When resizing the whole window, the sizes of the log area and output area are normally maintained as far as possible. If you hold the Ctrl key pressed while resizing the window, you can resize the log area and output area.

4.4 Working with Interactive Forms

The interactive forms constitute a very simple way of getting started with the software. You can enter the required information into various fields of the forms and finally “execute” the software. This means that a script file is generated from your inputs and then executed. Some outputs of the script are also displayed in fields of the forms.

If the software is in the script editor mode, you can get the interactive forms by using the “Show forms” button in the toolbar.

The inputs are organized in the following tabs:

- **Fiber modes:** This is for calculating the modes of the fiber from a refractive index profile. (You do not need this for power calculations if you know the intensity profiles from other sources.)
- **Active fiber:** Here, you can define the details of the active fiber.
- **Optical channels:** All light in the fiber is described with “optical channels” (see section 2.4), which you can define here. For example, an amplifier model may have several pump channels, several signal channels and ASE channels.
- **Definitions:** Here, you have the opportunity to enter some script commands which are integrated into the generated script. You may use this e.g. to define user-defined intensity shapes for some optical channels, or to define additional outputs.
- **Graphics:** Here, you can define which types of graphical output are generated. (Note that the graphics on fiber modes can be defined in the “Fiber mode” tab.)
- **Ultrashort pulses:** This tab is for simulating the propagation of ultrashort pulses.

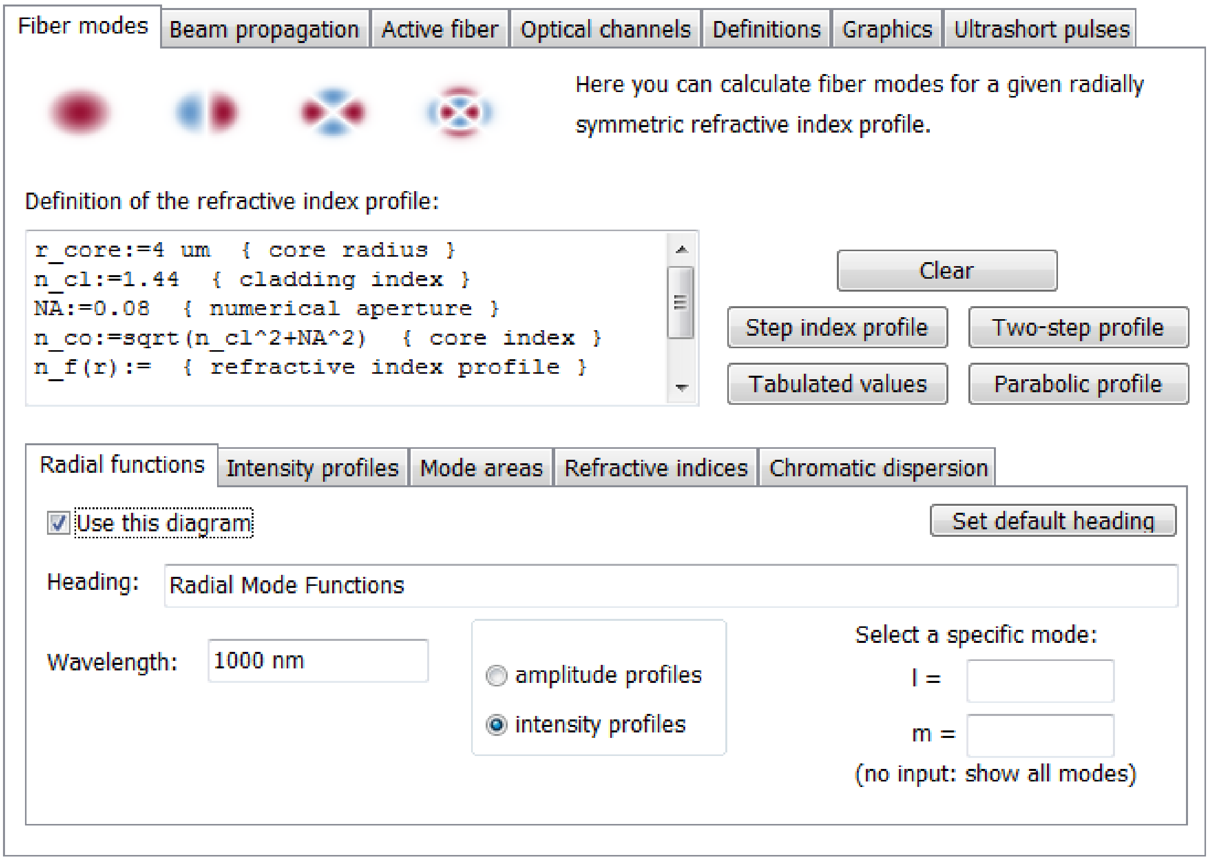
These tabs are explained in detail in the following sections.

In the toolbar, you find the blue “Play” button. When this is pressed, a script file name **FormScript.fpw** is generated from the form inputs and is executed. The menu item **Execute | Calculate (no graphics)** does the same, but graphical output is suppressed. You may go to the script editor mode in order to inspect the generated script and possibly save a copy of it, which you can edit further. (The generated script cannot be edited, because it may be overwritten the next time the form is executed.)

With the Save function (**File | Save** in the menu, or the floppy disk button in the toolbar) in the interactive forms mode, you can save the form settings to an **`.fpi`** file. This can later on be loaded with **File | Open**. The form settings are also saved in the settings file of your project folder.

4.4.1 Calculating Fiber Modes

The screen shot below shows the tab for calculating fiber modes.



Here you can calculate fiber modes for a given radially symmetric refractive index profile.

Definition of the refractive index profile:

```
r_core:=4 um  { core radius }
n_cl:=1.44  { cladding index }
NA:=0.08  { numerical aperture }
n_co:=sqrt(n_cl^2+NA^2)  { core index }
n_f(r):=  { refractive index profile }
```

☑ Use this diagram

Heading: Radial Mode Functions

Wavelength: 1000 nm

○ amplitude profiles

● intensity profiles

Select a specific mode:

l =

m =

(no input: show all modes)

(That tab control is part of the main form.)

In the text field “Definition of the refractive index profile”, you can insert script commands for defining the function **`n_f(r)`**, which represents the refractive index profile. (The index function can also have a second argument, the optical wavelength.) Before the actual function definition, one will usually define some parameters such as the cladding index and the core radius **`r_core`**.

The buttons to the right of this field allow one to obtain the code for various typical index profiles. Of course, you can then edit that code.

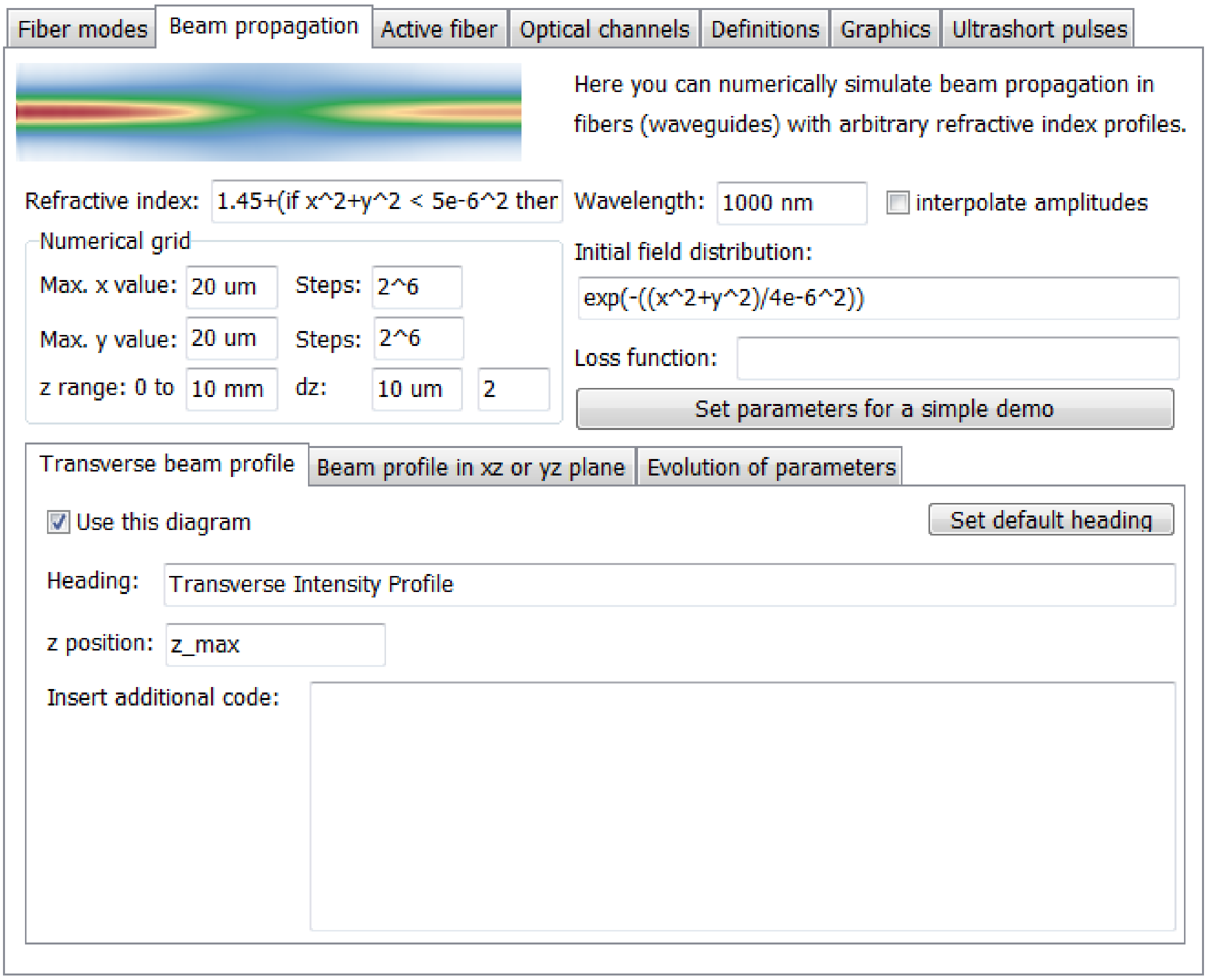
In the lower part of the form, you can select one or several diagrams for illustrating properties of fiber modes:

- Radial functions: plot the radial amplitude or intensity profiles of a specific mode, or of all guided modes.
- Intensity profiles: show the intensity or amplitude profile of a specific mode, or of all guided modes.
- Mode areas: plot the effective mode areas as functions of the wavelength.
- Refractive indices: plot the effective refractive indices or the group indices as functions of the wavelength.
- Chromatic dispersion: plot the group velocity dispersion versus wavelength.

Note that the appearance of the diagrams is influenced by some settings in the Graphics tab (at “Settings”, see section 4.4.6).

4.4.2 Beam Propagation

Here, you can create a simple simulation of beam propagation (see section 2.8).



Here you can numerically simulate beam propagation in fibers (waveguides) with arbitrary refractive index profiles.

Refractive index: 1.45+(if x^2+y^2 < 5e-6^2 ther

Wavelength: 1000 nm

□ interpolate amplitudes

Numerical grid

Max. x value: 20 um Steps: 2^6

Max. y value: 20 um Steps: 2^6

z range: 0 to 10 mm dz: 10 um 2

Initial field distribution: exp(-((x^2+y^2)/4e-6^2))

Loss function:

☑ Use this diagram

Heading: Transverse Intensity Profile

z position: z_max

Insert additional code:

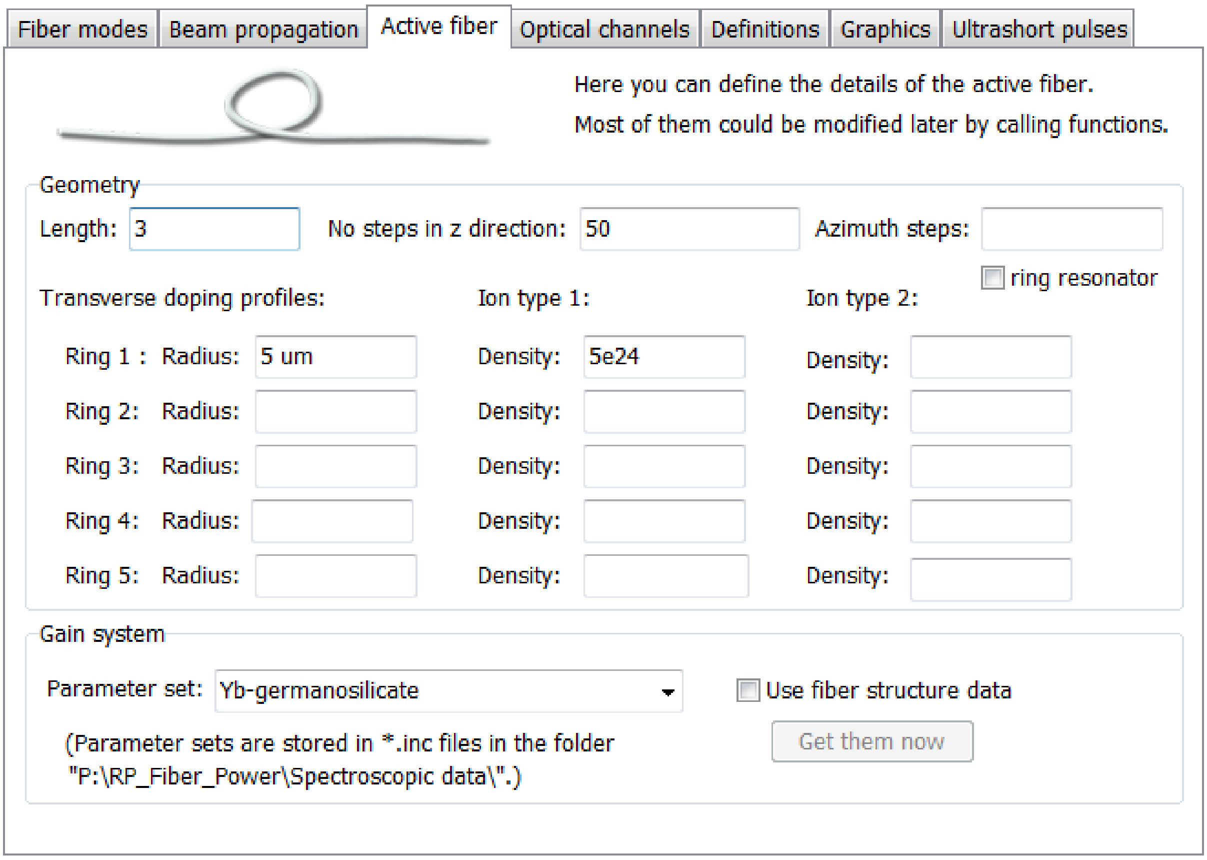
The following inputs can be inserted:

- Refractive index: define the index as a function of *x* and *y*. If the space in the field is not sufficient, you can predefine a function in the “Definitions” tab and call that function in the field.
- Wavelength: enter the wavelength for which the beam propagation should be done. (You can have only one optical channel with a certain wavelength here, but in scripts you could have many of them.)
- Numerical grid: enter suitable settings – see section 2.8.4 for details.
- Initial field distribution: enter a function for the complex amplitude, depending on *x* and *y*.
- Loss function: enter the local propagation loss, which may depend on *x* and *y*.

Further, you can use the lower part of the form for defining certain standard diagrams.

4.4.3 Defining the Parameters of an Active Fiber

The screen shot below shows the tab for the fiber parameters. You need this for simulating fiber amplifiers and lasers, but also for ultrashort pulse propagation in a passive fiber.



Here you can define the details of the active fiber.
Most of them could be modified later by calling functions.

Geometry

Length: 3

No steps in z direction: 50

Azimuth steps:

□ ring resonator

Transverse doping profiles:

		Ion type 1:	Ion type 2:
Ring 1 :	Radius: 5 um	Density: 5e24	Density:
Ring 2:	Radius:	Density:	Density:
Ring 3:	Radius:	Density:	Density:
Ring 4:	Radius:	Density:	Density:
Ring 5:	Radius:	Density:	Density:

Gain system

Parameter set: Yb-germanosilicate

□ Use fiber structure data

(Parameter sets are stored in *.inc files in the folder
"P:\RP_Fiber_Power\Spectroscopic data\".)

The data to be entered here are:

- the fiber length and the number of steps for the numerical representation (see section 2.1)
- the doping profiles: radii of up to five rings (first column) and the concentration values for one or two different laser-active ions (second and third column)
- the gain system: a parameter set containing the spectroscopic data (and possibly also data on the fiber structure and mode fields, see below)

For a ring resonator (see section 2.5), the “ring resonator” checkbox has to be activated.

For a passive fiber, you can select the parameter set “(passive fiber)”. In that case, you do not need to define any doping profile. This setting can be useful if you want to simulate ultrashort pulse propagation in a passive fiber.

The spectroscopic data are read from text files with the filename extension **.inc**, which are normally stored in a folder named “Spectroscopic data” (see section 3). If additional data files are put into that folder, they will automatically appear in the box for the selection of parameter sets.

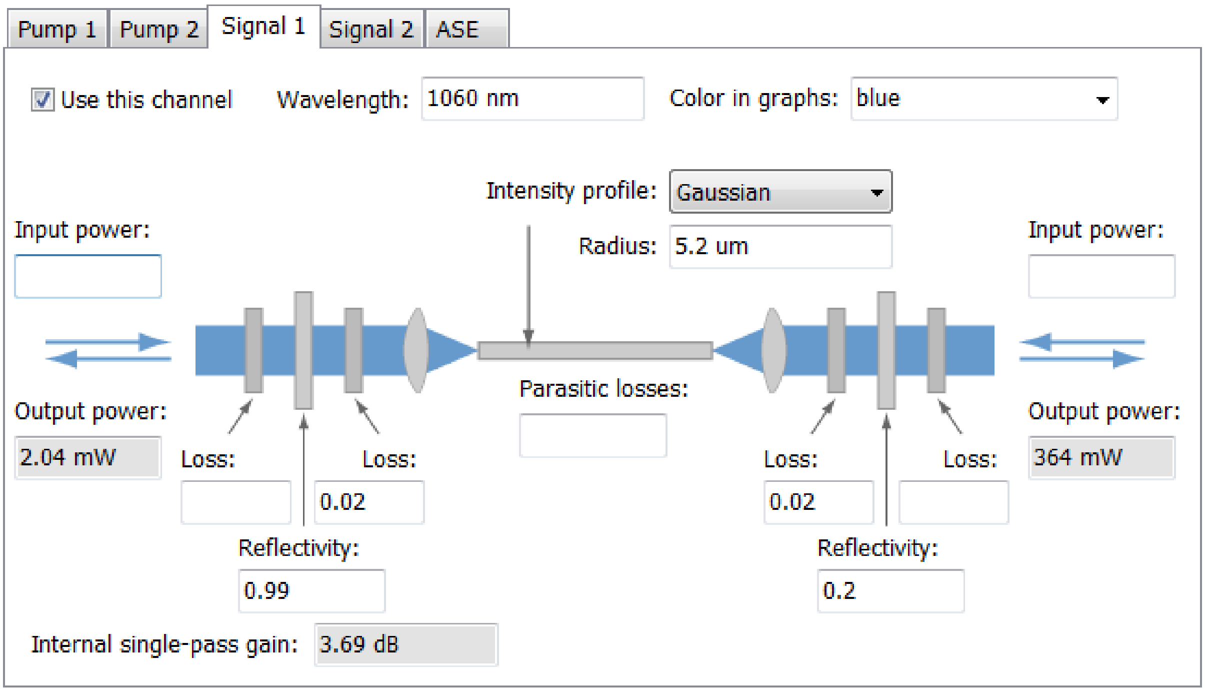
Note that the first two letters of the filename of the data file always indicate the type of laser-active ions.

In addition to spectroscopic data, fiber data files may contain information on the fiber structure[1], such as the core radius, doping concentration and mode profiles, which would normally be specified in the forms. In that case, these data are read from the file into the form, but only when the box “Use fiber structure data” (next to the data file selection box) is checked. The data are then read when the button “Get them now” is pressed or when the data set is executed. Data sets provided for commercial fibers usually contain such data, making it very easy to switch between different fibers.

In some cases, one needs to define the doping profile in a more sophisticated way than is possible via the form. For example, this is the case when one needs to define a rectangular grid (see section 2.1.2), or when one requires a large number of rings. In such a case, the doping profile can be defined in one of two other locations: in the spectroscopic data file or in the additional definitions, see the upper field of the “Definitions” tab (see section 4.4.5). At such a location, you then need to define a function named **`def_dopingprofile()`**. In addition, the variable **`max_r`** should then be defined, which is the maximum radial coordinate of the doped region. (This is used for the diagram with the transverse dependencies.) If the definition of the function **`def_dopingprofile()`** is found at one of the mentioned locations, a call to this function will be inserted into the generated script, instead of using doping data in the form.

4.4.4 Defining the Optical Channels

The screen shot below shows the tab for the optical channels.



☑ Use this channel

Wavelength: 1060 nm

Color in graphs: blue

Intensity profile: Gaussian

Radius: 5.2 um

Input power:

Output power: 2.04 mW

Loss:

Loss: 0.02

Reflectivity: 0.99

Parasitic losses:

Loss: 0.02

Loss:

Reflectivity: 0.2

Input power:

Output power: 364 mW

Internal single-pass gain: 3.69 dB

The form offers tabs for defining a forward-propagating and a backward-propagating optical channel for up to two pump wavelengths, two signal wavelengths, and many ASE (amplified spontaneous emission) wavelengths. For each of these, the data to be entered are the following:

- The check box “Use this channel” determines whether this kind of channel is considered.

[1] Such information is stored in lines beginning with “;IF_”.

- A color can be selected which used for that channel in all graphical output.
- Other parameters are the wavelength, the parasitic losses in the fiber (caused e.g. by Rayleigh scattering), the intensity shape, the radius of the intensity distribution, reflectivities at both fiber ends, additional parasitic losses at the ends, and input powers for the forward and backward direction.
- Multiple ASE channels can be defined by specifying a wavelength range and a wavelength step size. For example, for an ytterbium-doped amplifier one may choose wavelength channels from 960 nm to 1100 nm in steps of 5 nm; the bandwidth of each channel is then 5 nm.

Note that in case of a ring resonator both reflectivities R1 and R2 are applied to a forward-propagating channel only. (A backward-propagating laser channel can then not be defined via this form.) For a linear resonator, these reflectivities can couple two counterpropagating waves.

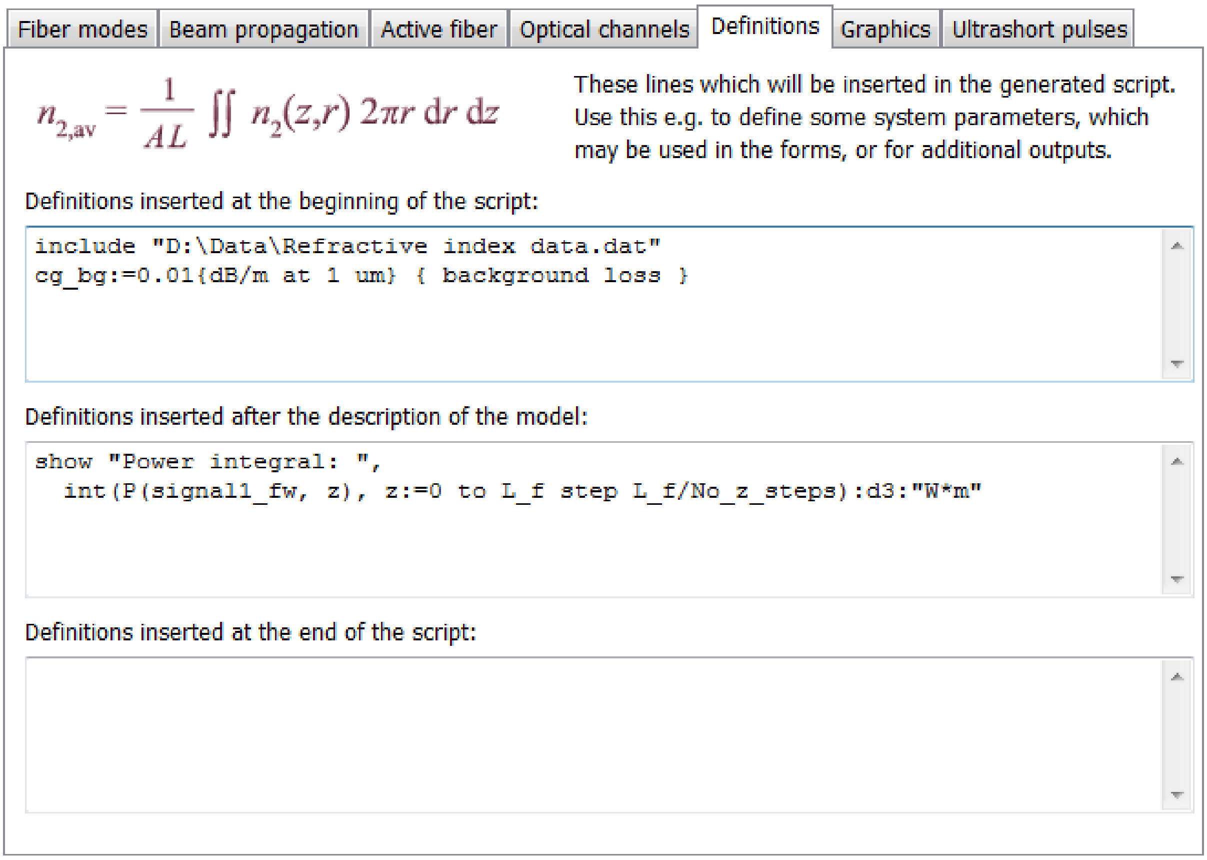
The slightly darker fields are output fields, displaying the calculated output powers and (for signal and ASE channels) the internal single-pass gain (which includes a reduction due to parasitic losses).

If you do a dynamical simulation (section 2.8), you must *not* enter time-dependent input powers in these fields. You should rather enter constant values, which determine a steady state. That steady state will be used as the initial state for the later dynamic simulation, as can be defined in the tab “Graphics” (see section 4.4.6).

For ultrashort pulse simulations (section 5.15), the end reflectivities are ignored.

4.4.5 Additional Definitions

The screen shot below shows the tab for entering additional definitions.



$$n_{2,\mathrm{av}} = \frac{1}{AL} \iint n_2(z,r)\, 2\pi r\, \mathrm{d}r\, \mathrm{d}z$$

These lines which will be inserted in the generated script. Use this e.g. to define some system parameters, which may be used in the forms, or for additional outputs.

Definitions inserted at the beginning of the script:

```
include "D:\Data\Refractive index data.dat"
cg_bg:=0.01{dB/m at 1 um} { background loss }
```

Definitions inserted after the description of the model:

```
show "Power integral: ",
  int(P(signal1_fw, z), z:=0 to L_f step L_f/No_z_steps):d3:"W*m"
```

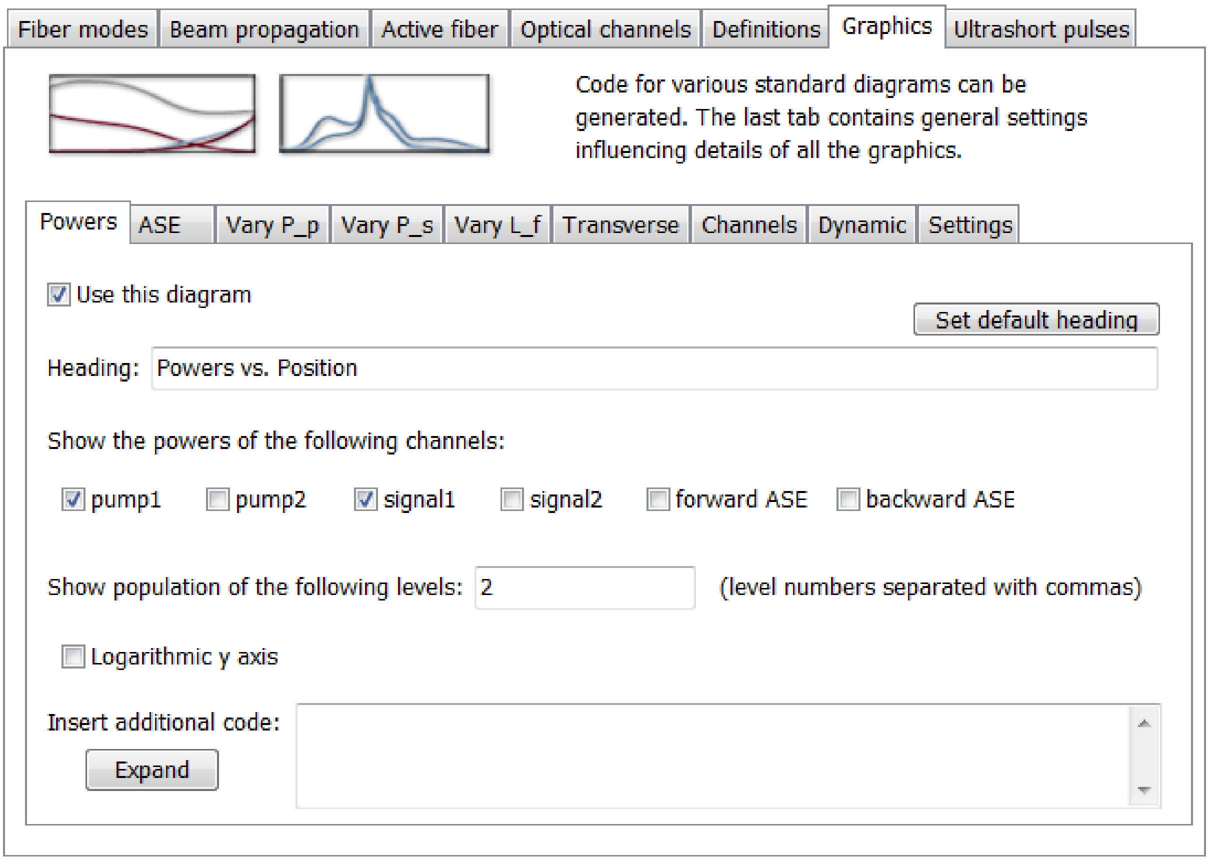
Definitions inserted at the end of the script:

Here, one can enter script commands which will be inserted into the generated script. The upper part is inserted before the definition of the model, while the middle part is inserted after that definition, and the lower part at the end. Normally, it is not necessary to enter anything here, but examples for applications are:

- If the channel named **`pump1`** should have a user-defined intensity profile, choose “user-defined” in the channel settings and define the function **`I_p1(r)`** in the upper definitions field.
- Also in the upper definitions field, use a definition like **`w:=5 um`**, and then simply write **`w`** in the “Radius” fields of all channels. In that way you can modify the mode radius of all channels by modifying one form entry only. In all such fields, you may enter some mathematical expression (possibly containing variable values) instead of a simple number.
- In the lower definitions field, insert the command
  `show P(signal1_fw, L_f/2):d3:”W”`
  in order to display the signal power in the middle of the fiber.

4.4.6 Defining the Graphical Output

The screen shot below shows the tab for defining graphical output.



Code for various standard diagrams can be generated. The last tab contains general settings influencing details of all the graphics.

☑ Use this diagram

Heading: Powers vs. Position

Show the powers of the following channels:

☑ pump1 □ pump2 ☑ signal1 □ signal2 □ forward ASE □ backward ASE

Show population of the following levels: 2 (level numbers separated with commas)

□ Logarithmic y axis

Insert additional code:

Within that form, there are further eight tabs. The first seven of these define different types of graphical output, while the last one allows to do some general settings. The offered types of diagrams are:

- **Powers vs. z**: Here, the output powers of variables channels can be plotted as functions of the position in the fiber.
- **ASE spectrum**: You can plot the optical spectrum of amplified spontaneous emission, i.e., the optical power per nanometer of bandwidth as a function of wavelength.
- **Vary P_p**, **Vary P_s**: Here, you can vary some pump or signal input power in a certain range and display how this affects the output powers or the internal single-pass gain of various channels.
- **Vary L_f**: Here, you can vary the fiber length.
- **Transverse**: Here, you can plot the transverse dependencies of doping concentrations and mode intensities. All functions are normalized so that they fit to the same scale in the diagram.
- **Channels**: This diagram displays the channels which are defined in the script. This can be used to check whether the channel definitions are correct.
- **Dynamic**: This diagram displays results of a dynamic simulation (section 2.8). You can define time-dependent input powers for all pump and signal channels. (For channels without time-dependent input powers, leave these fields empty; these input powers then

remain unchanged.) You can also determine which powers and which level populations are plotted as functions of time, and what is the temporal range and the temporal step size. For further refinements, such as time-dependent reflectivities and additional time delays outside the fiber, it is necessary to work on the level of the script language.

Note that it is often convenient to define functions for the time-dependent input powers in the upper field of the “Definitions” tab (section 4.4.5) and use those in the input power fields, where you would have little space for long formulas.

Also note that due to space constraints the forms do not allow you to define time-dependent reflectivities. For such extended simulations, you will need to work on the script level.

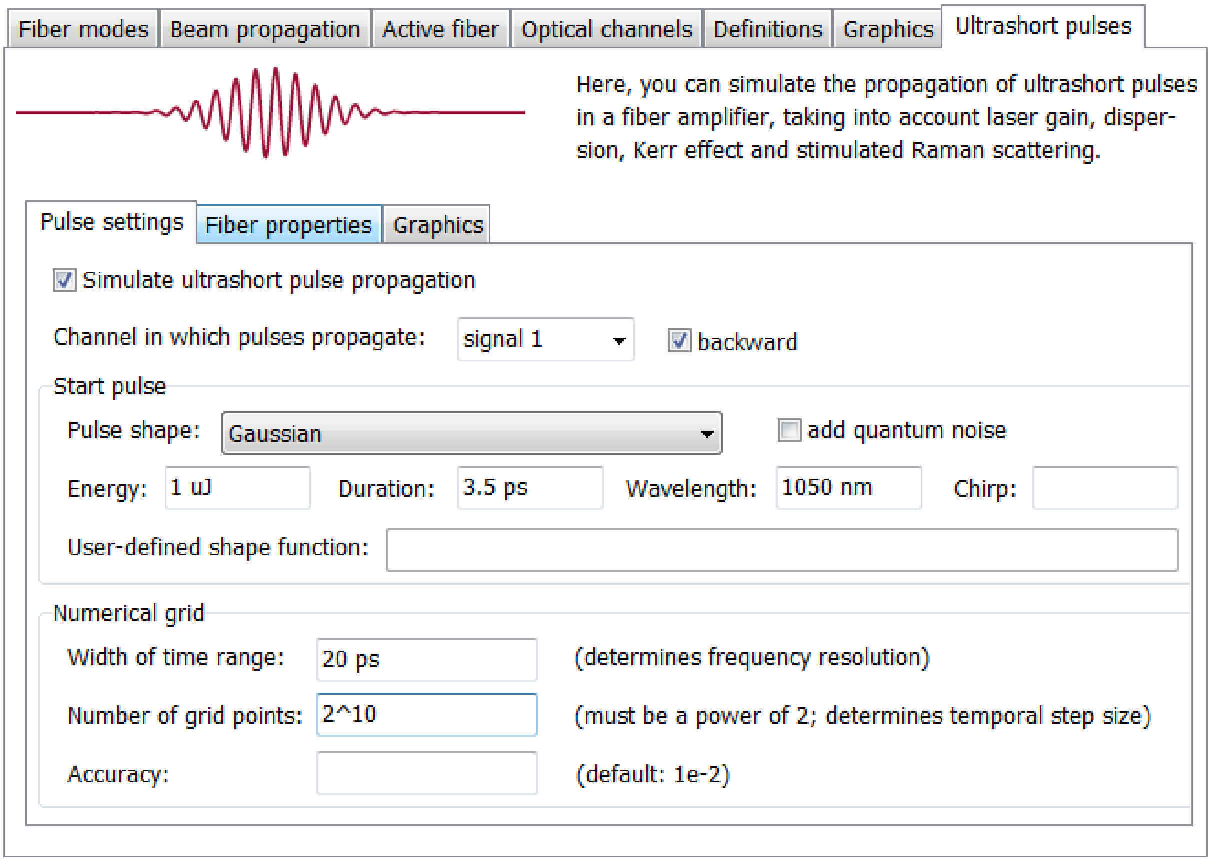
Most tabs have some input field named “Insert additional code” at the bottom. There, you can insert additional script code after the other definitions for the corresponding diagram. This can be used for displaying additional graphs and text labels, for example.

The tab with general **settings** allows one to define details like the character fonts and font sizes used in the diagrams, also the size of the generated graphics windows. For exporting graphics with high resolution into some other software, you may choose a large diagram size here.

Note that many more different diagrams could be defined in the script language. You may use a script generated with the interactive forms and then further refine it.

4.4.7 Ultrashort pulses

The screen shot below shows the tab for ultrashort pulse propagation.



Here, you can simulate the propagation of ultrashort pulses in a fiber amplifier, taking into account laser gain, dispersion, Kerr effect and stimulated Raman scattering.

☑ Simulate ultrashort pulse propagation

Channel in which pulses propagate: signal 1 ☑ backward

Start pulse

Pulse shape: Gaussian ☐ add quantum noise

Energy: 1 uJ Duration: 3.5 ps Wavelength: 1050 nm Chirp:

User-defined shape function:

Numerical grid

Width of time range: 20 ps (determines frequency resolution)

Number of grid points: 2^10 (must be a power of 2; determines temporal step size)

Accuracy: (default: 1e-2)

Within that form, there are three further tabs:

- **Pulse settings**: Here, you determine
  - whether ultrashort pulses are simulated
  - the details of the start pulse entering the fiber (a Gaussian or sech^2-shaped pulse with given parameters, or a pulse with a given temporal or spectral amplitude profile, or a pulse loaded from a file)
  - the parameters of the numerical grid: the width of the temporal range, the number of grid points and (optional) a parameter for the numerical accuracy
- **Fiber properties**: Here, you define additional details of the fiber, as are relevant for ultrashort pulses propagation:
  - the group velocity dispersion (GVD) (constant or wavelength-dependent)
  - the nonlinear refractive index
  - the self-steepening parameters
  - a function defining the delayed nonlinear response (for Raman scattering), and a weight factor for the nonlinear response
  - a coefficient for two-photon absorption (TPA) (not relevant for most fibers)

  You simply leave empty those fields which do not apply in your case.

  See section 2.9 for details on the meaning of these quantities.
- **Graphics**: Here, you can define details of graphical diagram showing results of the pulse propagation simulation:
  - **Pulses at z position**: This displays properties of the pulse at a given *z* position in the fiber in the time, frequency or wavelength domain.
  - **Parameters vs. z**: Here, the *z* position is varied along the horizontal axis, and a selected parameter of the pulse (e.g., the pulse energy) is plotted vs. *z*.
  - **Temporal evol**.: This displays the evolution in the time domain along the fiber.
  - **Spectral evol**.: This displays the evolution in the wavelength domain along the fiber.
  - **Variation of pump power**: The pump power (or a signal input power) is varied along the horizontal axis, and a selected pulse parameter is plotted.

  Each tab has an input field named “Insert additional code” at the bottom. There, you can insert additional script code after the other definitions for the corresponding diagram. This can be used for displaying additional graphs and text labels, for example.

4.5 The Menus

You find the main menu bar at the top of the program window. Select items e.g. by clicking them with the left mouse button. In the following we briefly list the available submenus.

Note that some of the actions can be called more easily by pressing the corresponding tool buttons (just below the main menu). A short explanation to a tool button is displayed if the mouse pointer rests on the button for a second.

4.5.1 The File Menu

This menu mainly deals with the script files.

- **New**: open an editor for a new **RP Fiber Power** file. If the file **`Template.fpw`** exists in the folder where the .exe file is stored, the new file will be created as a copy of that template file. Such a template is not provided, but you may make one.
- **Open**: In the script editor mode, you can open an editor with an existing script file (with extension **`.fpw`**), or in fact any plain text file. In the interactive forms mode, you can load form settings from an **`.fpi`** file.
- **Recently opened files**: this menu displays a list of files which were recently used. This makes it easy to re-open one of those files.
- **Save**: In the script editor mode, you can save the content of the active editor to the corresponding file. Note that saving is done automatically when a file is executed (see the Execute menu). In the interactive forms mode, you can save form settings to an **`.fpi`** file.
- **Save as**: same as **Save**, but you can first determine the filename.
- **Save all files**: save the content of all open editors to the corresponding files.
- **Close**: close the active editor.
- **Change the active folder**: switch to a different active folder (working folder). Note that each folder is considered to host a separate project. Therefore, the software stores the open files and some other settings for each of these folders separately in a file named **`RP_Fiber_Power-W.dsk`**. If you switch to a different folder, that file is updated (as it occurs also when you exit the program), all windows are closed, and the windows are opened which were open when you worked in that folder last time.
- **Clear the interactive form**s: this deletes all settings made in the interactive forms.
- **Print input file**: print out the content of the active editor.
- **Printer setup**: modify the settings for the printer.
- **Exit**: exit the program.

4.5.2 The Edit Menu

- **Undo**: undo the last editing operation.
- **Redo**: reverse the last undo operation
- **Delete word at cursor:**
- **Delete line**
- **Cut**: copy the selected text to the Windows clipboard and remove it from the text. Text can be selected by dragging the mouse or by moving the cursor while the Shift key is hold down.
- **Copy**: copy the selected text to the Windows clipboard.
- **Paste**: paste text from the Windows clipboard into the editor.
- **Find**: find some text in the editor.

- **Find next**: find the next occurrence of some text (entered before in the Find menu).
- **Replace text**: find some text in the editor and replace it with some other text.
- **Select all**: Select the whole content of the editor.
- **Goto**: go to the line with the given line number.

4.5.3 The View Menu

- **Forms**: go to the forms mode.
- **Editors**: go to the script editor mode.
- **Clear output area**: clear the output area.
- **Beam profile viewer**: show the interactive window for inspecting beam profiles from numerical beam propagation simulations (see section 4.10).
- **Pulse display window**: show the interactive window for displaying ultrashort pulses (see section 4.11).
- **Remove graphics windows**: remove the graphics windows belonging to the current simulation. Any graphics windows from a different model remain open.
- **Remove all graphics windows**: remove all graphics windows.

4.5.4 The Execute Menu

- **Graphics**: execute the script file in the active editor. Graphical output is displayed in one or more graphics windows, if it is defined in the script. Also, some information is displayed in the log area, and some outputs may be displayed in the Outputs area.
- **Graphics to printer**: execute the file in the active editor. Graphical output is sent to the printer. A dialog allows to set various parameters, e.g., how many images are placed on one page and how large the images are.
- **Graphics with Postscript**: execute the script and send all graphics to a Postscript file
- **Calculate (no graphics)**: execute the file in the active editor. No graphical output is generated. This is useful e.g. if only the Output area is wanted.

4.5.5 The Options Menu

This menu contains two entries:

- **Options ...**: This allows to change some parameters:
  - **Font name**: name of the font to be used in the editors.
  - **Font size**: font size to be used in the editors.
  - **Bold characters**: use bold style characters in the editors.
  - **Search path for include files**: specify one or several directories (separated with “;”) where include files (see section 5.2) are searched for when they are not found in the current directory. This search path also applies to several other commands where input files are needed, in particular to **`readlist`**.
- **Change location of fiber data folder**: The spectroscopic data (and possibly also data on fiber structures) are stored in **.inc** files in some folder. Here, you may choose a different

location for that folder. (This may be necessary if you do not have sufficient access rights to the original folder.) If you create a new folder, you are asked whether you like to have the old folder contents copied to the new location. See also section 3.

All Options settings are stored in the Windows registry.

4.5.6 The Help Menu

- **Contents**: show the contents section of the help file.
- **Index:** shows the help index.
- **User interface**: shows help on the user interface
- **Arithmetics**: shows help on the arithmetic features of the software, as is relevant for use of the script language.
- **Word at cursor**: shows help related to the keyword at the cursor position.
- **Display the PDF manual**: opens the PDF manual, which you may print out.
- **Visit the RP Photonics website**: uses your default browser to display that website.
- **Call technical support**: gives you information on technical support.
- **About**: opens the About window, showing information on the software version and the user license.

4.6 The Script Editors

In the upper left area, the software contains a tab control for one or several script editors. These can be used as any standard plain text editor. Their features are:

- Ctrl-C, Ctrl-V, Ctrl-X: copy, paste, cut operations with the Windows clipboard
- Ctrl-X, Ctrl-Y: undo, redo the last operation
- Ctrl-L: delete the line at the cursor position
- Ctrl-F: find some text
- Ctrl-Alt-F: find and replace some text
- F3: find next occurrence of text
- Ctrl-A: mark the whole script
- Ctrl-G: go to a certain line number
- Ctrl-K + digit (1 .. 9): move to line beginning with **`diagram 1`** (for example), so that one can quickly find the commands for preparing a certain graph.
- Ctrl-K + letter (a .. z): for example, for the letter “a”, move to the line beginning with **`;a:`**. Such a line is a comment, i.e., it is ignored when the script is executed, but you can use it for marking certain positions which you can then quickly reach with such a shortcut.
- F1: show the help contents.
- F2: show help on the keyword at the cursor position

See the Edit menu (section 4.5.2) for more editing commands. The font name and font size can be modified in the dialog **Options | Options**.

4.7 The Log Area

When a script file is executed, some information is written to the log area, which is displayed below the editor area and can be very useful for debugging purposes:

- Various commands produce additional information there. For example, when a graph is displayed in a diagram, it is displayed in the log area in which y range the calculated values are. This can be useful, for example, when a graph is not visible because its values are outside the range of the coordinate system.
- If any errors occur during script execution, these are displayed in the log area. This is useful for finding the cause of an error.
- The command `dump` can dump variables, arrays and functions to the log area. Examples:
    - `dump variables` dumps all variable values.
    - `dump arrays` lists all array names (but not all the values of array components)
    - `dump functions` lists all user-defined functions.
    - `dump all b*` lists all definitions beginning with “b”.
    - `dump variables alpha* to output` dumps its results to the output area (see section 4.8) instead of the log area.

During execution, the background color of that area changes. If the execution is not successful, the log area remains red until a script is executed again.

4.8 The Output Area

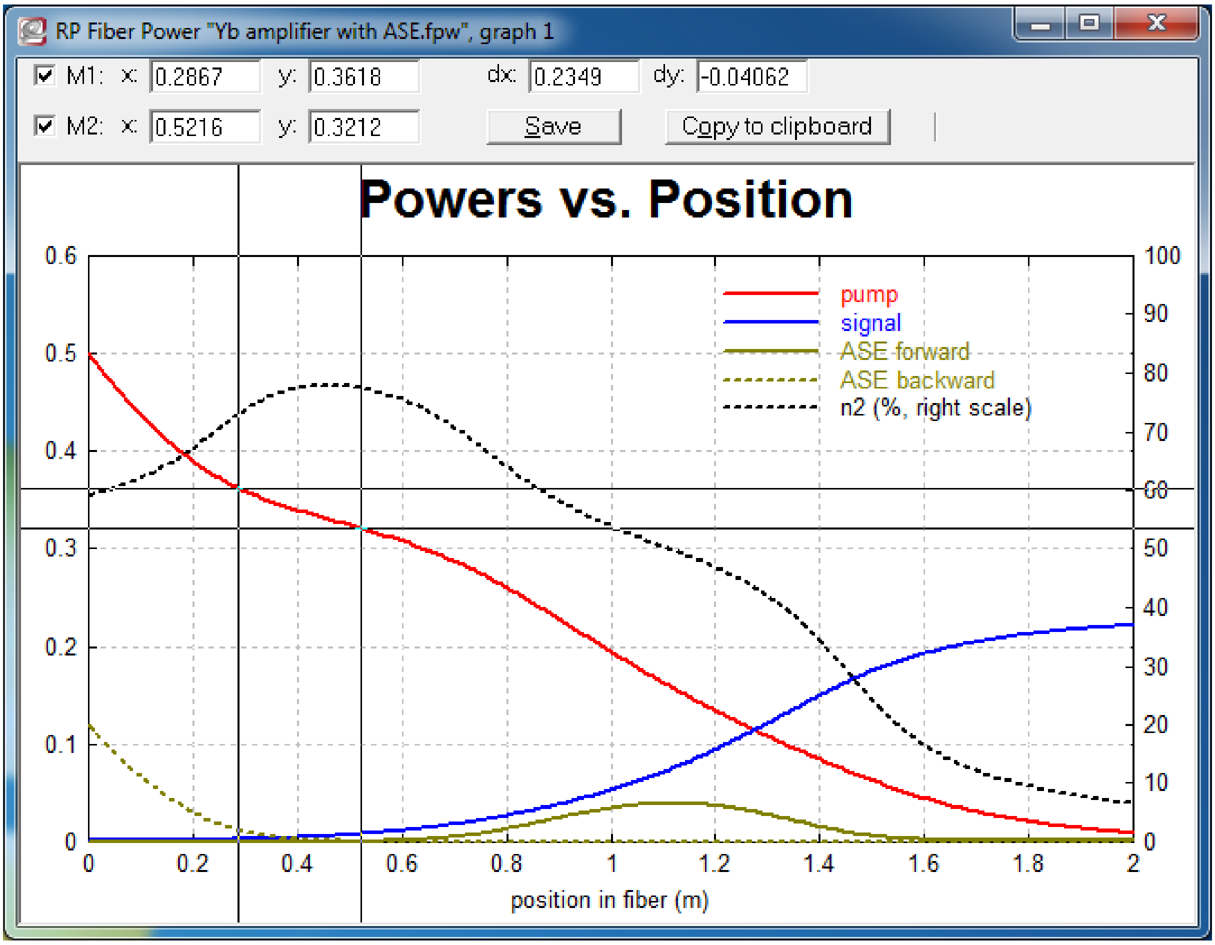
The `show` command (see section 5.17) in a script can be used to display results in numerical form in the output area. That area is found on the lower right part of the main form.

There is the function `showoutput()`, which can also display a numerical or string value in the output area, and can be called within any mathematical expression.

You may not only inspect results in the output area, but also copy parts of them to the clipboard with Ctrl-C.

4.9 The Graphics Windows

When **RP Fiber Power** creates plots, these are shown in separate graphics windows, which provide some additional controls. The screen shot below shows an example.



RP Fiber Power "Yb amplifier with ASE.fpw", graph 1

☑ M1: x: 0.2867 y: 0.3618 dx: 0.2349 dy: -0.04062

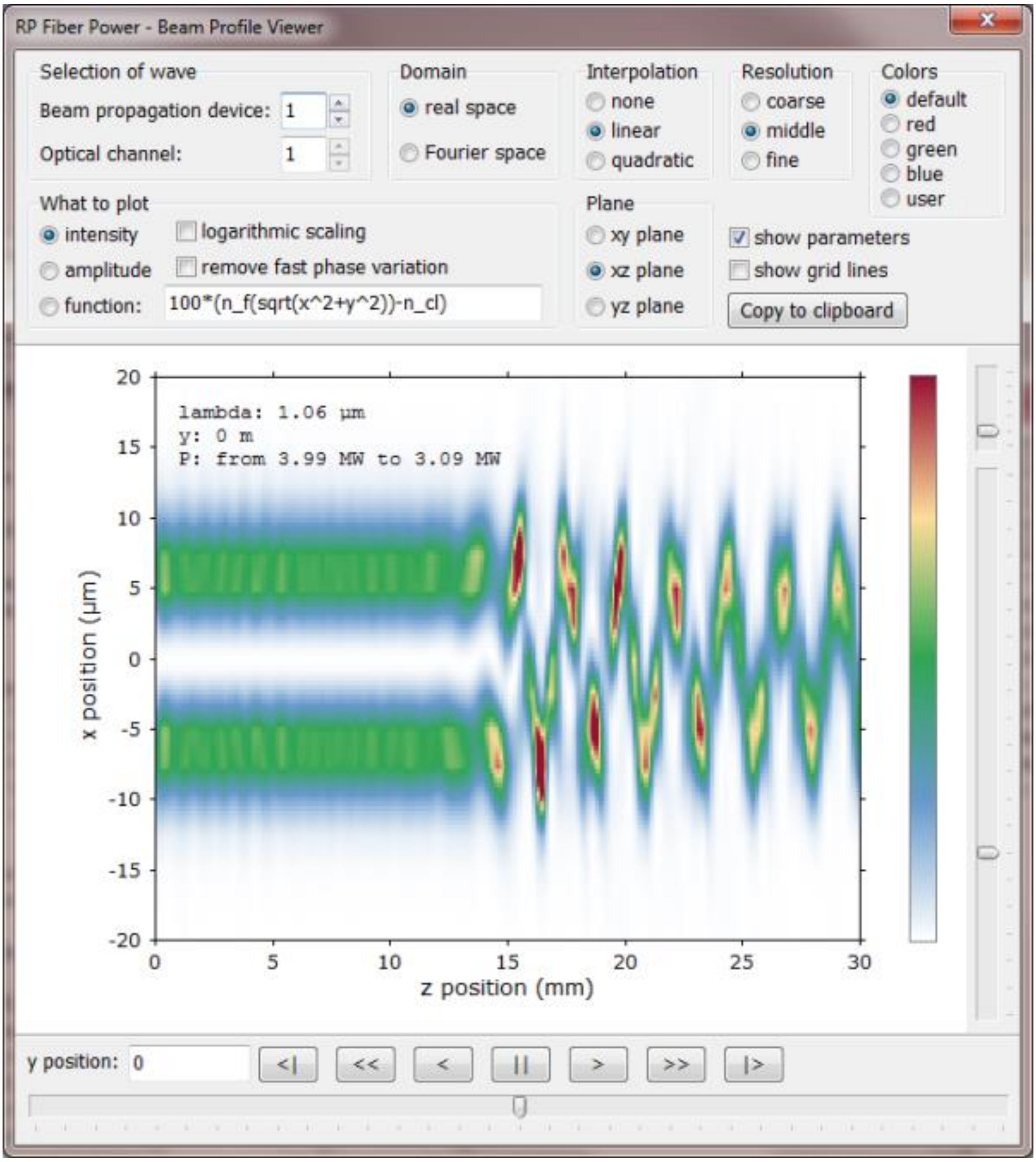
☑ M2: x: 0.5216 y: 0.3212

The controls “M1” and “M2” belong to markers, which can be switched on with the check boxes. The marker positions can be changed either by clicking into the diagram (left button for M1, right button for M2) or by changing the numbers and pressing Return. The fields “dx” and “dy” display the differences of the marker coordinates, allowing to measure distances.

The button “Save” allows to save the graphics as a .gif or .png file. After pressing the button “Copy to clipboard”, one can paste the graphics e.g. into a Word document.

4.10 The Beam Profile Viewer

For simulations of beam propagation, there is an interactive beam profile viewer, which can be called via the main menu with **View | Beam profile viewer** (or with Ctrl-B).



RP Fiber Power - Beam Profile Viewer

Selection of wave
Beam propagation device: 1
Optical channel: 1

Domain
● real space
○ Fourier space

Interpolation
○ none
● linear
○ quadratic

Resolution
○ coarse
● middle
○ fine

Colors
● default
○ red
○ green
○ blue
○ user

What to plot
● intensity
○ amplitude
○ function: 100*(n_f(sqrt(x^2+y^2))-n_cl)
□ logarithmic scaling
□ remove fast phase variation

Plane
○ xy plane
● xz plane
○ yz plane

☑ show parameters
□ show grid lines

```
lambda: 1.06 µm
y: 0 m
P: from 3.99 MW to 3.09 MW
```

y position: 0

The controls work as follows:

- Beam propagation device: one of the devices defined in the last run can be selected here. If you enter 0, you can access beam profiles which have been saved with the function **`bp_store_profile()`**.
- Optical channel: select one of the optical channels of the selected beam propagation device.
- Domain: intensities or amplitudes can be displayed either in real space or in Fourier space. The latter means that a Fourier transform is applied to the data in the *x*-*y* plane (but not in *z* direction). If the profile in the *x*-*y* plane is shown, in the Fourier space one essentially sees the shape of the far field in real space.

- Interpolation: here, one can select if the color at one point is calculated based on the nearest point of the numerical grid or with a linear or quadratic interpolation.
- Resolution: With the setting “fine”, the color diagram is made pixel by pixel. A higher speed of the display is achieved with the setting “medium” or “coarse”.
- Colors:
  - For intensity values, one has a choice between a default scale (with multiple colors), a red, green or blue color scale, and a user-defined color scale based on the user-defined function named **`colorscale(I)`** (with the argument varying between 0 and 1).
  - For complex amplitude values, in the default color scale the different phase values are encoded as different colors, while the magnitude determines the color saturation. The user-defined scale is based on the function **`colorscale_A(A%)`**, which may also depend on amplitude and phase. The red, green and blue scales use only the amplitude information.
- What to plot: Here, one can determine how the color at each point in the diagram is calculated:
  - With “intensity”, it is based on the intensity (modulus squared of the complex amplitude). If “logarithmic scaling” is checked, the values are logarithmically rescaled such that values in a range of 30 dB can be displayed; weak satellite structures and the like are then more easily recognized.
  - With “amplitude”, it is directly based on the complex amplitude, i.e., it also shows phase information. If “remove fast phase variation” is checked, the fast variation in *z* direction ($k_{\mathrm{av}} z$) is removed; that can be helpful particularly when data in the *x*-*z* or *y*-*z* plane are displayed.
  - With “function”, one can display values based on the function of *x*, *y* and *z* specified here.
- Plane: One can select between the *x*-*y*, *x*-*z* and *y*-*z* plane. The position concerning the remaining coordinate (*z*, *y* or *x*) is adjusted with the slider bar at the bottom. That position can also be modified with the buttons near the bottom: one can move to the ends, to the middle, or in fine or coarse steps to the left or right.
- Show parameters: If this is checked, various beam parameters (depending on the selected plane and domain) are displayed.
- Show grid lines: If this is checked, grid lines are shown.
- Zoom bar (the upper vertical track bar on the right side of the diagram): With this setting, one can magnify the center part of the diagram, i.e., reduce the axis ranges of the coordinate system (in *x* and *y* direction only).
- Scaling bar (the lower vertical track bar on the right side of the diagram): Here, you can modify the scaling of the color values. The higher the setting, the less optical intensity is required to reach a certain color level.
- Copy to clipboard: This allows one to copy the diagram (not the whole form content) to the Windows clipboard, so that it can be pasted into a Word document or graphics program, for example.

On the right side of the diagram, the currently used color scale is displayed. In case of the default scale for complex amplitudes, the colors encode phase values, whereas the intensity influences the color saturation.

One can also click into the diagram in order to have cross-sections of the intensity profiles shown as gray curves.

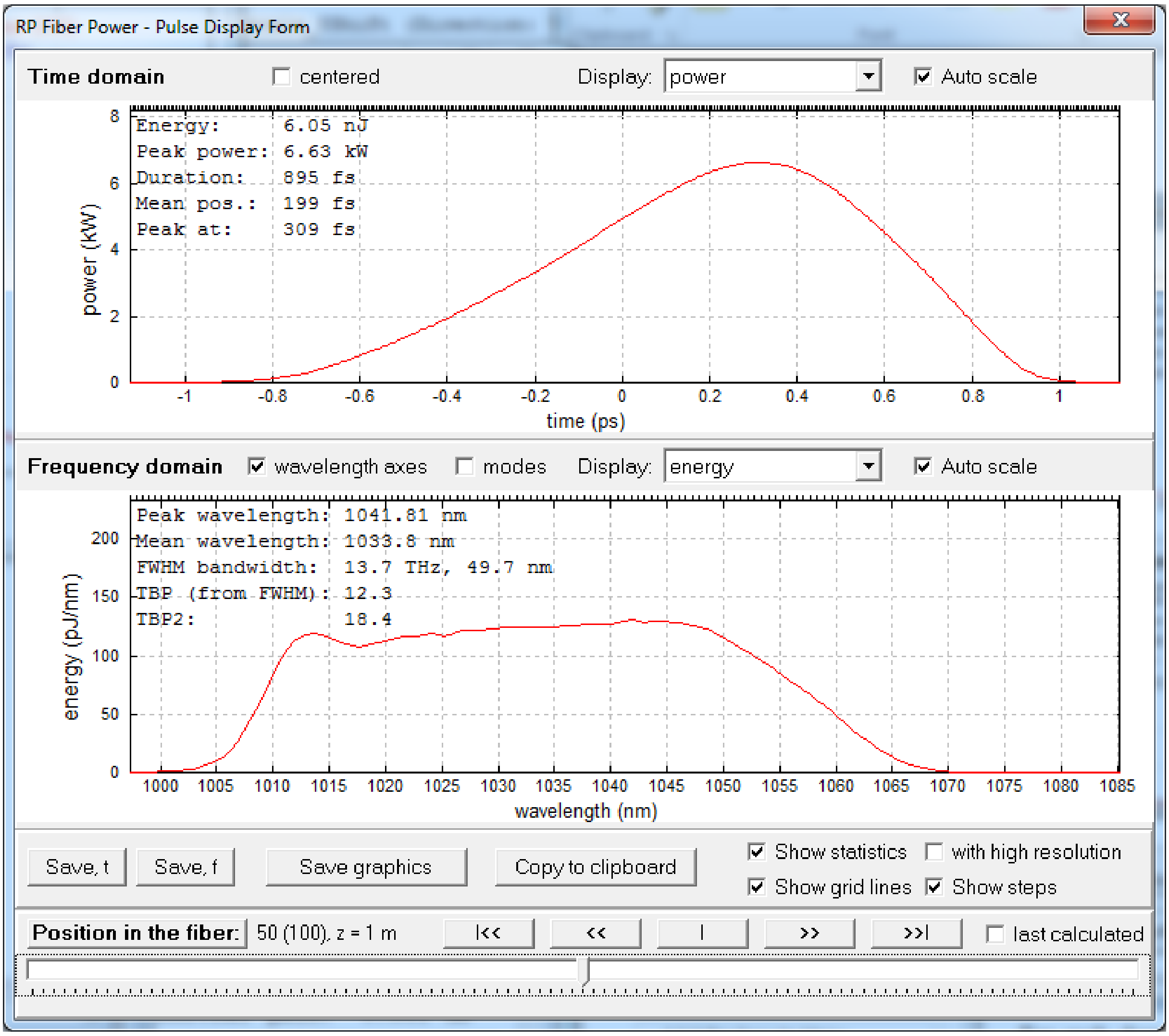
4.11 The Pulse Display Window

The interactive pulse display can be obtained via the main menu at **View | Pulse display window** (or with the shortcut Ctrl-D). If ultrashort pulse propagation has been simulated, this window can be used to display the pulses at different positions, as selected in the lowest part of the window:

- Normally, you see “Position in the fiber” displayed on the button. This displays a pulse at some position in the fiber, as obtained from the last pulse propagation through a fiber. The control bar then allows you to select any grid position in the fiber. You may use the buttons to jump to certain locations: **`I<<`** = left side of the fiber, **`<<`** = one position toward the left end, **`I`** = middle of the fiber, etc.
- If you click on the button “Position in the fiber:”, it changes to “Select stored pulse”. In this mode, you can display pulses which have been previously stored with the function **`store_pulse()`** (see section 5.15). However, if the control bar is in the very left position, the current pulse is displayed.

If you again click on the button, the window returns to the original position mode.

The upper diagram shows a property of the pulse in the time domain, the lower one in the frequency or wavelength domain. (One can switch between frequency and wavelength axis with the checkbox “wavelength axis”.)



□ centered Display: power ☑ Auto scale

```
Energy:      6.05 nJ
Peak power:  6.63 kW
Duration:    895 fs
Mean pos.:   199 fs
Peak at:     309 fs
```

☑ wavelength axes □ modes Display: energy ☑ Auto scale

```
Peak wavelength:  1041.81 nm
Mean wavelength:  1033.8 nm
FWHM bandwidth:   13.7 THz, 49.7 nm
TBP (from FWHM):  12.3
TBP2:             18.4
```

☑ Show statistics □ with high resolution
☑ Show grid lines ☑ Show steps

Position in the fiber: 50 (100), z = 1 m □ last calculated

For each domain, one can select which property should be displayed. In the **time domain**, there are the following properties available:

- optical power
- field amplitude (modulus of the complex amplitude)
- electric field (except if the field oscillations are too fast for the display; in that case, the amplitude is shown)
- phase (in the range –π to +π) (shown together with the power)
- continuous phase
- delta f: deviation of the instantaneous frequency from the center frequency (shown together with the power)
- delta l: like delta f, but in terms of wavelength
- AC_I: intensity autocorrelation
- AC_I_dB: same in decibels
- AC_E: interferometric autocorrelation

In the **frequency domain**, the following properties are available:

- energy (more precisely, an energy density in units like pJ/THz or pJ/nm)
- energy in dBm in 1 THz or 1 nm
- spectral amplitude
- spectral phase (in the range –π to +π) (shown together with the energy)
- continuous phase
- group delay (shown together with the energy)

Further possible settings are:

- Checkbox “centered”: center the pulse trace in the time domain.
- Auto scale: automatically set the horizontal and vertical scale of the diagram for the time or frequency domain.
- Arrow buttons (available only when “auto scale” is switched off): modify the axis scaling in the horizontal or vertical direction. If you click on such an arrow while holding the Ctrl key, you can shift the coordinate range.
- modes: in the frequency domain, show vertical lines instead of a curve (available only if the lines are not too close).
- Button “**Save, t**”: save the displayed pulse data in the time domain to a file. With “**Save, f**” you can do the same for the frequency domain.
- Button “**Save graphics**”: save the graphical content of the window to a GIF or PNG file.
- Button “**Copy to clipboard**” you can copy the graphical content of the form to the clipboard. From there you could paste it e.g. into a document.
- The buttons in the position section can be used set the position bar to certain positions: pulse outside the fiber on the left or right end, pulse at the left or right fiber end, or pulse in the middle of the fiber.

Further hints:

- With the keys **Home** and **End** you can move to the first and to the last calculated pulse, respectively.
- Click into a diagram in order to display a cross at the mouse cursor position and numerically display its coordinates. When you hold the **Ctrl** key at the time of clicking, the cross moves to the nearest data point of the displayed curve.
- If the pulse display window is opened before a script is executed, it can be used even before the whole pulse propagation has been calculated. It may just be that only a part of the pulses is calculated already, and the others cannot yet be displayed.
- The form settings can also be modified with various function calls in a script. See section 6.1.6 for the details.
- If the interaction of two pulses has been simulated with the function **`pp_fiber_2p()`** (see section 5.15), the pulse display form has an additional button to switch between the display of pulse 1 and pulse 2.

5 The Script Language of RP Fiber Power

5.1 General Remarks

RP Fiber Power takes all information for its output from a **RP Fiber Power** input script file (see section 1.1). This section describes the commands allowed in **RP Fiber Power** scripts.

RP Fiber Power scripts are plain text files with the filename extension `.fpw`. They can be edited not only with **RP Fiber Power**'s editor, but with any plain text editor. The general rules and most of the commands have been inherited from the program **JPLOT**. In this documentation, only those inherited features are explained which are relevant for normal use of the program **RP Fiber Power**.

Advantages of the script approach are:

- It is extremely flexible, allowing you e.g. to import and export data in various formats, set up new types of diagrams, or mathematically process and input or output data.
- You can simply copy and paste parts of the provided demo files, or of your earlier script files, in order to reuse code. In complicated cases, RP Photonics can send you some lines of code.
- A script file perfectly documents your work. When you read it later on, you easily see what you have done. (Ideally, you insert concise comments in your scripts.) You do not need to remember e.g. what settings you have done in what window.

Initially, you may prefer to use the input forms. In many cases, these can generate the required scripts, which are then executed. You may then have a look at these scripts and further refine them, if required.

5.2 General Rules for Scripts

A script file contains all the information the software needs to do its calculations and prepare its outputs. This can e.g. be

- numerical input data
- mathematical formulae for processing data
- definitions of output in text format or graphical form

Once a script is "executed", the following happens:

- All previously existing definitions (variables, functions, etc.) are deleted.
- The script file is read line by line. The commands are executed, so that various details such as variables, functions and graphical outputs are defined.
- If the script is successfully read to the end, i.e., no error occurs (e.g. a syntax error due to wrong input), and graphical output has been defined, that graphical output is now generated. For example, if a function graph has been defined with the `f:` command, the corresponding expression will only now be evaluated to calculate the details of the function plot.
- In any case, some information is written to the log area. This can give you important hints if something unexpected happened.

Some general rules for script commands are:

- Any command, variable or function definition, etc., has to occur at the beginning of a line. This means that commands must not be preceded by blanks (space characters).
- A command can extend over more than a single line; in that case, all lines but the first one have to be indented, i.e., begin with blanks.
- Lines beginning with a semicolon are interpreted as comments, i.e., ignored when a file is executed. The same holds for lines between one line with **(*** and one line with ***)**. These methods can be used to temporarily "comment out" some commands which are not required.
- The command **`include`** allows to read the content of some other file while processing a script file. For example, one could “outsource” the spectroscopic data of ytterbium in germanosilicate glass with

  ```
  include "Yb-germanosilicate.dat"
  ```

5.3 Overview on the Definition of a Model

The recommended way to define all parameters of the fiber is to combine all corresponding function calls in one function (spanning several lines), and then calling this function. A simple example is given, where it is assumed that various variables have been defined already (for example, **`r_c`** = core radius, **`N_dop`** = dopant concentration, etc.), and the specific details of various functions are explained in the following:

```
def_model():=
  begin
  set_fiber(L_f, N_z, gainsystem$);
  add_ring(r_c, N_dop);
  pump:=addinputchannel(P_p_in,l_p,'I_p',loss_p,backward);
  signal:=addinputchannel(P_s_in,l_s,'I_s',loss_s,forward);
  finish_fiber();
  end;
calc def_model()
```

Do not forget the last line – without it, the function **`def_model()`** would have been defined, but not executed.

It is possible later on to modify some of the parameters and call the function **`def_model()`** again in order to make the changes effective.

Within the definition of **`def_model()`**, you must obey to the following rules:

- First call **`set_fiber()`** in order to initiate the fiber definition and to define the basic parameters of the fiber.
- In case that there is an azimuthal dependence of the intensity profiles, call **`set_phi_steps()`** to define the number of azimuthal steps. (This was not needed in the example above.)
- In some cases, a rectangular grid is more desirable. This can be done with a function call like **`set_xy_steps(x_min,x_max,dx,y_min,y_max,dy)`**, which defines the minimum and maximum *x* and *y* values and also the step sizes.
- Then call **`add_ring()`** in order to define the radial structure of the dopant(s).

- Then define all optical channels by calling the functions **`addinputchannel()`** and/or **`addASEchannel()`**.
- Finally, call **`finish_fiber()`** in order to tell the software that the fiber definition is finished.

The following subsections give more details on defining fiber parameters and optical channels.

Note that all program-specific functions handle values with dimensions with the assumption that basic SI units are used. For example, fiber lengths as well as wavelengths are assumed to be specified in meters, optical powers in watts, etc.

5.4 Definition of Fiber Parameters

The basic fiber parameters are defined with a function call as the following:

```
set_fiber(L_f,N_z,gainsystem$)
```

The function **`set_fiber()`** has three parameters:

- the fiber length (here called **`L_f`**) in meters
- the number of numerical steps along the fiber

  (For example, for a 4 m long fiber and **`N_z`** = 40, the step size is 10 cm. A larger number of steps increases both the accuracy of the output powers and the resolution for fiber-internal quantities such as local powers, but requires longer computation times.)
- the gain system, defined as a string such as **`'Yb'`** (for ytterbium-doped gain media) or **`'Er'`** (erbium) for a simplified gain model; an empty string is given for more complicated gain systems, and **`'-'`** indicates an undoped fiber.

All such parameters may be constants, variables, or some mathematical expressions.

In case that a ring laser configuration is simulated, the function

```
make_ring()
```

(without parameters) has to be called.

Further, the transverse profile of the number density of the laser-active ions has to be defined. Normally, the concentration profile is assumed to be constant within certain rings around the fiber axis, but each ring can have a different concentration value. We define the ring structure by calling the function

```
add_ring(r,N_dop)
```

first for the inner circle with radius **`r`** and ion concentration **`N_dop`** (in ions per cubic meter), then with increasing **`r`** for all additional rings. Here, **`r`** is always the outer radius, while the inner radius is the outer radius of the previous ring. It is not necessary to define an outer region with zero doping concentration, as this would anyway not contribute to the gain or loss.

If an azimuthal dependence must be taken into account, one must define the number of azimuthal steps with a function call like **`set_phi_steps(4)`** *before* calling **`add_ring()`**. Note that the latter function defines some doping concentration for the whole ring, with no azimuthal dependence of that property. An additional azimuthal dependence of the dopant concentration can be defined with the function **`set_N_dop()`**, see section 5.5.2.

It is possible to define up to 30 rings and up to 128 steps in the azimuthal direction.

If a rectangular grid is to be used instead of a ring structure, one must define the minimum and maximum *x* and *y* values and also the step sizes with a function call like **`set_xy_steps(x_min,x_max,dx,y_min,y_max,dy)`**. Thereafter, one has to define the doping profile by calling **`set_N_dop(k, x, y, N)`** once for every grid segment, where **`N`** is the density of ions of type **`k`** and position **`x`** and **`y`**.

Example 1: High-NA Large-core Multimode Fiber

The case of a multimode fiber with relatively high numerical aperture is the simplest case, as the optical field intensities of all optical channels can be assumed to be constant within the core and zero outside the core. Here, a single function call like **`add_ring(r_core, N_dop)`** is sufficient.

Example 2: Step-index Single-mode Fiber with Uniform Doping

In that case, a simplified solution is to again define a single ring as before, and using e.g. Gaussian intensity profiles for the optical channels. However, the software then only calculates the degree of overlap of each optical channel with the core, while ignoring the variation of the optical intensity within the core.

A simple extension, which is more precise, is to use several rings having the same doping concentration. For 3 rings, e.g., this could read:

```
   for j:=1 to 3 do add_ring(r_core*(j/3), N_dop)
```

Although this defines exactly the same doping profile as before, the software then takes into account three rings with different optical intensities. This increases the accuracy, but also increases the computation time. Of course, an increased accuracy is not obtained when transition cross-section data are used which have been calculated from measured data using a simple top-hat approximation for the mode intensities. (This is not uncommon.)

Example 3: Few-mode Fiber

In this example, we consider a fiber supporting only the modes LP_{01} and LP_{11}. It may be sufficient to use only 4 azimuthal segments in a single ring. We can then first call **`set_phi_steps(4)`** to get 4 azimuthal segments, and then define a single ring with a call like **`add_ring(5e-6, 20e24)`**.

Redefining the Ring Structure

By calling **`add_ring(0,0)`** or **`set_xy_steps()`** one can reset the whole dopant structure (i.e., remove all previously defined rings and also reset the number of azimuthal segments, or change the rectangular grid) if it needs to be redefined.

It is also possible to recall the doping concentration of ion type **`i`** at a certain radius and azimuth with the function **`N_dop(i,r,phi)`** or **`N_dop(i,x,y)`**.

5.5 Definition of Spectroscopic Data

Next we need to provide the spectroscopic data. Section 3 explains where such data are stored and which data are provided with the software. In principle, you need to know about the following in this section only if you need to define fiber data yourself.

The software distinguishes two different cases of laser-active ions:

- ions with a simple level scheme, having only a single metastable state

- ions with a more complicated level scheme, which can involve more metastable levels and energy transfers

The required inputs in these cases are discussed in the next two subsections. Thereafter, another subsection summarizes some rules for spectroscopic data files.

5.5.1 Simple Level System

The underlying model for the laser-active ions is as follows:

- Each ion has a ground state and a single metastable level, which is the upper laser level. The fractional populations of these levels are in the following denoted by n_1 and n_2, with $n_1 + n_2 = 1$.
- The temporal evolution of the fractional excitation n_2 is described by the following differential equation:

$$\frac{\partial}{\partial t} n_2 = -\frac{n_2}{\tau_2} - q\, N_{\mathrm{dop}} n_2^2 + \sum_i \frac{\xi_i\, P_i}{A\, h\nu_i} \left[\sigma_{\mathrm{a},i} \left(1 - n_2\right) - \sigma_{\mathrm{e},i} n_2 \right]$$

  where τ_2 is the upper-state lifetime, q is a parameter describing quenching effects (e.g. caused via cooperative upconversion), ξ_i is the overlap factor for the optical wave with index i, P_i is the optical power, A is the core area, $h\nu_i$ the photon energy, $\sigma_{\mathrm{a},i}$ the absorption cross section, and $\sigma_{\mathrm{e},i}$ the emission cross section.

- The coupling of the dopant ions to the optical intensities is determined by so-called effective cross sections, which are wavelength-dependent. Another important spectroscopic parameter is the upper-state lifetime, i.e., the lifetime of the level 2.

In the software, the spectroscopic data are defined via various variables and functions:

- The upper-state lifetime of the laser-active ion must be defined as a variable with a name such as **`tau_Yb`** in the case of an ytterbium-doped gain medium. (The variable name is always constructed using the name of the gain system as given with the function **`set_fiber()`**, see section 5.4.)
- Quenching processes can be described with an additional variable, named e.g. **`q_Yb`** in the case of an Yb-doped medium, defined as the parameter q in the equation above.
- The absorption and emission cross sections must be defined as functions as e.g. **`s_a_Yb(l)`** and **`s_e_Yb(l)`** in the case of an ytterbium-doped gain medium. The function argument is the wavelength in meters.

The cross-section functions may be defined in different ways. One possibility is to use some analytical formula. Using data from G. C. Valley, Opt. Fiber Technol. 7, 21 (2001), we can use:

```
f(l,l_c,l_w,n):=exp(-abs((l-l_c)/l_w)^n)
s_a_Yb(l):=1e-24*
  (0.09*f(l,913e-9,8e-9,2)+0.13*f(l,950e-9,40e-9,4)
   +0.2*f(l,968e-9,40e-9,2.4)+1.08*f(l,975.8e-9,3e-9,1.5))
c:=2.9979e8
```

```
h:=6.626e-34
l0:=975e-9
kT:=1.38e-23*293
s_e_Yb(l):=s_a_Yb(l)*exp(h*c*(1/l0-1/l)/(kT))
tau_Yb:=1.5e-3
```

This applies for the ytterbium cross sections in glasses as used for the core of Er/Yb-doped amplifiers.

Another possibility is to use tabulated values. Example:

```
defarray s_abs_Yb[900,1100,1], s_em_Yb[900,1100,1]
readlist l, s_abs_Yb[l], s_em_Yb[l]:
  900, 725, 11
  901, 737, 12
  ...
s_abs_Yb(l):=1e-24*s_abs_Yb~[l/1e-9]
s_em_Yb(l):=1e-24*s_em_Yb~[l/1e-9]
```

The first line defines two arrays holding the data. The command **readlist** fills the data into the arrays. Finally, the cross-section functions are defined to get their results from the arrays. In the particular case, they also rescale the wavelengths (because these are in nanometers for the given arrays) as well as the results. The tilde (**~**) indicates that linear interpolation is used when function arguments occur which do not exactly correspond to certain array indices.

5.5.2 Extended Level System

If the active fiber contains laser-active ions which are too sophisticated to be described with the simplified model as in section 5.5.1, their properties are described in the software with somewhat more sophisticated function calls. For example, this part is relevant if one is dealing with a level system having more than a single metastable state or if energy transfers must be included. There can then be multiple types of ions, and each one has some doping concentrations (possibly with a radial dependence) and two or more electronic levels.

The data are provided in the following way:

- In the function **set_fiber()** (see section 5.4), an empty string is given for the type of ion.
- For each type of ion, call the function **def_ion(IonType$,l1,l2)**, where **IonType$** is a name for the ion (e.g. ‘Er’), **l1** is the number of the first level, and **l2** is the number of the last level. For example, if erbium with 3 levels and ytterbium with 2 levels are needed in the model, call **def_ion('Er',1,3)** and **def_ion('Yb',4,5)**. Note that the levels are numbered subsequently; it is thus clear later on that level 5 is the excited level of Yb. Within one ion system, the level numbers must be assigned in the order of increasing excitation energies, i.e., the ground state always comes first.
- If there is more than one ion type, or if the dopant concentrations are not radially symmetric, use the function **set_N_dop(k, r, phi, N)** or **set_N_dop(k, x, y, N)** to define the full dopant profiles. Here, **k** is the ion type (e.g. 2 for the second ion), **r** and **phi** are the radial and azimuthal coordinates, respectively, and **N** is the dopant concentration. For a rectangular grid, **r** and **phi** are replaced with **x** and **y**.
- Define all processes affecting the population values. There are five different types of processes, which are all explained with examples fitting to the erbium-ytterbium system as discussed above:

- **`def_spont(5, 4, 1/tau_Yb)`** defines a spontaneous transition from level 5 to level 4 with a given rate (number of events per second and ion in the starting level). Obviously, both levels must belong to the same type of ion.
- **`def_stim(4, 5, 's_a_Yb')`** defines a stimulated transition from level 4 to level 5, where the third parameter is the name of a function with which the transition cross section can be calculated. (That function must have the wavelength as a parameter.) The software recognizes automatically which optical channels (see section 5.6) are relevant for that transition, as it ignores those channels where the transition cross sections are zero. It is thus important that the given functions really return zero for wavelengths outside the range of their transition. The type of these transitions can be absorption (as in the example given) or stimulated emission (when the first level given is higher than the second one), and their effects on the optical fields are calculated automatically.
- **`def_quench(2, 1, 2, Q21)`** defines a quenching process, where a transition from level 2 to 1 occurs, and the rate is **`Q21`** times the square of the fractional population in the starting level. If the third parameter were 3 instead of 2, the rate would be **`Q21`** times the *cube* of the fractional population in the starting level.
- **`def_upcon(2, 1, 3, 2, U)`** defines an upconversion process, starting with two ions in level 2, where one goes to level 1 and the other one to level 3, and the rate of depleting the starting level is **`U`** times the square of the fractional population in the starting level. If the fourth parameter were 3 instead of 2, the rate would be **`U`** times the *cube* of the fractional population in the starting level. The rates of ions getting into the ending levels 1 and 3 are half the rate of depleting the starting level 2.
- **`def_transfer(5, 4, 1, 3, T_YbEr)`** defines an energy transfer, where one ion is transferred from level 5 to 4, and another ion is transferred from level 1 to 3. The rate of this process is in general different for the two ions involved (except, of course, if the energy transfer would involve only one type of ion): for the erbium ions, it is **`T_YbEr`** times the ytterbium concentration times the fractional populations in levels 5 and 1, whereas for ytterbium it is **`T_YbEr`** times the erbium concentration times the fractional populations in levels 5 and 1.

5.5.3 Rules for Spectroscopic Data Files

In principle, the files with spectroscopic data sets are simply include files (with extension **`.inc`**), which are read when called via an **`include`** command (see section 5.2), just as if their content had been copied into the main script. Therefore, the same rules apply as for all script code. However, there are some special rules which are relevant when such a data file is used in combination with the interactive forms (see section 4.4.3):

- If a data set should appear in the list of parameter sets in the interactive form for an active fiber, the corresponding data file must be stored in the spectroscopic data folder (see section 3).
- The filename should always begin with two letters indicating the type of active ion. For example, it could be "**`Yb-my special glass.inc`**" for an ytterbium-doped fiber.
- When using a gain model with a simple level system (see section 5.5.1), for example for an Yb-doped fiber, the file should define the following:

    - **`s_a_Yb(l)`** and **`s_e_Yb(l)`** are functions for the absorption and emission cross sections, respectively.
    - **`tau_Yb`** is the upper-state lifetime.
- When using a gain model with an extended level system (see section 5.5.2), a function **`def_ionsystem()`** needs to be defined, which contains the definitions of all processes causing transitions between different energy levels. When an interactive form is executed, it is checked whether the spectroscopic data file contains the definition of the function **`def_ionsystem()`**. If it is found, a call to that function will be inserted after the definition of the doping profile in the generated script.
- If additional values are supposed to be read from a data file into the interactive forms, these must be defined in a special format as explained in the following. For each field, there must be a line in the data file beginning with "**`;IF_`**", followed by the name of the field, "**`:=`**" and the field content. For example,

  `;IF_NoRings:=1`

  defines the number of rings,

  `;IF_r1:=r_core`

  defines the radius of the first "ring", and

  `;IF_conc12:=1.5e25`

  defines the concentration in ring 1 of ion type 2. Further possible variables are **`shape_p1`** (shape of the intensity profile of pump channel 1, e.g. **`Gaussian`**), **`w_p1`** (profile radius of pump channel 1), **`shape_ASE`** (shape of profile for ASE), **`w_ASE`** (profile radius for ASE), and **`loss_p1`** (parasitic losses of pump channel 1).

5.6 Input and ASE Channels

Each optical wave propagating in the fiber is described by a so-called **channel**, which can be an **input channel** or an **ASE channel**, and is characterized by parameters like wavelength, coupling strength, propagation direction, and others.

There are two different kinds of channels:

- **Input channels** allow to inject optical powers such as e.g. pump powers or signal input powers. They have a certain wavelength and (nominally) no bandwidth.
- **ASE channels** can not have inputs. Instead, they are fed with fluorescence from the excited gain medium, which is then of course also subject to the laser gain as well as the any losses.

In any case, each channel has a certain propagation direction, which can either be **`forward`** (a predefined variable with value 1) or **`backward`** (-1).

For referencing a channel later on (e.g. to retrieve its output power), each channel obtains a **reference number** when it is defined. For example, the third defined channel obtains the reference number 3.

After all channel definitions, the function **`finish_fiber()`** has to be called. Before that, it is not possible call functions for calculating optical powers and the like.

Normally, all channels are defined at the beginning of the script, and not modified during the calculation, except that input powers may be modified. It is possible, however, to redefine channels later on by removing all defined channels with the function **clearchannels()** and again using functions to defined all channels. This may be desirable e.g. when the number of ASE channels has to be increased for a final plot after doing more approximate (but faster) calculations with fewer ASE channels initially.

Particularly in lasers, it occurs that forward- and backward-propagating channels are coupled to each other. See section 5.6.3 for details.

Further details are given in the following two sections.

5.6.1 Input Channels

An **input channel** is defined with the function **addinputchannel()**. Examples:

```
pump:=addinputchannel(P_p_in,l_p,'I_p',loss_p,backward)
signal:=addinputchannel(P_s_in,l_s,'I_s',loss_s,forward)
```

The variables **pump** and **signal** store the reference numbers (see section 5.6) corresponding to the two channels. We obtain a backward-propagating pump channel with the reference number 1 and a forward-propagating signal channel with the reference number 2. These values are stored in variables for later access to the channels (e.g., for retrieving powers or for modifying input powers).

The parameters of the function **addinputchannel()** are:

- the input power, e.g. seen as power of the left fiber end for the forward-propagating signal and the power at the right end for the backward-propagating pump
- the wavelength (in meters)
- the name of a function specifying the transverse dependence of the mode intensity
- the background loss (in dB/m) (not including absorption by the dopant)
- the propagation direction, which can either be **forward** or **backward**.

The function for the mode intensity must have either a single argument **r** (for a radial dependency only) or two arguments **r** and **phi** (radial and azimuthal coordinate). If a rectangular grid has been defined with the function **set_xy_steps()**, the arguments of the intensity function must be **x** and **y**.

As an example for a mode function, take the following for the pump wave:

```
w_p:=5 um
I_p(r):=exp(-2*(r/w_p)^2)
```

The input power of a channel can later be modified with the function **set_P_in(ch,P)**, where the first argument is the channel number and the second one the new input power. Example:

```
calc set_P_in(pump,P_p)
```

There are similar functions for modifying other parameters: **set_lambda(ch,l)** for modifying the wavelength, **set_dlambda(ch,l)** for modifying the bandwidth of an ASE channel, and **set_loss(ch,lo)** for modifying the parasitic losses.

5.6.2 ASE Channels

An **ASE channel** is defined with the function **addASEchannel()**. Example:

```
ASE_fw:=addASEchannel(l_s, 10e-9, 1, 'I_s', 0, forward)
```

```
ASE_bw:=addASEchannel(l_s, 10e-9, 1, 'I_s', 0, backward)
```

The result values are the channel reference numbers (see section 5.6), in the same way as for the function **addinputchannel()**. The parameters are:

- the wavelength (in meters)
- the bandwidth (in meters)
- the number of spatial modes (e.g. 2 for a single-mode fiber with two polarization directions)
- the name of a function specifying the radial dependence of the mode intensity
- the background loss (in dB/m)
- the propagation direction

The function for the mode intensity must have either a single argument **r** (for a radial dependency only) or two arguments **r** and **phi**.

ASE channels have no inputs, but are fed by spontaneous emission.

Usually, one will have a whole array of ASE channels in order to properly sample the whole ASE spectrum. An example for the used code is given in the following:

```
l1_ASE:=960 nm  { minimum ASE wavelength }
l2_ASE:=1080 nm  { maximum ASE wavelength }
dl_ASE:=5 nm  { ASE bandwidth in m }
defarray c_ASE_fw[l1_ASE,l2_ASE,dl_ASE]
defarray c_ASE_bw[l1_ASE,l2_ASE,dl_ASE]
w_ASE:=5.5 um
l_s:=0
I_ASE(r):=exp(-2*(r/w_ASE)^2)
calc
  for l:=l1_ASE to l2_ASE step dl_ASE
  do begin
     c_ASE_fw[l]:=addASEchannel(l,dl_ASE,1,'I_ASE',l_s,forward);
     c_ASE_bw[l]:=addASEchannel(l,dl_ASE,1,'I_ASE',l_s,backward);
     end;
```

Here, one first defines a range of ASE wavelengths and the width of the individual ASE channels. Then one defines two arrays to store the reference numbers of all ASE channels. Finally, one defines the channels.

5.6.3 Reflections at the Fiber Ends

There is the function **set_R(channel,R1,R2)** to define the reflectivities of a certain forward-propagating channel at both fiber ends. This is mostly used for modeling lasers.

For example, a fiber laser could be defined with:

```
pump:=addinputchannel(P_p_in,l_p,'I_p',0,forward)
signal_fw:=addinputchannel(P_s_in,l_s,'I_s',0,forward)
signal_bw:=addinputchannel(P_s_in,l_s,'I_s',0,backward)
calc set_R(signal_fw,1,0.90)
```

Here, the reflectivities for the laser wavelength are 100% at the left end (with the pump input) and 90% at the right end (for the laser output). In effect, the call of **set_R()** coupled the two channels to each other. (The backward channel **signal_bw** is not explicitly specified, but automatically recognized as the corresponding channel, because it has the same wavelength.)

One can also define localized losses inside and outside the reflector (see the next section).

Note that the function **set_R()** must be called before calling **finish_fiber()**.

For ultrashort pulse simulations (section 5.15), the end reflectivities are ignored.

5.7 Localized Internal and External Losses

With a function call like **set_loss_int(channel,li1,li2)** one can define localized power losses, which apply on the way between the left or right fiber end, respectively, and the corresponding end mirror. In other words, these losses become effective before and after reflection at the end mirror (even in a ring resonator).

Similarly, with **set_loss_ext(channel,le1,le2)** one can define localized power losses, which apply outside the end mirrors.

In any case, the loss values must be between 0 and 1. For example, 0.1 would mean 10% power loss.

The input power to the fiber for a forward-propagation optical channel will be **P_in * (1 – le1) * (1 – R1) * (1 – li1)**, where **P_in** is the input power and **R1** is the reflectivity at the left side. The output power of a forward-propagating optical channel will be **P_fiber * (1 – li2) * (1 – R2)* (1 – le2)**, where **P_fiber** is the power at the fiber end and **R2** is the reflectivity at the right side (see section 5.7).

For corresponding backward-propagating optical channels, these losses do not have to be set separately.

For ultrashort pulse simulations (section 5.15), the localized losses are ignored.

5.8 Retrieving Calculated Results

A number of functions can be used to retrieve all results from the calculations:

- **P(ch,z)** calculates the internal power of the channel with reference number **ch** at position **z** with linear interpolation.
- **P_out(ch)** is the output power of a channel. For a reflecting fiber end, it is the power just before the end times the transmission of the reflecting surface.
- **I(channel,z,r,phi)** calculates the intensity within the fiber at the longitudinal position **z**, radius **r** and azimuth **phi**. For a rectangular grid, **r** and **phi** are replaced with **x** and **y**. If a negative **z** value is given, the intensity is calculated for 1 W of optical power.
- **sp_gain(ch)** is the internal single-pass power gain (in decibels) of a channel.
- **gs_abs(ch)** is the ground state absorption (in decibels per meter) of a channel (i.e., the absorption of the fiber in the non-excited state).
- **NF(ch)** is the noise figure of a signal or ASE channel.
- **n(z,l)** is the fractional excitation of level **l** of the laser-active ions at position **z**, averaged in the transverse dimension, with linear interpolation in **z** direction.

- **n_tr(z,r,phi,l)** is the fractional excitation of level **l** of the laser-active ions at a given position **z**, radius **r** (without interpolation) and azimuth angle **phi**. For a rectangular grid, **r** and **phi** are replaced with **x** and **y**.
- **n_av(l)** is the fractional excitation of level **l**, averaged over the whole fiber.
- **E_sat(ch)** is the saturation energy for some optical channel. (It is calculated for the ring which has the highest intensities and thus shows the strongest saturation effects.)

The call of such a function automatically starts the calculation of the required data, as necessary. For example, the first call of the function **P()** calculates the powers of all channels along the fiber and will thus require some time; subsequent calls can then use the already calculated data. The data are automatically recalculated e.g. when a change of an input power invalidated the previous results.

With **set_P_in(ch, P)** you can modify the input power for channel **ch**. If you call one of the functions above thereafter, these data are automatically recalculated.

5.9 Modifying Model Parameters

If some model parameters need to be modified, this can always be done by modifying the corresponding variables and doing the whole model definition again. This is not too cumbersome if a function has been defined to do the whole model definition (see function **def_model()** in section 5.3). As an example, let us assume that we need to modify some quenching parameter **q** of the laser-active ions. We can simply modify the corresponding variable and then call **def_model()** again.

Some modifications can be done without redefining the whole model:

- The doping concentration values can be changed with the function **set_N_dop()** as explained in section 5.5.2.
- The fiber length can be modified by calling the function **set_L(L)** with a new length value **L**.
- All channel definitions can be removed with **clear_channels()**.
- The input power of a channel can be modified with **set_P_in()**.

5.10 Switching Between Different Devices

With a function call like **set_device(j)** one can switch between different amplifier or laser devices, where **j** is a number between 1 and 10. By default, the software always uses device 1. If one switches to another one, one can again define fiber properties, channels, input powers, etc., and calculate outputs. One can then at any time switch between the different devices in order to modify some parameters or calculate some outputs.

This feature can be useful, for example, when analyzing multi-stage amplifiers. Each stage is then considered as one device. It is possible, for example, in a diagram to plot the output power of the second stage as a function of an input power of the first stage. This is demonstrated in a demo file (see section 8.16).

Another possible use is to switch between different versions of an amplifier or laser design. This may be more convenient than using separate scripts for these devices.

5.11 Calculation of the Noise Figure

For optical amplifiers, it is often of interest to calculate the so-called noise figure. This is defined as a factor which tells how much higher the noise power spectral density of the amplified output is compared to the input noise power spectral density times the amplification factor, assuming that the input noise is at the shot noise level. (See http://www.rp-photonics.com/noise_figure.html for details.) Often it is specified in decibels.

RP Fiber Power allows one to calculate the noise figure of some ASE channel with the function `NF(ch)`, where `ch` is the channel number of the signal. This noise figure is a factor (ratio of noise powers); to convert it to decibels, take `10*lg(NF(ch))`.

If a signal channel is specified instead of an ASE channel, the software searches for an ASE channel, the wavelength interval of which contains the signal wavelength, and which has the same propagation direction. The noise figure is calculated for a single pass, not taking into account reflections at the ends.

In order to become familiar with the concept, you may test the following cases:

- For a high-gain four-level amplifier without background loss, the noise figure must be ≈3 dB.
- For a quasi-three-level amplifier, the noise figure is higher. For backward pumping, the noise figure is better than for forward pumping.
- If there is only 10 dB background loss but no amplification, the noise figure is 10 dB. In that case, the signal is attenuated, while the noise stays at the shot noise level.

5.12 Using the Mode Solver

RP Fiber Power contains a mode solver, which can calculate the properties of all guided modes of the fiber from a given refractive index profile. This feature can be used as follows:

- First, define a function like `n_f(r)` which gives the refractive index profile. (See below for some examples.)

  (Note that one cannot define the function `n(r)` since the function name `n` is reserved for a predefined function retrieving excitation densities.)
- Then, associate that refractive index function with the fiber by calling the function `set_n_profile("n_f", r_core)` where the first argument is the function name and `r_core` is the core radius. The software assumes that the refractive index stays constant outside this radius.

  It is allowed to modify the index profile later on in order to see how that affects the modes. One just has to call the `set_n_profile()` function again.

  In case that a script contains more than one fiber device, these can have different mode profiles. Simply use different function names for those, such as `n1` and `n2`.
- Finally, you can retrieve all mode properties with the following functions:
  - `ModeExists(l, m, lambda)` returns 1 if the corresponding mode exists, and 0 otherwise.
  - `A_lm(l, m, lambda, r)` is the amplitude for mode indices `l` and `m` for the wavelength `lambda` at the radius `r`. This has to be multiplied with `cos(l phi)` or `sin(l phi)` to obtain the full mode function. That function (including the

azimuthal factor) is normalized such that the integral of its squared modulus over the whole area (core and cladding) is unity.

- **`A_lm_xy(l, m, lambda, x, y)`** is the amplitude for the position with Cartesian coordinates **`x`** and **`y`**, assuming the **`cos(l phi)`** dependence. For obtaining the **`sin(l phi)`** dependence, take **`A_lm_xy(-l, m, lambda, x, y)`** (i.e., the value for negative **`l`**).
- **`I_lm(l, m, lambda, r)`** is the intensity for 1 W of optical power. It is the squared modulus of **`A_lm(l, m, lambda, r)`**. It has to be multiplied with **`cos(l phi)^2`** or **`sin(l phi)^2`**.
- **`I_lm_xy(l, m, lambda, x, y)`** is the intensity for the position with Cartesian coordinates **`x`** and **`y`**, assuming the **`cos(l phi)`** dependence for positive **`l`** and the **`sin(l phi)`** dependence for negative **`l`**.
- **`beta_lm(l, m, lambda)`** calculates the propagation constant of a mode, i.e., the phase change per unit propagation distance. (This is not relevant for the power calculations, but may be of interest for other purposes.)
- **`w_lm_x(l, m, lambda)`** calculates the mode radius in *x* direction for the mode function with the $\cos l\varphi$ dependence (or the radius in the y direction for the mode function with the $\sin l\varphi$ dependence). This is based on the second moment of the intensity distribution.
- **`w_lm_y(l, m, lambda)`** calculates the mode radius in *y* direction for the mode function with the $\cos l\varphi$ dependence (or the radius in the x direction for the mode function with the $\sin l\varphi$ dependence).
- **`A_eff_lm(l, m, lambda)`** calculates the effective mode area of a mode, defined as

  $$A_{\text{eff}} = \frac{\left(\int |E|^2 \, \mathrm{d}A\right)^2}{\int |E|^4 \, \mathrm{d}A} = \frac{\left(\int I \, \mathrm{d}A\right)^2}{\int I^2 \, \mathrm{d}A}$$

  Note that this is *not* π times the product of the mode radii in *x* and *y* direction.
- **`A_eff_lm(l, m, lambda)`** calculates the effective mode area of a mode. Note that this is *not* π times the product of mode radii in *x* and *y* direction.
- **`p_core_lm(l, m, lambda)`** calculates the fraction of the mode power which is guided within the core.
- **`NoModes(lambda)`** returns the number of modes, not taking into account polarization. For example, this would be 1 for a single-mode fiber (with the LP_{01} mode only) or 2 for a two-mode fiber with LP_{01} and LP_{11}.
- **`NoModes_h(lambda)`** is similar to **`NoModes(lambda)`**, but it takes into account the different orientations (the cos and sin versions) for modes where **`l`** is not zero. For example, it is 3 for a fiber with the modes LP_{01} and LP_{11}, because the latter has a cos φ and a sin φ version.
- **`l_max(lambda)`** returns the maximum *l* value for all modes.

    - **`m_max(lambda)`** returns the maximum *m* value for all modes.
    - **`cut_off(l, m)`** returns the cut-off wavelength of a mode, i.e., the wavelength below which the mode exists.
    - Further functions are available for calculating the group velocities and chromatic dispersion; see for the function reference in section 6.1.6.

  As soon as one of these functions is called for the first time for some wavelength value, the properties of all guided modes for that wavelengths are calculated and stored. Thereafter, one can very quickly access the properties of other modes at that wavelength – even if mode properties for other wavelengths have been calculated in the meantime.

- A calculated mode intensity profile can also be directly used in the definition of an optical channel. Instead of the name of a regular intensity profile function, one can specify, for example, **`"I_lm(1,1,cos)"`**, where an LP_{11} mode with a cos φ dependence is selected. For modes with $l = 0$, the φ dependence must be omitted; for example, use **`"I_lm(0,1)"`**. The software then automatically passes the wavelength, radial coordinate and azimuth coordinate to the mode solver.

Some examples and more details for refractive index functions:

- A simple step-index fiber design can be defined as follows:

  ```
  n_cl:=1.44  { cladding index }
  n_co:=n_cl+0.35e-3  { core index }
  n_f(r):=if r<=r_core then n_co else n_cl
  ```

  If one knows the numerical aperture of the fiber, one can calculate the core index from this:

  ```
  n_co:=sqrt(n_cl^2+NA^2)
  ```

- A refractive index function can also have a second argument, which is the wavelength. In that way, one can take into account the material dispersion. Example:

  ```
  n_f(r,lambda):=
    if r<=r_core
    then n_germanosilicate(lambda)
    else n_silica(lambda)
  ```

  where it is assumed that the functions **`n_germanosilicate(l)`** and **`n_silica(l)`** have been defined already (for example, using Sellmeier equations). It would also be possible to interpolate between functions for silica and germania according to a known profile of the germanium concentration.

- A parabolic refractive index profile could be defined as

  ```
  r_core:=10e-6  { radius of the parabolic region }
  dn_max:=50e-3  { maximum increase of index }
  n_f(r):=if r<=r_core then n_cl+dn_max*(1-(r/ r_core)^2) else n_cl
  ```

- In some cases, one has a text list containing refractive index data. For example, the list may contain equidistant data pairs. Such a profile can be used to define a function with the following commands:

  ```
  n_cl:=1.44  { cladding index }
  r_core:=10e-6  { core radius }
  dr:=2e-6  { radial resolution }
  ```

```
defarray n_f[0,r_core,dr]
readlist r, n_f[1e-6*r]:
 0, 1.442
 2, 1.444
 4, 1.444
 6, 1.443
 8, 1.441
 10, 1.44
n_f(r):=if r<=r_core then n_f~~[r] else n_cl
```

The index function uses the tabulated values in the core region only, and interpolates them.

See also the demo scripts described in sections 8.1, 8.11, 8.12 and 8.14.

5.13 Simulating Beam Propagation

The principle model used for beam propagation is explained in section 2.8. In the following, you can learn how to implement such simulations.

Define a Beam Propagation Device

With a function call like **bp_set_device(1)** you can define a beam propagation device, which is referenced with a number between 1 and 100. Following function calls for defining a grid or optical channels, for example, always refer to the currently set beam propagation device.

If you use only a single device, you do not need to use that function; device 1 will be used by default.

Define the Propagation Direction

By default, the fields are always propagated from *z* = 0 upwards. However, it is possible to reverse the propagation direction by calling **bp_set_direction(backward)**. The effect of that is that the initial field will thereafter always be stored for the *maximum z* position, and that the fields will be propagated backwards. One may later on call **bp_set_direction(forward)** in order to get forward propagation again.

Define Curvature Radii

If the waveguide of the current beam propagation device is curved, one defines the radius of curvature in *x* and *y* direction. The curvature radii may depend on *z*; for example, a waveguide may be straight in the beginning and then exhibit a sharp bend. Such things can be defined with a function call like **bp_set_R('R_x(z)', '0')**. The two arguments indicate mathematical expressions for the curvature radius in *x* and *y* direction, respectively, which may depend on *z*. In the given example, the curvature radius in *x* direction is given by a previously defined function **R_x(z)**, whereas there is no curvature in *y* direction. (An infinite radius of curvature is specified as zero.)

Define the Numerical Grid

Once a beam propagation device has been set, its numerical grid can be defined with a function call like **bp_set_grid(x_max, N_x, y_max, N_y, z_max, N_z, N_s)**. Some details:

- The grid covers the *x* range from **-x_max to +x_max** with **N_x** steps. The step size is **2 x_max / N_x** (note the factor 2), and the number of steps must be a power of 2.

- Analogous rules apply to the *y* direction.
- The *z* range is from 0 to **`z_max`** with **`N_z`** steps, so that the *z* step size is **`z_max / N_z`**.

  For the beam propagation calculations, however, **`N_s`** sub-steps are done for each interval **`dz = z_max / N_z`**. One uses **`N_s`** larger than 1 if one requires finer numerical steps without storing all the intermediate amplitude profiles. Note that quantities like refractive indices or curvature radii are by default recalculated only for every full *z* step, not for every sub-step.

  The value of **`N_s`** can also be changed automatically during the simulation. For that purpose, and for having the spatial factors updated in every sub-step, one uses the function **`bp_set_N_s()`** (see below).

Note that if the grid is redefined after defining optical channels, all optical channels are deleted, since they must use the same grid parameters.

Section 2.8.4 gives useful hints concerning the determination of appropriate values for the grid dimensions and grid resolution.

Define Reflecting Boundaries

By default, the numerical grid has periodic boundary conditions. For example, a wave reaching the boundary with the maximum *x* values will reenter the grid on the opposite side. That behavior is not physically realistic, but matters only if a significant optical power can reach the boundaries.

One can make the grid boundaries reflecting by calling **`bp_set_reflection('x')`** (for reflections in the *x* direction only), **`bp_set_reflection('y')`** or **`bp_set_reflection('xy')`** (for reflections at all boundaries).

Another possibility is to define a reflecting surface via a Boolean expression which can depend on *x* and *y*. It must evaluate to a non-zero value for all points outside that boundary. For example, **`bp_set_reflection('(x/a_x)^2+(y/a_y)^2 >= 1')`** defines an elliptical reflecting boundary.

Note that such boundaries are highly reflecting only for sufficiently small *z* steps in the propagation. Otherwise, some field may penetrate the reflecting region. The reflecting regions are numerically realized simply by introducing a π phase shift (in each step) to all amplitudes, but not by enforcing zero amplitudes, as that would lead to some loss of optical power.

Define the Kerr Nonlinearity

If self-phase modulation and cross-phase modulation via the Kerr effect needs to be taken into account, one can set the nonlinear index n_2 with a function call like **`bp_set_n2('n2(x,y)')`**. The argument can be a function of *x* and *y*; for example, one may have a stronger nonlinearity in the fiber core than in the cladding.

Different optical channels (see below) can interact via cross-phase modulation (XPM). The relative strength of that interaction depends on the polarization directions. If all involved waves have the same polarization, that relative strength is 2 (which is the assumed default value). For two cross-polarized optical channels, one can set the reduced factor 2/3 with **`bp_set_XPM(2/3)`**.

Caution: do not confuse the mentioned functions with the functions **`set_n2()`** and **`set_XPM()`**, which apply to ultrashort pulse propagation models (see section 5.15).

Define the Laser-active Doping

If the fiber contains laser-active ions or ions serving as saturable absorbers, their doping concentrations have to be defined with a function call like **`bp_set_N_dop('Yb', 'N_dop(x,y,z)')`**. The first argument indicates the type of ion, and the second argument gives a mathematical expression which results in the doping concentration, which can depend on the *x*, *y* and *z* coordinates.

Note that the laser-active doping must be defined before any optical channels are defined.

In beam propagation simulations, the software uses the simple gain system as explained in section 5.5.1. Essentially, the laser-active ions have only a single metastable level, which is the upper laser level. If the laser-active ions are of the ytterbium type (Yb), for example, the transition cross sections must be defined as functions **`s_a_Yb(l)`** and **`s_e_Yb(l)`** (for absorption and emission, respectively), and the upper-state lifetime is stored in the variable **`tau_Yb`**. These data are normally read from a spectroscopic data file (see section 5.5).

Note that more sophisticated gain models, e.g. involving energy transfers and upconversion, cannot be used in the context of beam propagation, but only in power propagation models (see section 2.2).

All optical channels (see below) interact with the laser-active ions, as far as they have non-zero cross sections for the relevant wavelengths. In addition, one can define a pump intensity distribution with a function call like **`bp_set_I_p('I_p(x,y,z)', lambda_p)`**. Here, the first argument is an expression for calculating the pump intensity at any location in the fiber, while the second argument indicates the pump wavelength. The pump intensity is immediately calculated and stored for the whole grid.

There can be multiple types of such ions, for which the parameters can be set independently. This means that **`bp_set_N_dop()`** is called more than once, each time with a different ion name. Thereafter, one can call all the other functions described above to define further details for the corresponding ion type. One can also use **`bp_set_N_dop()`** to return to an already defined ion type (without changing it) by calling it with the second parameter (normally the doping concentration) being an empty string. This allows one, for example, to tell which ion type is meant for further calls of functions like **`bp_set_I_p()`** or **`bp_excitation()`**.

There are different ways how the interaction of light with the laser-active ions (or saturable absorber ions) is simulated:

- By default, the software always calculates the steady-state solution based on the local optical intensities of all optical channels and the pump wave. For that purpose, it calculates the local steady-state excitation densities of the ions and stores theses values.
- It is also possible to do dynamic simulations. When the beam propagation is done, the change of the excitation densities within a given time interval is simulated. For that, the time interval must be defined with the function call **`bp_set_dt(dt)`**, where the argument is the size of the time interval. The temporal step size should be so small that the excitations do not change substantially during one step. By setting a negative step size, one can totally suppress the back-action of optical fields on the excitations, which can be useful for some tests but also to save time in situations where that back-action is extremely weak anyway.

  One may later on change the temporal step size. For example, one can reduce it in a Q-switched laser simulation once lasing begins, as the fields then develop much faster.

If the time step is set to zero, this indicates that the steady-state values should be calculated (as is done by default). One may, for example, first use this setting, perform a single propagation to get the stored steady-state excitations as an initial state, and then use a positive temporal step size for simulating the further evolution under additional influences.

The initial excitation densities are all zero. They can be changed according to the given pump intensity distribution only. The function call **`bp_do_pumping(t)`** updates the excitation densities if the ions interact only with the pump wave for a time **`t`**, if that value is positive. If it is zero, the steady-state values (for pumping only) are calculated.

Another possibility is to directly set the excitation densities with the function call **`bp_set_excitation(x, y, z, n)`**, where the last argument is the fractional excitation (between 0 and 1).

At any time, one can retrieve the local excitation densities (more precisely, the fraction of excited ions) with the function **`bp_excitation(x, y, z)`**.

If the Kramers–Kronig effect (phase changes related to gain changes) should be included, one calls the function **`bp_set_alpha(alpha)`**, where the argument is the linewidth enhancement factor. The phase change in one *z* step is the linewidth enhancement factor times the amplitude gain.

Define the Optical Channels

One may define multiple optical channels of the currently selected beam propagation device with function calls like **`signal:=bp_define_channel(lambda)`**, where **`lambda`** is the vacuum wavelength of the channel, and the function result is the number of the channel (e.g., 2 for the second defined channel).

After that, one can do more definitions, which all apply only to the current optical channel:

- **`bp_set_n(func$)`** defines the refractive index profile. The argument is a string which defines a mathematical expression depending on *x* and *y*. For example, **`'n_f(sqrt(x^2+y^2))'`** would refer to a previously defined function of the radial coordinate.
- If the refractive index also depends on the *z* position, one can use **`bp_set_n_z(func$, test$)`** with a refractive index function which may in addition depend on *z*. The second argument is an expression which is used to determine whether the refractive indices for a certain *z* position must be recalculated: they are recalculated only if the value of the test expression changes. For example, it is always recalculated if the test expression is simply **`z`**, or it is recalculated only once per millimeter if it is **`trunc(z/mm)`**.
- One can apply a smoothing procedure to the refractive index values, where each value is replaced with a weighted average (using a Gaussian weight function) which includes neighbored values. This has little effect on already smooth profiles, but reduces certain numerical artifacts for step-index profiles. One can set the strength of the smoothing by defining the value of the radius of the above-mentioned Gaussian function with **`bp_set_n_smooth(r)`** where **`r`** can be between 0 (no smoothing) and 10 (strong smoothing); 1 means that mostly only the directly neighbored points are used for smoothing. Particularly in cases where the transverse resolution is high, and the actual index profile is not that steep, using larger values of the smoothing parameter is recommended.

- The value of **N_s** (number of numerical sub-steps per *z* step) can be changed automatically during the simulation (for each optical channel separately), e.g. according to the local power or peak intensity. For that purpose, one uses a function call like **bp_set_N_s(‘bp_I_max(z)/1e5’, u)**, which assigns a *z*-dependent expression (first argument) for recalculating **N_s** after each *z* step. The value must be between 0 (effectively taken as 1) and 1000. If the second parameter is not zero, the spatial factors (resulting from refractive index and curvature radii) will be recalculated in each sub-step. Note that **N_s** should not be changed too often, as every time some quantities need to be recalculated.

- **bp_set_loss(func$)** defines a linear loss profile. The argument is a function of *x* and *y*, and its values indicate the local propagation losses in units of dB/m. For example, with **bp_set_loss('((x^2+y^2)/(10 um)^2)^4')** one would define a loss which sharply rises at the edges of the grid.

  Note that such losses can be introduced to suppress reflections from the boundaries of the grid (see section 2.8.4 for more details). Also, it is possible to specify a negative loss, which means gain. (For saturable laser gain, however, one should use **bp_set_N_dop()**.)

- **bp_set_A_factors(z, 'f%(x,y)')** defines a complex factor, which will be applied to the complex field amplitudes in the spatial domain when a certain *z* position is reached. That can be used, e.g., to filter out cladding modes at some point, or to implement thin lenses or apertures in a free-space simulation. (Such factors can be defined for any *z* positions.) If the given *z* value is negative, these amplitudes will be applied after *every z* step.

- **bp_set_A_f_factors(z, 'f%(f_x,f_y)')** does the same as the function **bp_set_A_factors()**, but in the spatial Fourier domain.

- **bp_set_A0(func$)** defines the initial amplitude profile of the current channel as a function of *x* and *y*. For example, **'sqrt(P/(0.5*pi*w0^2)) * exp(-(x^2+y^2)/w0^2)'** would define a Gaussian function with power **P**, Gaussian radius **w0** and zero complex phase. Amplitudes are scaled such that their modulus squared is the optical intensity (power per unit area).

- **bp_set_A(z, func$)** defines an amplitude profile just as **bp_set_A0(func$)**, but at an arbitrary z position.

- **bp_copy_A(d,ch,z)** copies a field distribution from some beam propagation device **d** and channel **ch** at position **z** to the initial field distribution of the current propagation channel. (The involved optical channels must have the same wavelength.) One may, for example, copy the field at the end of the same propagation channel in order to start a further propagation without requiring more memory. If the two fields have different numerical grids (in terms of size or resolution), quadratic interpolation is used for calculating the new field values.

- **bp_set_color(s$)** defines the color setting in the profile viewer window (see section 4.10). The argument can be one of the possible settings the beam profile viewer: **'default'**, **'red'**, **'green'**, **'blue'** or **'user'**.

- **bp_set_label(s$)** sets a text label for the current channel, which can be displayed in the beam profile viewer.

Note that all these settings have to be done for every optical channel; for example, the refractive index profile needs to be defined for every optical channel, and may depend on its wavelength.

Usually, one does all these settings before using any functions for calculated quantities (see below). However, one can also set fiber properties later on; this will reset already calculated profile data, so that they will recalculated if needed later on.

One can switch between already defined channels using function calls like **`bp_set_channel(j)`**, where **`j`** is the channel number.

You can later remove optical channels by calling the function **`bp_remove_channel(j)`**. This is advisable if calculations for many wavelengths are done subsequently, since the accumulation of data for many channels could lead to a shortage of main memory.

Obtain Calculated Quantities

One can now obtain various calculated properties of the optical fields in the fiber:

- The function **`bp_A%(x, y, z)`** delivers the complex amplitude for the given position and the currently selected optical channel.
- **`bp_I(x, y, z)`** is the optical intensity (the squared modulus of the amplitude).
- **`bp_I_max(z)`** is the maximum optical intensity of a profile.
- **`bp_P(z)`** is the optical power, i.e., the intensity integrated over the full area.
- **`bp_x_m(z)`** is the mean *x* position of the beam profile for a given *z* coordinate. It is calculated as the “center of gravity” of the intensity distribution. In an analogous way, **`bp_y_m(z)`** is the mean *y* position.
- **`bp_w_x(z)`** and **`bp_w_y(z)`** calculate the beam radius in *x* and *y* direction, respectively, based on the second moments of the intensity profile.
- **`bp_A_eff(z)`** calculates the effective mode area as $A_{\text{eff}} = \frac{\left(\int I \,\mathrm{d}A\right)^2}{\int I^2 \,\mathrm{d}A}$. Note that this is *not* directly related to the beam radii based on the second moment of the intensity profile; the relation between these quantities depends on the shape of the intensity profile.
- **`bp_A_f%(fx, fy, z)`** delivers the amplitude distribution in Fourier space. This is calculated by applying a two-dimensional Fourier transform in the *x*-*y* plane to the amplitude distribution as obtained with **`bp_A%(x, y, z)`**.

  For example, an argument **`fx`** = 20000 would correspond to 20000 oscillations per meter in *x* direction. If the vacuum wavelength is 1000 nm, that would correspond to a propagation angle of (20000 / m) / (1 / 1 µm) = 20 mrad outside the fiber, considering only the *x* direction.

  Note that this function is limited to arguments below the Nyquist frequency; for example, the maximum **`fx`** value is 0.5 / **`dx`**, where **`dx`** is the resolution in *x* direction. If the spatial resolution of the simulation is chosen high enough, the Fourier-transformed amplitudes will usually get quite small for frequencies close to the Nyquist frequency.

- **`bp_theta_x(z)`** and **`bp_theta_y(z)`** calculate the half-angle divergence of the beam, if the beam profile at that location were released to vacuum. (Inside a glass, for example, the actual divergence angles are smaller. Note that the refractive index may vary within the glass, if it is inhomogeneous.) This calculation is based on the second moment of the amplitude profile in Fourier space.
- **`bp_M2_x(z)`** and **`bp_M2_y(z)`** calculate the beam quality M^2 factors in x and y direction, respectively. If the wavefronts are already known to be flat, as is the case e.g. for modes in lossless fibers, one can use the faster versions **`bp_M2_x_col(z)`** and **`bp_M2_y_col(z)`**.

By default, the coordinates are in such functions rounded to those of the grid points. However, one can obtain linear interpolation with a function call like **`bp_set_interpol(1)`** before calling the other functions. With **`bp_set_interpol(2)`**, one obtains quadratic interpolation, and the argument 0 would switch off the interpolation.

Store Beam Profiles

In some situations, one needs to store calculated beam profiles. For example, one may want to simulate multiple round-trips of a light beam through some device and store the beam profile after each round trip. For such purposes, one can create a storage region for some number of profiles, store profiles in that region, and recall such profiles:

- **`bp_set_storage(x_max, N_x, y_max, N_y, N)`** defines a store region, having a numerical grid with given extensions and number of points in x and y direction (as in the function **`bp_set_grid()`**). After that function call, beam profiles can be stored with an index between 0 and **`N`** (the last argument in the call above).
- **`bp_store_profile(z, j)`** stores the current beam profile at a given **`z`** position in the storage region, using the storage place **`j`**.
- **`bp_get_stored()`** (without any arguments) makes the stored profiles accessible with functions like **`bp_I(x, y, z)`** or **`bp_P(z)`** (see above).

In the beam profile viewer (section 4.10), the z coordinate represents the indices (0 … N) of the stored profiles.

Modify Settings of the Beam Profile Viewer

With certain functions, you can change settings of the beam profile viewer form, which one would normally change manually in that form:

- **`bpv_show(r)`** shows (for non-zero **`r`**) or hides the beam profile viewer.
- **`bpv_set_device(j)`** sets a certain beam propagation device as the source for the viewer.
- **`bpv_set_channel(j)`** sets a certain beam propagation channel.
- **`bpv_set_domain(s$)`** sets the spatial domain with the argument **`'real space'`** or **`'Fourier space'`**.
- **`bpv_set_interpolation(s$)`** sets the interpolation with the argument **`'none'`**, **`'linear'`** or **`'quadratic'`**.
- **`bpv_set_resolution(s$)`** sets the resolution with the argument **`'coarse'`**, **`'middle'`** or **`'fine'`**.

- **`bpv_set_color(s$)`** sets the color with the argument **`'default'`**, **`'red'`**, **`'green'`**, **`'blue'`** or **`'user'`**.
- **`bpv_set_plot(s$)`** sets the plot with the argument **`'intensity'`**, **`'amplitude'`** or **`'function'`**.
- **`bpv_set_function(s$)`** sets a plot function.
- **`bpv_set_log_scaling(r)`** switches the logarithmic scaling on (**`r`** ≠ 0) or off (**`r`** = 0).
- **`bpv_set_remove_fast_phase(r)`** switches the removal of the fast phase variation in *z* direction on (**`r`** ≠ 0) or off (**`r`** = 0).
- **`bpv_set_plane(s$)`** sets the display plane with the argument **`'xy'`**, **`'xz'`** or **`'yz'`**.
- **`bpv_set_show_parameters(r)`** switches the display of parameters on or off.
- **`bpv_set_show_grid_lines(r)`** switches the display of grid lines on or off.
- **`bpv_set_scaling(j)`** sets the scaling of color values to some number of decibels between –10 and +10.
- **`bpv_set_zoom(j)`** sets the zoom level. The *x* and *y* coordinate ranges are reduced by the factor 2^j.
- **`bpv_set_position(r)`** sets the *x*, *y* or *z* position (depending on the display plane) to a certain value (in meters, or in 1/m if the Fourier space is set for the *yz* or *xz* plane).

5.14 Dynamic Simulations

RP Fiber Power allows one to simulate the temporal evolution of active fiber systems. Some fundamental issues and two different kinds of numerical model have been discussed in section 2.8. In the following, it is explained how to implement such models with **RP Fiber Power**:

- First we need to determine the initial state of the fiber, i.e., the electronic excitations at the beginning of the simulation. For that, we usually first calculate the steady-state populations in the usual way. As an example, consider a fiber amplifier with continuous pumping and the amplification of intense signal pulses at a low repetition rate. In that case, simulate the steady-state situation for a certain pump power but zero signal power. The same would apply to a Q-switched laser: the initial state can be taken as the steady state for pumping without lasing.

  Of course, you may also do a dynamic simulation just after another dynamic simulation, starting with the final state of that simulation.
- Next, we must define the time-dependent input powers. For each optical channel with time-dependent input, make a function call like **`set_P_in_dyn(signal, 'P_s_in(t)')`**. This assigns an expression (here: a function) for the time-dependent input power to some optical channel. One may also directly enter a time-dependent term; example: **`set_P_in_dyn(ch, 'exp(-(t/5e-9)^2)')`**. (A call with an empty string could later be used to suppress the time dependence for future simulations.)
- If reflectivities at fiber ends (see section 2.4) are also time-dependent (as it is the case in Q-switched lasers), these can be defined with function calls like **`set_R_dyn(signal,`**

'R1(t)','R2(t)'). This assigns two previously defined functions for the time-dependent reflectivities at both fiber ends.

- Time-dependent localized losses inside the resonator (left and right of the fiber) can be defined with **set_loss_int_dyn(signal, 'l1(t) ', 'l2(t) ')**. This assigns two time-dependent functions for these losses (with values between 0 and 1). This function can be used for simulating active Q switching, for example.
- If it is necessary to access dynamic optical powers not only at the fiber output, but also at any position within the fiber, one has to make a function call like **calc_P_dyn_z(ch,flag)** for each channel where that is the case. The first function parameter is the channel number, and the second one a flag, a non-zero value indicates that powers have to be stored. That function call has to be done before calling **calc_dyn()**. (Note that the required amount of memory is proportional to the number of selected channels, the number of spatial steps in the fiber, and the number of temporal steps of the dynamic simulation.)
- If it is necessary to access dynamic excitations at any position within the fiber, rather than only values averaged over the fiber length, one has to make a function call like **calc_n_dyn_z(flag)**, where the parameter must be non-zero. That function call has to be done before calling **calc_dyn()**.
- Then start the dynamic calculation with a function call like **calc_dyn(t1, t2, dt)**, which defines a time interval [**t1**, **t2**] and a temporal step size **dt**. The software then simulates the time evolution of the system within that time interval and stores the output powers of all channels. The third parameter (**dt**) determines the kind of model used, as explained in section 2.8:
    - A non-zero value of **dt** indicates that a simplified model should be used, where the propagation times are neglected and the user can specify a temporal step size which may be much larger than the time required by light to propagate through some fiber section, or even a whole round trip. Note that the user is then responsible for making the time interval sufficiently short to properly sample the varying optical powers. Beyond that, the software automatically chooses finer temporal sub-steps if the level populations would otherwise change too rapidly to produce accurate results.
    - If 0 is used for **dt**, this indicates that propagation times should not be ignored, and the used temporal step size corresponds to the time required by light to propagate through one fiber section. That time is calculated as the length of a fiber section times the group index divided by the vacuum velocity of light. The group index is taken from the variable **n_g**, which must be defined beforehand by the user. (Typical values for glass fibers would be of the order of 1.5.) Also, you may use the function **set_delays(T1, T2)** (before calling **calc_dyn()**) to define time delays (in seconds) occurring on the left and right side of the active fiber, respectively, if the laser resonator contains additional undoped fibers or air spaces.
    - There is a modified version **calc_dyn_cond()**, having an additional string parameter which is a condition (example: **'P_out_dyn(signal, t) < 1 MW'**). Here, the propagation is only done while the condition is fulfilled. This feature can be used, e.g. to stop the propagation when some optical power or gain reaches a certain threshold.

- Finally, you can retrieve the calculated time-dependent values with various functions:
  - Time-dependent output powers are obtained with function calls like **`P_out_dyn(ch, t)`**.
  - Time-dependent powers anywhere within the fiber can be calculated with calls like **`P_dyn(ch, z, t)`**, provided that **`calc_P_dyn_z(ch,l)`** has been called before **`calc_dyn()`** for the corresponding channel.
  - The maximum output power can be obtained as **`P_out_dyn_max(ch)`**, and the peak position with **`t_peak(ch)`**.
  - You can obtain the time-dependent level populations (averaged over the fiber length) with **`n_av_dyn(l, t)`** and the time-dependent single-pass gain with **`sp_gain_dyn(Ch, t)`**.
  - Time-dependent excitations anywhere within the fiber can be calculated with calls like **`n_dyn(l, z, t)`**, provided that **`calc_n_dyn_z(l)`** has been called before **`calc_dyn()`**.
  - In addition, you may use functions like **`P(ch,z)`** and **`n(z,l)`** to calculate position-dependent quantities, which then apply to the end of the simulated time interval.

It is also possible to do multiple dynamic simulations, covering subsequent time intervals. For example, when modeling the emission of pulses from a Q-switched laser, it is sensible to use the precise method (**`dt`** = 0) for the time interval with the open Q-switch (where the pulse is emitted), later to use the simplified method for the pumping phase with closed Q-switch (where a substantially larger temporal step size can be used) and then model further pulse emission. After each call of the **`calc_dyn(t1, t2, dt)`** function, the dynamic data for powers, excitations and single-pass gain are available for the given time interval, but no more for time intervals corresponding to previous function calls.

The use of these functions is illustrated in two demo files, see sections 8.26 and 8.27.

5.15 Simulating Ultrashort Pulse Propagation

RP Fiber Power can simulate the propagation of ultrashort pulses. It is possible not only to simulate single-pass propagation through a fiber amplifier, but also to simulate effects of other optical components (for example, saturable absorbers or spectral filters). One can apply such effects on a pulse in any sequence, so that one can for example simulate the pulse evolution in a mode-locked fiber laser, the resonator of which contains multiple optical components.

The underlying principles are explained in section 2.9. The following steps have to be done to implement a simulation:

Define the Numerical Grid

The numerical grid for the pulses has to be defined with a function call like **`set_pulse_grid(T,N,lambda)`**, where **`T`** is the width of the temporal range, **`N`** is the number of numerical samples (which has to be a power of 2), and **`lambda`** is the center wavelength. Some hints for determining the numerical parameters:

- The temporal width **`T`** must be large enough to fully contain the temporal pulse profile (not only for the initial pulse, but for the whole propagation). The frequency resolution will be the inverse of the temporal width of the grid.

Note that for the simulation of stimulated Raman scattering, a high temporal resolution is required. For silica fibers, the temporal resolution should be better than 10 fs.

- In the frequency domain, the resolution will be **`1/T`** (i.e., the inverse of the temporal width of the grid), and the total covered spectral range will be **`N/T`**. That range must be large enough to fully contain the spectral pulse profile.

It is the user's responsibility to correctly choose these parameters. A test can be to check whether the results change significantly if the numerical resolution is doubled, either by doubling **`T`** and **`N`** (using a doubled temporal range but the same spectral range) or by doubling only **`N`** (using twice the spectral range).

The number of numerical samples can be very large – one may sometimes need **`N`** = 2^{16} or even 2^{18}, although in most cases values like 2^8 or 2^{10} are fully sufficient. A large number implies that the computations becomes slower and more memory is needed. That may limit the number of pulses which can be kept in the main memory.

Define the Initial Pulse

The initial pulse has to be defined. In the simplest case, one uses a Gaussian-shaped pulse, calling **`startpulse_G(E,tau,chirp)`** where **`E`** is the pulse energy, **`tau`** the pulse duration (full width at half maximum) and **`chirp`** the chirp parameter (rate with which the optical frequency changes, in units of Hz/s). The spectral maximum is assumed to be at the center wavelength of the chosen channel.

Similarly, a sech^2-shaped pulse can be obtained with **`startpulse_s(E,tau,chirp)`**. That pulse shape (with no chirp) is typical for fundamental soliton pulses.

For defining an arbitrary initial pulse based on a temporal pulse profile, one can instead call **`startpulse_t("A0%(t)")`** where the argument is a previously defined complex envelope function **`A0%(t)`**. Similarly, with **`startpulse_f("A0%(f)")`** one can determine a spectral profile, defined by a function **`A0%(f)`** where **`f`** is the (absolute) optical frequency. The integral of the squared modulus of such a function over all times or frequencies, respectively, must be the pulse energy.

Finally, it is also possible to use a pulse which was previously stored with the function **`store_pulse(j)`**, where **`j`** indicated a storage index (for example, 0). One can use that pulse with **`startpulse_recall(j)`**.

Define the Fiber Details and Fiber-specific Simulation Parameters

If pulses are to be propagated through a fiber, one first has to define some additional fiber parameters with certain function calls:

- **Wavelength-dependent background losses** (originating not from the laser-active dopant, but for example from impurities and scattering) can be defined with **`set_bgloss(loss[])`** where the argument is an array containing wavelength-dependent loss values in units of dB/m. That array must be defined and populated before using **`set_bgloss()`**. If that function is used, the wavelength-independent background loss of the fiber (see **`addinputchannel()`**) is ignored.

- **Group velocity dispersion (GVD)** is defined with **`set_GVD(GVD)`**. This defines a constant group velocity dispersion (in units of s^2/m). If the GVD is wavelength-dependent, one instead calls **`set_GVD_l("GVD_fiber(l)")`**, where the argument is an expression depending on the wavelength **`l`**. (In the example case, a GVD function

GVD_fiber(l) has been defined beforehand.) Alternatively, one can use **set_n_eff("n_eff(l)")** to define a wavelength-dependent effective refractive index function.

In case that the function **pp_fiber_2p()** is used for sending two pulses through a fiber, the second pulse will by default see the same group velocity dispersion as the first one. However, one can call the function **set_GVD2()**, **set_GVD_l2()** or **set_n_eff2()** (in analogy to the functions described above, and after calling these) to define a different dispersion for the second pulse.

- The **nonlinear index** can be defined with the function **set_n2(n2)**. (The argument must be the actual numerical value in units of m^2/W – for example, 2.6e–20 for silica.) Together with the effective mode area of the optical channel, this determines the strength of the nonlinearity.
- **set_ss(s)** defines a coefficient for **self-steepening**. If the given value is 1, the normal magnitude of the self-steepening effect is taken. This is appropriate if the nonlinear index and the fiber's effective mode area are not wavelength-dependent. One may use larger values for simulating the influence of the wavelength dependence of the nonlinear index and the effective mode area.
- **set_dnr("h_R(t)",f_R)** defines the parameters of a **delayed nonlinear response**, which leads to **Raman scattering**. (For pure self-phase modulation, this can be omitted.) The first argument is the (previously defined) Raman response function. **f_R** is a parameter between 0 and 1, which specifies the relative strength of the delayed nonlinear response (see section 2.9.3).
- If the function **pp_fiber_2p()** is used for sending two pulses through a fiber, one can set the strength of the nonlinear interaction (essentially, cross-phase modulation) with the function **set_XPM(x)**. Here, **x** is a dimensionless parameter, the default value of which is 2. That means that the phase change of pulse 2 per watt of power of pulse 1 is 2 times that resulting from self-phase modulation. This is the situation when two pulses propagate in the same optical channel with the same polarization direction. If the polarization directions are linear but mutually orthogonal, and the medium is isotropic (e.g., a glass), the correct value of **x** would be 2 / 3. That value could also be modified to take into account a reduced spatial overlap, for example.
- **set_TPA(beta)** defines a coefficient for two-photon absorption (TPA). The optical power in the time domain is attenuated according to $\frac{\partial P}{\partial z} = -\beta\, P(t) / A_{\text{eff}}$ where β is a coefficient with units of m/W and A_{eff} is the effective area of the intensity profile. Note that TPA is not relevant for most optical fibers.
- **set_MPA(beta,n)** defines a coefficient for multi-photon absorption (a generalization of TPA). The optical power in the time domain is attenuated according to $\frac{\partial P}{\partial z} = -\beta \left(P(t) / A_{\text{eff}} \right)^{n-1}$ where β is a coefficient determining the strength, n is the order (between 2 and 10), and A_{eff} is the effective area of the intensity profile. Note that for $n > 2$ and non-flat intensity profiles the effective area would strictly speaking have to be redefined (taking into account the relatively higher weight of the high-intensity parts), but that can also be taken into account via the β parameter.

- In case that the effect of gain saturation should *not* be simulated, one can call the function **`set_gain_sat(0)`**. With the argument 1, that gain saturation is normally treated; this is the default. Larger values can be used to simulate gain saturation in multiple passes.

For active (amplifying) fibers, it is important that the correct initial state of the fiber is prepared. For example, if a pulse is amplified after a long time of pumping the fiber without a signal input, one may calculate the steady state of the fiber with zero signal input.

If a pulse train with a high repetition rate is amplified and the steady state is of interest, one first calculates the output power of a continuous-wave signal, the power of which is equals the average power of the pulse train. (For broadband pulses, one may have to combine multiple signal channels to properly simulate the pulse spectrum.)

Strictly speaking, the explained method does not lead to an exact self-consistent solution, because the pulse spectrum may change during propagation, and this can also influence the actual amplifier gain for the pulses. This, however, should rarely be a significant problem. If it ever is, one has to simulate a sequence of amplification passes and pump phases; ask for technical support in that case.

The used algorithm automatically monitors the numerical accuracy and adjusts the numerical step size in *z* direction accordingly. The default settings should normally be appropriate. They lead to fairly high accuracy. Nevertheless, the computation time is not much longer than possible even for a substantially reduced accuracy.

The desired accuracy (which influences the automatic step size control) can be modified by calling the function **`set_pulse_tolerance(tol)`** where **`tol`** is a value between 10^{-6} (1e–6) and 10^{-1}. The default value is 10^{-2}, which means that the square root of the time- or frequency-integrated amplitude errors is of the order of 10^{-2} times the square root of the pulse energy. If a high accuracy is required, the computation time may be longer, and in some cases the desired accuracy might not be achieved (which triggers a numerical error, aborting the calculation). However, if the function **`set_pulse_tolerance()`** is called with a *negative* argument, the pulse tolerance is taken to be its modulus, and an additional rule applies: an error is not triggered when the desired accuracy cannot be reached even with the finest possible numerical steps.

Note that the numerical error can only be estimated by the software, as the precise solution is not known. The used method for estimating the magnitude of numerical errors is rather conservative. Therefore, the actually achieved accuracy is usually much higher than the desired one.

Note also that reflectivities at the fiber ends, as can be defined with the **`set_R()`** function (see section 5.6.3), are ignored for ultrashort pulse propagation. The same holds for localized losses (section 5.7).

Propagate through Optical Elements

A number of functions can be used to propagate the current pulse through various kinds of optical elements. The following list describes these functions only briefly; see the reference section 6.1.6 for more details.

- **`pp_fiber(fn,ch)`** propagates the pulse through the fiber **`fn`**, using the optical channel **`ch`**. The fiber number **`fn`** is usually 1, but can be different if different fibers have been defined using **`set_device()`**. The assigned optical channel (see section 5.6) determines the propagation direction (forward or backward), the intensity profile (which is relevant for nonlinear effects) and the background losses for the pulses. The resulting pulse will be the pulse as it exits the fiber at the other end.

With the function **get_pulse(z)**, one can obtain the pulse at a particular position within the fiber, but only after calling **pp_fiber()**.

- **pp_fiber_2p(fn,ch1,ch2,p2_in,p2_out)** propagates two different pulses through a fiber where they can interact via nonlinearities. The optical channels **ch1** and **ch2** of the two pulses can be the same or different. While the first pulse is taken from the current pulse, the second pulse is taken to be the stored pulse (see the function **store_pulse()**) with index **p2_in**, and the second pulse after the interaction will be stored with the index **p2_out**. The second pulse at a position within the fiber can be obtained with **get_pulse2()**.

  Note that two pulses could in principle also be combined in a single time or frequency trace and propagated via **pp_fiber()**. However, **pp_fiber_2p()** is better suited if the pulses have very different center wavelengths. In that situation, the frequency trace would have to be extremely wide when using **pp_fiber()**.

- **pp_dispersion(D2,D3,D4)** applies chromatic dispersion of second, third and fourth order. The units of these parameters are s^2, s^3 and s^4, respectively.

- **pp_compress(order)** calculates the dispersive compression of a pulse. The argument of the function call determines the details of the compression:

  - If it is zero, a perfect compressor is assumed, which set the spectral phase at all frequencies to zero.
  - If it is between 2 and 6, chromatic dispersion up to the corresponding order is applied such that the peak power of the compressed pulse is maximized.
  - If it is between −2 and −6, chromatic dispersion up to the corresponding order (modulus of the argument) is applied such that the duration of the compressed pulse is minimized.

- **pp_loss(fl)** simulates a wavelength-independent linear loss. The argument is the fractional loss of power (e.g, 0.10 for 10% loss). For wavelength-dependent losses, use the function **pp_multiply_f()** (see below).

- **pp_multiply_t(f_t%[])** multiplies the whole time trace (i.e., the temporal amplitudes) with the contents of an array with the given name.

- **pp_multiply_f(f_f%[])** works like **pp_multiply_t()**, but in the frequency domain. That function can be used, for example, for simulating spectral filters or for chromatic dispersion with arbitrary wavelength dependence.

- **pp_SPM(gamma)** applies a nonlinear optical element where self-phase modulation (SPM) occurs. The argument is a nonlinear coefficient with units of rad/W.

- **pp_sat_abs(loss,tau,E_sat)** applies a saturable absorber with some initial power loss (e.g., 0.10 for 10%), a recovery time **tau** and the saturation energy **E_sat**. If the recovery time is zero, one has a fast saturable absorber (adjusting the losses instantaneously to the optical power), and the last argument is the saturation power (not energy).

- **`pp_loss_p(lossfunc$,tau)`** provides a more general optical element with power-dependent losses. The first argument is the name of a function which determines the fractional loss (e.g, 0.10 for 10% loss) from the optical power **`P`**, if the second argument is zero. In case of a non-zero second argument **`tau`**, the function depends on **`E`**, the temporally integrated incident power, and **`tau`** is the time constant with which the excitation decays.

  For example, a fast saturable absorber with saturation power **`P_sat`** could be modeled with

  ```
  nl_loss(P):=loss_unsat/(1+P/P_sat)
  ```

  ```
  calc pp_loss_p("nl_loss",0)
  ```

  while a slow absorber could be modeled with

  ```
  nl_loss(E):=loss_unsat*exp(-E/E_sat_a)
  ```

  ```
  calc pp_loss_p("nl_loss",tau_a)
  ```

  The function allows also one to model induced absorption, for example. Note that the accuracy of this approach is low for a slow absorber if the saturation within a single time step is substantial.

- **`pp_TPA(beta)`** applies an optical element with two-photon absorption (TPA). For small absorption, the optical power in the time domain is attenuated according to $P_\mathrm{out}(t) = \exp(-\beta P(t)) \cdot P$ where $P(t)$ is the incident power and β is a coefficient with units of W^{-1}. For larger absorption, the output power will be higher than according to the formula, as TPA becomes weaker during the passage through the device.

- **`pp_MPA(beta,n)`** applies an optical element with multi-photon absorption (MPA). For small absorption, the optical power in the time domain is attenuated according to $P_\mathrm{out}(t) = \exp(-\beta P(t)^{n-1}) \cdot P$ where $P(t)$ is the incident power, n is the order (2 to 10), and β is a coefficient with units of $\mathrm{W}^{-(n-1)}$. For larger absorption, the output power will be higher than according to the formula, as MPA becomes weaker during the passage through the device.

- **`pp_noise(p)`** applies white (frequency-independent) random noise to a pulse. The argument directly determines the noise power, if it is positive. If it is negative, its modulus defines a noise power relative to the quantum noise limit.

- **`pp_center(w)`** centers the pulse in the time domain, i.e., it shifts the time trace such that the maximum power occurs at *t* = 0. If the argument is non-zero, amplitudes at the edges of the time trace can “wrap around”. With the function **`dt_c()`** one can obtain the total temporal shift applied by all calls of the **`pp_center()`** function.

- **`pp_shift_t(dt,wrap)`** shifts the temporal pulse envelope (without affecting the phase) by **`dt`**. If **`wrap`** is not zero, the amplitudes can wrap around at the edges of the time trace.

- **`pp_shift_phi(dphi)`** changes the optical phase of the whole pulse by **`dphi`**.

- **`pp_add_pulse(j,f%)`** adds the amplitudes of the stored pulse with index **`j`**, multiplied with **`f%`**, to the current pulse. This may be used for modeling interferometers, for example.

After each such function call, one can obtain the pulse properties with the functions explained in the next section. One can apply any sequence of optical elements.

Obtain the Pulse Properties

One can now obtain the pulse properties with a variety of functions. Some examples:

- **`P_t(t)`** is the time-dependent optical power.
- **`P_p()`** is the peak power.
- **`tau_p()`** is the pulse duration (FWHM = full width at half maximum).
- **`P_f(f)`** returns the power spectral density with respect to the absolute optical frequency in units of J/Hz. Similarly, **`P_l(l)`** returns the power spectral density with respect to the wavelength in units of J/m. (When integrating such a power spectral density over all frequencies or wavelengths, one obtains the pulse energy.)
- **`df_p()`** is the pulse bandwidth (FWHM) in Hz, **`dl_p()`** the pulse bandwidth (FWHM) in the wavelength domain (in meters).
- **`E_p()`** is the pulse energy.

The complete list of available functions is given in section 6.1.6.

Saving and Recalling Pulses

- A pulse can be saved in a file with **`save_pulse_t(n$)`**, where the function parameter is the filename. This will save the time-domain information. With **`save_pulse_f(n$)`**, one can save the frequency-domain information. In any case, the other domain will be recalculated automatically after loading if needed.
- A pulse can be loaded from a file with **`load_pulse(n$)`**, where the parameter is the filename.

  Note that the save and load functions return the value 1 if successful, otherwise 0.
- It is also possible to store up to 10 000 pulses in the main memory. These pulses can be distinguished with a storage index between 0 and 9 999. One can store the current pulse with **`store_pulse(j)`** and recall it later with **`recall_pulse(j)`**.

Example

Consider a simple case where a Gaussian initial pulse is amplified in a fiber with constant second-order dispersion (GVD) and the Kerr nonlinearity.

It is often convenient to pack the required function calls into a user-defined function, because this makes it easy to repeat everything if, for example, the calculations later need to be redone for each point in a function graph.

Assuming that a fiber model with a signal channel called **`signal1_fw`** has been set up already, and the initial state of the fiber has already been calculated (for example, for zero input signal), the used code may be as follows:

```
; Parameters of initial pulse:
lambda0:=1060 nm  { center wavelength }
E0:=0.1 nJ  { pulse energy }
tau0:=100 fs  { pulse duration }
chirp0:=0  { chirp }
```

```
; Fiber details for ultrashort pulse amplification:
GVD_f:=20000 fs^2 { per meter }
n2_f:=3e-20  { nonlinear index }

; Numerical parameters:
N_t:=2^10
T:=30*tau0

DoPulsePropagation():=
  begin  { define the required inputs }
  set_pulse_grid(T, N_t, lambda0);
  startpulse_G(E0, tau0, 0);
  set_GVD(GVD_f);
  set_n2(n2_f);  { nonlinear index }
  pp_fiber(1,signal1_fw);
  end
calc DoPulsePropagation()
  { execute the function now }

show "Pulse energy: ", E_p():d3:"J"
show "final gain:   ", sp_gain(signal):d3:np:"dB"
```

If one later requires a function graph (see section 5.16) showing the output pulse energy as a function of the input pulse energy, the code for that graph may look as follows:

```
f: (E0:=x*nJ; DoPulsePropagation(); E_p())
```

Note that this will leave the pulses modified after the graph. In order to restore their initial state, one may use the following:

```
f: (E0:=x*nJ; DoPulsePropagation(); E_p()),
  init E00:=E0,
  finish (E0:=E00; DoPulsePropagation())
```

5.16 Graphical Output of Results

RP Fiber Power has inherited from **JPLOT** a large number of features to produce graphical output. Not all features of **JPLOT** are explained in this document, but the online help system gives more details. In the following, we consider an example for a plot which shows the distribution of optical powers and the upper-state population along the fiber, assuming that the parameters of an Yb-doped amplifier have been defined:

```
diagram 1:
"Powers along the Fiber"

x: 0, L_f
"position (m)", @x
y: 0, 2
"power (W)", @y
y2: 0, 100
frame

f: P(pump,x),
  color=lightred, width=3, ″pump″
f: P(signal_fw,x),
  color=lightblue, width=3, ″signal, forward″
f: P(signal_bw,x),
  color=magenta, width=3, ″signal, backward″
f: 100*n(x), yscale=2, style=dotted, ″excitation″
```

Some comments:

- A script file can define multiple diagrams by heading each set of graphics definitions with a command like "**`diagram 1:`**". By default, all defined diagrams will be shown; if this is not desired, one may use a command like **`diagram shown: 1, 3`** to define which diagrams should be shown.
- The first text string without further specification is taken as the heading of the graph. Further text strings would be taken as subheadings (with a smaller font).
- The commands **`x:`** and **`y:`** define the ranges of the coordinate axes, and **`frame`** changes the general kind of the coordinate system from the default two-axis setup to one with a kind of frame around the diagram. The strings defined after the coordinate ranges have the options **`@x`** and **`@y`**, indicating that these should be placed along the coordinate axes.
- The **`y2:`** command defines a second y axis. That is displayed only if the **`frame`** command is also used. The option **`yscale=2`** in the last graph selects the second y axis for vertical scaling.

  Similarly, a second x axis could be obtained with **`x2:`**, and with the option **`xscale=2`** a graph could be horizontally scaled according to that axis.
- The command **`f:`** defines a function graph. It basically requires only one argument: a mathematical expression depending on the horizontal coordinate **`x`**, with the result defining the vertical position. With commas, further options can be appended. The demonstrated options define the graph color (where some predefined values are black, blue, green, cyan, red, magenta, brown, lightgray, darkgray, lightblue, lightgreen, lightcyan, lightred, lightmagenta, yellow, orange, violet, white), the line width (scaled to the diagram size; you may require a value of at least 3 to see an effect), the line style (solid, dashed, fdashed, dotted, dotdashed), and a text string shown as a label in the diagram. One could also use the option **`step=10`** to have the expression evaluated only every 10 pixels (to save time).

  The command **`f:`** also supports the option **`init expr`**, where **`expr`** stands for a mathematical expression which is evaluated just before the graph is drawn. Similarly, **`finish expr`** would evaluate the expression after drawing the graph. One has to write e.g.

  ```
  f: P(signal_fw,x), init set_P_in(signal_fw,0.1e-3)
  f: P(signal_fw,x), init set_P_in(signal_fw,0.2e-3)
  ```

  rather than

  ```
  ! set_P_in(signal_fw,0.1e-3)
  f: P(signal_fw,x)
  ! set_P_in(signal_fw,0.2e-3)
  f: P(signal_fw,x)
  ```

  because the latter example would set the input powers at the wrong times: when the script is read, not when the diagrams are made.

- Note that the graphics are not immediately prepared when the script is read, but only once the whole script has been successfully read (i.e., without encountering e.g. a syntax error).
- When some settings have to be changed *during* the preparation of a graph (e.g., for varying the pump power along the horizontal axis), one uses a composite expression (see also section 5.19) for the vertical coordinate. For example,

  ```
  f: (set_P_in(pump,x); P_out(signal_fw))
  ```

  has an expression which first sets the pump power according to the horizontal position and then calculates and returns the signal output power. Such composite expression can have an arbitrary number of sub-expressions, separated by semicolons and embraced either in parenthesis or by **begin** and **end**. One could thus also write

  ```
  f: begin
     set_P_in(pump,x);
     P_out(signal_fw);
     end
  ```

- Instead of drawing a whole graph, one sometimes prefers only to show some points in the diagram. For example:

  ```
  ! for x:=3 to 4 step 0.1 do point(x+i*P(signal,x),"o")
  ```

  would display open circles at a few positions, specified as complex numbers with real part (horizontal coordinate) **x** and some power as the imaginary part. Other possible markers would be closed circles (“O”), rectangles (“r”), triangles (“t”), inverted triangles (“n”), rhombs (“h”), crosses (“x”), stars (“*”), single-pixel dots (“p”), or thicker dots (“P”).
- The **cp** command allows one to generate color plots, where each point in the coordinate system obtains a color as calculated from the *x* and *y* coordinates. The user must specify a mathematical expression which calculates an rgb color value from *x* and *y*. For example,

  ```
  colorscale(r):=
    if r >= 0
    then red
    else blue
  cp: colorscale(sin(x)*cos(y))
  ```

  first defines a function converting real values into colors, and the **cp** command uses this to indicate where the function argument is positive or negative. More sophisticated functions, using the **rgb()** function, can be used for defining continuous color scales. There are also the functions **color_I(r)** and **color_A(z%)** for intensity and complex amplitude color scales. The online help gives more details.

5.17 Display of Numerical Results

The command **show** allows to display some calculated values in the output area. For example:

```
show "P_s_out:  ", P_s_out:d3:"W"
```

which display a text string, followed by the value of a (previously calculated) variable, formatted with three valid digits and the units “W”.

Note that there is the function **showoutput()**, which can also display a numerical or string value in the output area, and can be called within any mathematical expression.

5.18 Output of Results to a File

The **JPLOT** command **`write`** allows to generate numerical output to files with great flexibility. This allows e.g. to export calculated values to other software. The principle is to specify a list of expressions (in brackets) which evaluates to a text string; this text string is sent to a file, and it can be repeated using the **`for`** option. As an example, consider the command

```
write [z:d5, ", ", P(signal,z):d5],
  for z:=0 to L_f step L_f/100,
  >”P_signal.dat”
```

which generates an output such as

```
0, 0.5613
0.1, 0.5782
0.2, 0.5968
...
10, 3.265
```

describing the variation of signal power along the fiber.

5.19 Overview on the Arithmetics

This software has very extensive arithmetics features, which it has inherited from **JPLOT**. This manual (other than the online help system) does not give a complete documentation of these features, but an overview on everything which should normally be required.

5.19.1 General Properties

Variable names without special indications can have real values. Complex variables have a name ending on “`%`” (e.g. **`E%`**), while a “**`$`**” sign is used for string variables.

Real numbers can be written with the usual exponential notation, e.g. **`3e-5`** = $3{\cdot}10^{-5}$. For complex numbers, write e.g. **`3+4*i`** or **`3%4`**.

The available arithmetic operators are "**`+`**", "**`-`**", "**`*`**", "**`/`**", and “**`^`**”, where the latter means “to the power of”. These operators have the usual hierarchy. Besides, there are the usual relational operators **<**, **<=**, **>=**, **>**, **=**, and **<>**.

Predefined constants are **`i`** (the imaginary unit) and **`pi`**.

5.19.2 Arrays

Arrays can hold real, complex or string values, can have multiple dimensions, and have non-integer index values. For example,

```
defarray z%[1,2,0.5; 0,10]
```

defines a two-dimensional complex array, with the first index running from 1 to 2 in steps of 0.5, and the second one from 0 to 10 in steps of 1. When a value is recalled e.g. with **`z%[1.3,5.8]`**, the array indices are rounded to the nearest index values. With a “**~**” before the bracket, the values are linearly interpolated, while a double “**~**” means quadratic interpolation, and a triple “**~**” means cubic spline interpolation.

5.19.3 Composite Expressions

For sophisticated calculations, one often uses composite expressions, consisting of several terms which are evaluated one after the other, with the final result being determined by the last one. The syntax is either

```
  (expr1; expr2; ...; exprN)
```

or

```
  begin expr1; expr2; ...; exprN; end
```

where **expr1** etc. are normal mathematical expressions. As an example, take

```
  (a:=4; b:=B/a; c:=-3*sqrt(C); (-b+sqrt(b^2-4*a*c))/(2*a))
```

which first assigns values to the variables **a**, **b** and **c**, and then calculates the final result from these.

Within such a composite expression, one may use local variables:

```
f(x):=
  begin
  var a, b, c;
  a:=4*x;
  b:=B/a;
  c:=-3*sqrt(C); (-b+sqrt(b^2-4*a*c))/(2*a));
  end
```

where **a**, **b** and **c** are local in the sense that the don’t interfere with any global variables having these names.

Since version V5, one can (and should) also declare the access to global variables. (Such declarations are required only in begin/end blocks of function definitions, and only if the general settings of the software demand it.) Example:

```
g(x):=
  begin
  global last_x;
  last_x:=x;
  sin(x);
  end
```

Another possibility is to declare local constants:

```
E(r):=
  begin
  const e = 1.609e-19;
  e/r^2;
  end
```

5.19.4 Functions for Arithmetics

There is a wide range of standard functions for doing arithmetics, which are listed in the following table. Some specialized functions are explained in the following sections.

Function	Description
sgn(r)	signum, i.e., 1 for r > 0, –1 for r < 0, 0 for r = 0
round(r)	rounding to nearest integer, e.g. **3.8** → 4, **-3.8** → –4
trunc(r)	rounding to the next smaller integer, e.g. **3.8** → 3, **-3.8** → –4

`abs(z)`	modulus (absolute value)
`abs2(z)`	squared modulus
`sqr(z)`	square (z^2)
`sqrt(z)`	square root
`cube(z)`	third power (z^3)
`rt3(z)`	cubic root ($z^{1/3}$)
`rnd(r)`	r > 0: equally distributed random numbers with $0 \le x < r$ r < 0: Gaussian distribution with standard deviation |r| r = 0: random numbers according to parameters set with `init_rnd()` (see below)
`init_rnd (sigma,tau,f0)`	initialize the generation of random numbers with `rnd(0)`: sigma = standard deviation, tau = coherence time (rel. to temporal spacing of subsequent samples), f0 = frequency of maximum spectral power density
`fac(n)`	factorial (equivalent to n!)
`re(z)`	real part of complex number
`im(z)`	imaginary part of complex number
`reim(r1,r2)`	assemble real and imaginary part to a complex number `r1 + i*r2`
`r2c(r)`	conversion from real to complex
`rphi(r1,r2)`	assemble modulus (absolute value) and phase to complex number `r1 * exp(i*r2)`
`conj(z)`	complex conjugate of a complex number
`exp(z)`	exponential function, base `e` ≈ 2.7182818
`expi(r)`	`exp(i*r)`
`expd(r)`	10^r
`expb(r)`	2^r
`ln(z)`	natural logarithm
`lg(z)`	logarithm with base 10
`lb(z)`	logarithm with base 2
`sin(z)`	trigonometric function; similar: `cos`, `tan`, `cot`, `arcsin`, `arccos`, `arctan`, `arccot`
`sinh(r)`	hyperbolic function; similar: `cosh`, `tanh`, `coth`, `arcsinh`, `arcosh`, `artanh`, `arcoth`
`not(r)`	logical negation (1 for `r` = 0, otherwise 0)
`minr(r1,r2)`	minimum (with up to 8 arguments)

`maxr(r1,r2)`	maximum (with up to 8 arguments)
`sum(j:=1 to n,f(j))`	sum
`prod(j:=1 to n,f(j))`	product
`FFT(...)`	fast Fourier transform (section 5.19.7)
`int(...)`	numerical integration (see section 5.19.5)
`zero()`	numerical root finding (see section 5.19.5)
`inc(v), dec(v)`	increment or decrement a variable or array value by 1
`inc(v,r), dec(v,r)`	increment or decrement a variable or array value by `r`
`defined(term)`	1, if the term is defined, otherwise 0; examples: `defined(sqrt(2))=1`, `defined(sqrt(-2))=0`

5.19.5 Numerical Integration

Numerical integration is done with the function `int()`. By default, this function uses the Simpson method, where the integrand is evaluated at equidistant points, including the ends of the integration interval. For example,

```
  int(f(x), x:=a to b, tol=1e-6)
```

will numerically evaluate the integral of `f(x)`, where the integration variable `x` runs through the range `a`...`b` and `tol` is the required (absolute) tolerance. The numerical approximation is refined until the changes to the result are smaller than `tol`. The numerical error is then usually significantly smaller than `tol`, but as it is always with numerical approaches, there is no guarantee. Instead of the absolute tolerance `tol`, a relative tolerance `rtol` can be used.

5.19.6 Numerical Root Finding

A special case is the function `zero` for root finding.

The syntax is:

```
  zero(f(x),x in [a,b],ytol=c,xtol=d)
```

where `a`, `b`, `c`, `d` can be numbers or expressions. This will numerically search for a zero of the real function `f(x)` in the interval [`a`,`b`]. (If `f(a)*f(b) > 0`, the routine will try to expand the interval so that it incorporates a zero, but normally the interval [`a,b`] should be chosen so that it contains the zero and `f(a)  f(b)  ≤  0`.) The function `f` is assumed to be continuous (but not necessarily differentiable). An iteration is used to find the root. It ends when a value `x` is found for which we have either `|f(x)| ≤ |ytol|` or `|x-x_0| ≤ |xtol|` (`x_0` being the exact zero). It is sufficient to specify either `xtol` or `ytol`.

5.19.7 Fast Fourier Transforms (FFT)

The function `FFT()` does a one-dimensional fast Fourier transform. It has three arguments:

- The first argument is the input array (with empty brackets), which must be real or complex and must have a number of components which is a power of 2.
- The second argument is the output array (with empty brackets), which must be complex.
- The third argument must be +1 or –1, indicating the direction of the transform: +1 means that the argument of the integral contains the exponential with the argument $+i\omega t$.

The function then calculates the Fourier transform of the input array and stores the results in the output array. It also sets the size and the correct index range of the output array (so that the initial array size and index range does not matter). For example, if the input array has the index range 1 to 1024, then the output array has 1024 components ranging from to –511/1024 to +512/1024 with steps of 1/1024. (Transforming this back, the index range would be –511 to +512.)

Example:

```
dummy:=FFT(x[],X%[],+1)
```

5.19.8 Control Structures

For distinguishing between different cases within an expression, one can use the if-then-else structure. Example:

```
a:=if b>=0 then sqrt(b) else 0
```

For iterations, there are three basically self-explanatory types of loops. Examples:

```
while n>1 do n:=n/2;
repeat
  n:=n/2;
until n<1;
for z:=1 to 2 step 0.1 do n:=n+1/z;
```

The basic difference between the while-do loop and the repeat-until loop is that in the latter case the loop body is executed at least once, because the condition is checked only at the end. In the for-do loop, `to` can be replaced with `downto` for the case that the end value is lower than the start value.

5.19.9 Functions for the Handling of Strings

Various functions can be used for manipulation strings, which consist of a variable number of characters:

Function	Description
`length(s$)`	length of a string
`pos(s1$, s2$)`	position of string `s1` in `s2`. Example: `pos('es','Test') = 2`.
`copy(s$, p, l)`	copies a portion of length `l` at position `p` in a string
`between(s1$, s2$, s$)`	The part of `s$` which is found between `s1$` and `s2$`. Example: `between('[',']','Test[abc]-') = 'abc'`.
`char(n)`	character with ANSI code n
`ord(s$)`	ANSI code value of a character
`replacestr(s$, old$, new$)`	replace `old$` with `new$` in `s$`
`uppercase(s$)`	converts lower-case to upper-case letters
`lowercase(s$)`	converts upper-case to lower-case letters

`str(t)`	string form of a real or complex term **`t`**. A format specifier (see the following section) can be appended to the argument.
`rval(s$)`	real value of a string argument, e.g. **`rval('3.5') = 3.5`**
`cval(s$)`	complex value of a string argument, e.g. **`rval('3%4') = 3+4i`**
`matches_re(s$, reg$, options$)`	Checks whether the string **`s$`** matches the regular expression **`reg$`**. The third parameter is optional and can contain some options, e.g. **`'i'`** = ignore case (treat lower case like upper case letters) or **`'n=10'`** = find only up to 10 matches. An optional fourth parameter (e.g. **`m$[]`**) is a string array in which the matches are stored, if the regular expression contains one or several portions in parentheses.
`replace_re(s$, reg$, new$, options$)`	Replace the findings in **`s$`** according to the regular expression **`reg$`** with **`new$`**, and return that as the result. The fourth parameter is optional and can contain some options, e.g. **`'i'`** = ignore case (treat lower case like upper case letters) or **`'n=3'`** = replace only up to 3 matches.
`split_re(s$, reg$, p$[])`	Split the string **`s$`** into single strings, to be stored in the array **`p$[]`**, using the separator given as the regular expression **`reg$`**. The result is the number of parts obtained.

5.19.10Formatting of Numerical and String Values

For output to files or the Output area (see section 4.8), it is often desirable to format numerical values, e.g., to limit the number of digits and to use some units. For example, you may want to obtain “1.34 THz” instead of the unformatted value of “1.3452989014e12”. Formatting is achieved by appending a colon ("**`:`**") and some additional characters at the end of an expression. The rules are:

- Several format instructions can be appended, each beginning with a colon ("**`:`**").
- A non-zero natural number defines the minimum total length of the resulting string; this minimum length will be achieved by insertion of blanks, if necessary. A letter **`l`**, **`c`** or **`r`** directly after the number specifies the justification: left, center or right.
- An instruction like **`:d3`** specifies the total number of digits.
- With **`:e3`** (example) the last digit refers to 10^3, with **`:e-3`** to 10^{-3}.
- The format **`:f3`** results in 3 digits after the decimal point.
- The format **`:z3`** rounds to an integer number and adds leading zeros if the length would be less than 3 digits.
- Zeros at the end, after the decimal point, are suppressed by default To change this, append **`:n`**, e.g., in **`:d3:n`**.

- Units (max. 20 characters) are specified between double quotes, e.g., **"Hz"**. By default a blank is written between number and units; this can be avoided by writing a backslash ("\") in front of the units (e.g., **:"\Hz"**). If appropriate, a prefix like **k** for "kilo" (103) or **m** (10-3) is used; if you prefer an exponential to a prefix, append **:np** ("no prefix"). If the unit string begins with a prefix in parenthesis, the program will use this prefix (provided that the number string would not be too long). Similarly, a number in parenthesis specifies a preferred exponent. No prefix will be used if the unit string begins with "**/**" or "**%**".
- The format **:c** gives non-numerical output: if the given number is 1, the first character of the units string will be returned. For 2 it is the second character, etc. For a number outside of the valid range, "**?**" is returned.
- The format **:w** is similar to **:c** but works with words which are separated by commas: if the given number is 2, the second word of the units string will be returned. For a number outside of the valid range, "**?**" is returned. Note that the units string can not contain more than 20 characters.
- The format **:m** gives multiple copies of the unit string. For example, **3:m:"*-"** results in "***-*-*-**".

For **string expressions** there are the following rules:

- Several format instructions can be appended, each beginning with a colon ("**:**").
- A non-zero natural number defines the minimum total length of the resulting string; this minimum length will be achieved by insertion of blanks, if necessary. A letter **l**, **c** or **r** directly after the number specifies the justification: left, center or right.
- The format **:*3** produces 3 copies of the string. (A minimum length (if specified) applies to the whole string in this case.)

5.19.11Functions for Graphics

Function	Description
point(z%,m$)	plots a marker at the location **z%** (complex number containing x and y coordinates) with the type **m$** which can indicate a rectangle (“r”), closed circle (“O”), rectangle (“r”), triangle (“t”), inverted triangle (“n”), rhomb (“h”), cross (“x”), star (“*”), single-pixel dot (“p”), or thicker dot (“P”)
moveto(z%)	sets the current point
lineto(z%)	draws a line from the current point to the given point
rmoveto(z%), rlineto(z%)	work with coordinates relative to the current point
line(z1%,z2%)	draws a line from the point **z1%** to **z2%**
rect(z1%,z2%)	draws a rectangle
box(z1%,z2%)	draws a rectangle which is filled with the current color
circle(z%,r)	draws a circle around the point **z%** with the radius **r**

`getpos%()`	returns the current drawing position
`rgb(r,g,b)`	returns an RGB color value (with arguments between 0 and 1)
`egacolor(j)`	generates an EGA color value (with an argument between 0 = black and 15 = white)
`color_I(r)`	returns a color value for arguments between 0 and 1, based on a standard color scale ranging from blue over green to dark red
`color_A(z%)`	returns a color value for complex arguments with magnitudes up to 1. The complex phase determines the color (e.g., red for positive real values, green for negative real values), and the magnitude determines the color saturation.
`setcolor(c)`	sets a color for the following drawing operations; example: **`setcolor(blue)`** or **`setcolor(rgb(0.6,0.4,0.3))`**
`setrgb(r,g,b)`	does the same as **`setcolor(rgb(r,g,b))`**
`setwidth(w)`	sets the linewidth (in units of 1/1000 of the full width of the graph)
`set_scaling(sx,sy)`	allows one to change the scaling for following drawing operations. By default, all x coordinates are scaled according to the main x axis, and all y coordinates according to the first y axis. However, if one has defined a second y axis with the command **y2:**, one can use **set_scaling(1,2)** to scale all y values according to that axis. Similarly, with **set_scaling(2,1)** one would use the second x axis but the main y axis.

5.19.12Functions for Handling Files

Function	Description
`fileexists(n$)`	returns 1 if the file with the given name exists, otherwise 0
`copyfile(n1$,n2$)`	copies a file to another file and returns 1 if successful, otherwise 0
`deletefile(n$)`	deletes a file and returns 1 if successful, otherwise 0
`open_file(n$,m$)`	open a file with the name **`n$`**; the file mode is given by the string m$, e.g. **`'wt'`** = writing a text file, **`'rb'`** = reading a binary file. A file handle is returned.
`close_file(f)`	close a file with the given file handle
`write(f, x, ',', y)`	write data to a text file
`write_bin(f, x, y)`	write data to a binary file
`read_line(f, s$)`	read a line from a text file

read_bin(f, x)	read data from a binary file
eof(f)	checks whether the end of a file is reached
read_file_content(n$, s$)	read the whole content of a file to a string **s$**, or alternatively to a text array **s$[]**
write_file_content(n$, s$)	write the content of a string **s$** (or alternatively of an array **s$[]**) to a file
read_directory(dir$, files$[], options$)	Read the content of a Windows directory to a string array. The result is 1 if it is successful (directory exists), 0 otherwise. The optional string **options$** can contain **'r'** (recurse sub-directories).

5.19.13Various Other Functions

Function	Description
showmessage(...)	displays between 1 and 4 messages of real, complex or string type (one in each line); execution continues when the user closes the form
showoutput(...)	displays the argument in the output area
now()	current date and time, in seconds since 01/01/2000, 0:00
play(f,T)	generates a sound with frequency **f** (in Hz, i.e., oscillations per second) and duration **T** (in seconds)
exec(cmd$, params$, dir$)	executes a program, a Windows command or opens a document
download(URL$)	downloads a file to a string variable

6 Reference for Model-specific Functions

6.1.1 Functions for Defining or Modifying the Fiber Model

The following functions can be used to define the model:

- **set_fiber(L_f, N_z, gainsystem$)**: defines the parameters of the fiber, see section 5.4
- **set_phi_steps(n)**: defines the number of azimuthal steps (up to 128), if the intensity profile functions are not all radially symmetric.
- **make_ring()**: indicates that a ring laser configuration is assumed.
- **add_ring(r, N_dop)**: defines a doped ring with the outer radius **r**. If the dopant concentration is not radially symmetric, one may use the function **set_N_dop()** (see below). Up to 30 rings can be defined.
- **def_ion(IonType$, l1, l2)**: defines the name and the energy levels of some type of laser-active ions
- **set_N_dop(k, r, phi, N)**: defines (or modifies) the doping concentration (number density) **N** of ion type number **k** at the radius **r** and azimuth angle **phi**. If **phi** is negative,

the doping concentration is set for the whole ring. If a rectangular grid is used, the coordinates r and phi are replaced with **x** and **y**.

- **def_spont(l1, l2, R)**: defines a spontaneous (radiative or non-radiative, but not light-induced) transition from level **l1** to level **l2** with a given rate **R**
- **def_stim(l1, l2, 'sigma')**: defines a stimulated (light-induced) transition from level **l1** to level **l2** with the transition cross section function having the name **sigma()**
- **def_quench(l1, l2, p, Q)**: defines a quenching process, where a transition from level **l1** to **l2** occurs, and the rate is **Q** times the square (for **p** = 2) or the cube (for **p** = 3) of the fractional population in the starting level. The units of **Q** are m^3/s (for **p** = 2) or m^6/s (for **p** = 3). (See the online help system for more details.)
- **def_upcon(l1, l2, l3, p, K)**: defines an upconversion process, starting with two ions in level **l1**, where one goes to level **l2** and the other one to level **l3**. The rate of depleting the starting level is **K** times square (for **p** = 2) or the cube (for **p** = 3) of the fractional population in the starting level. The units of **K** are m3/s (for **p** = 2) or m6/s (for **p** = 3). The rates of ions getting into the ending levels **l2** and **l3** are half the rate of depleting the starting level. (See the online help system for more details.)
- **def_transfer(l1, l2, l3, l4, K)**: defines an energy transfer process, where one ion is transferred from level **l1** to **l2**, and another ion is transferred from level **l3** to **l4**. The units of **K** are m3/s. (See the online help system for more details.)
- **addinputchannel(P_in, lambda, 'I', loss, direction)**: defines an input channel, see section 5.6.1. The given intensity function (third argument) may also depend on **r** and **phi**. If a refractive index profile has been defined with **set_n_profile()**, one can also refer to a calculated mode function. For example, the third argument could be **'I_lm(0,1)'** for the LP_{01} mode, or **'I_lm(1,1,cos)'** for the LP_{11} mode with the cos φ dependence.
- **addASEchannel(lambda, dlambda, M, 'I(r)', loss, direction)**: defines an ASE channel, see section 5.6.2. The intensity function may also depend on **r** and **phi**, or may refer to a calculate mode function in the same way as **addinputchannel()**.
- **set_R(ch, R1, R2)**: sets the reflectivities for coupling channels, see section 5.6.3
- **set_loss_int(channel,li1,li2)** defines localized power losses, which apply on the way between the left or right fiber end, respectively, and the corresponding end mirror.
- **set_loss_ext(channel,le1,le2)** defines localized power losses which apply outside the end mirrors.
- **clear_channels()** removes all channel definitions, so that the channels can be redefined from scratch.
- **finish_fiber()** is called to indicate the end of the model definitions.
- **checkplausibility()**: switches plausibility checking on or off. By default, plausibility checking is on, and you may want to switch it off when working with extreme parameter values, where the plausibility checking generates error messages.

- **`set_lambda(ch, l)`**: modifies the wavelength of channel **`ch`** (possible also after the whole fiber definition has been finished)
- **`set_dlambda(ch, dl)`**: modifies the wavelength bandwidth of an ASE channel **`ch`**
- **`set_loss(ch, l)`**: modifies the parasitic losses of channel **`ch`**

6.1.2 Functions for Obtaining Calculated Results

The following functions can be used to obtain results calculated with the model (excluding functions for use in dynamic simulations, for the mode solver and ultrashort pulses):

- **`P(ch, z)`**: power propagating in an optical channel at a certain position, see section 5.8
- **`P_out(ch)`**: output power from a channel, taking into account possible reflection at the end, see section 5.8
- **`I(ch, z, r, phi)`**: optical intensity of channel **`ch`** at the longitudinal position **`z`**, radial position **`r`** and azimuth **phi**. For a rectangular grid, **`r`** and **`phi`** are replaced with **`x`** and **`y`**. If a negative **`z`** value is given, the intensity is calculated for 1 W of optical power.
- **`sp_gain(ch)`**: internal single-pass power gain (in decibels) of a channel
- **`gs_abs(ch)`**: ground state absorption (in decibels per meter) of a channel
- **`E_sat(ch)`**: saturation energy of a channel

  Note that the saturation energy is calculated based on the transverse ring segment with highest optical intensity (taking the average intensity of each ring). For pulse energies of that magnitude, significant changes of ionic excitations must be expected.
- **`NF(ch)`**: noise figure of a channel
- **`n(z, l)`**: fractional excitation of the level **`l`** of laser-active ions at a certain position **`z`**, averaged in the transverse direction; see section 5.8
- **`n_tr(z, r, phi, l)`**: fractional excitation of the level **`l`** of laser-active ions at a certain longitudinal position **`z`**, radial position **`r`** and azimuth **`phi`**, see section 5.8. For a rectangular grid, **`r`** and **`phi`** are replaced with **`x`** and **`y`**.
- **`n_av(l)`**: average fractional excitation of the level **`l`** of laser-active ions (averaging over the whole fiber in the longitudinal and transverse direction)
- **`N_dop(k, r, phi)`**: doping concentration of ion type **`k`** at the radial position **`r`** and azimuth **`phi`**. For a rectangular grid, **`r`** and **`phi`** are replaced with **`x`** and **`y`**.
- **`E_sat(ch)`** is the saturation energy for some optical channel.
- **`set_P_in(ch, P)`**: sets the input power for a channel
- **`get_lambda(ch)`**: retrieves the wavelength of channel **`ch`**

6.1.3 Functions for the Mode Solver

- **`set_n_profile("n_f", r_core)`** associates a refractive index profile function with the fiber. This is the basic input for the mode solver. The second argument is the core radius. The software assumes that the refractive index stays constant outside this radius.

By calling this function again, one can modify the index profile. All modes will be recalculated as required.

- **`ModeExists(l, m, lambda)`** returns 1 if the corresponding mode exists, and 0 otherwise.
- **`A_lm(l, m, lambda, r)`** is the amplitude for mode indices **`l`** and **`m`** for the wavelength **`lambda`** at the radius **`r`**. This has to be multiplied with **`cos(l phi)`** or **`sin(l phi)`** to obtain the full mode function. That function (including the azimuthal factor) is normalized such that the integral of its squared modulus over the whole area (core and cladding) is unity.
- **`A_lm_xy(l, m, lambda, x, y)`** is the amplitude for the position with Cartesian coordinates **`x`** and **`y`**, assuming the **`cos(l phi)`** dependence. For obtaining the **`sin(l phi)`** dependence, take **`A_lm_xy(-l, m, lambda, x, y)`** (i.e., the value for negative **`l`**).
- **`I_lm(l, m, lambda, r)`** is the intensity, which is simply the squared modulus of **`A_lm(l, m, lambda, r)`**. It has to be multiplied with **`cos(l phi)^2`** or **`sin(l phi)^2`**.
- **`I_lm_xy(l, m, lambda, x, y)`** is the intensity for the position with Cartesian coordinates **`x`** and **`y`**, assuming the **`cos(l phi)`** dependence for positive **`l`** and the **`sin(l phi)`** dependence for negative **`l`**.
- **`beta_lm(l, m, lambda)`** calculates the propagation constant of a mode, i.e., the phase change per unit propagation distance.
- **`n_eff_lm(l, m, lambda)`** calculates the effective refractive index of a mode.
- **`w_lm_x(l, m, lambda)`** calculates the mode radius in x direction for the mode function with the $\cos l\varphi$ dependence (or the radius in the y direction for the mode function with the $\sin l\varphi$ dependence)
- **`w_lm_y(l, m, lambda)`** calculates the mode radius in y direction for the mode function with the $\cos l\varphi$ dependence (or the radius in the x direction for the mode function with the $\sin l\varphi$ dependence)
- **`A_eff_lm(l, m, lambda)`** calculates the effective mode area of a mode, defined as

$$A_{\mathrm{eff}} = \frac{\left(\int |E|^2 \,\mathrm{d}A\right)^2}{\int |E|^4 \,\mathrm{d}A} = \frac{\left(\int I \,\mathrm{d}A\right)^2}{\int I^2 \,\mathrm{d}A} .$$

  Note that this is *not* π times the product of the mode radii in x and y direction.

- **`p_core_lm(l, m, lambda)`** calculates the fraction of the mode power which is guided within the core.
- **`v_g_lm(l, m, lambda)`** calculates the group velocity of a mode.
- **`n_g_lm(l, m, lambda)`** calculates the group index of a mode.
- **`GVD_lm(l, m, lambda)`** calculates the group velocity dispersion (in s^2/m) of a mode.

- **GVD_ps_lm(l, m, lambda)** calculates the group velocity dispersion of a mode in units of ps / (nm km).
- **NoModes(lambda)** returns the number of modes, not taking into account polarization. For example, this would be 1 for a single-mode fiber (with the LP_{01} mode only) or 2 for a two-mode fiber with LP_{01} and LP_{11}.
- **NoModes_h(lambda)** is similar to **NoModes(lambda)**, but it takes into account the different orientations (the $\cos l\varphi$ and $\sin l\varphi$ versions) for modes where **l** is not zero. For example, it is 3 for a fiber with the modes LP_{01} and LP_{11}, because the latter has a $\cos l\varphi$ and a $\sin l\varphi$ version.
- **l_max(lambda)** returns the maximum *l* value for all modes.
- **m_max(lambda)** returns the maximum *m* value for all modes.
- **cut_off(l, m)** returns the cut-off wavelength of a mode, i.e., the wavelength below which the mode exists.

6.1.4 Functions for Numerical Beam Propagation

- **bp_set_device(j)** defines a beam propagation device, which is referenced with a number between 1 and 100. Following function calls for defining a grid or optical channels, for example, always refer to the currently set beam propagation device.
- **bp_set_direction(backward)** reverses the propagation direction, and **bp_set_direction(forward)** reverses it again.
- **bp_set_R('R_x(z)', '0')**, for example, means that the curvature radius in *x* direction is given by a previously defined function **R_x(z)**, whereas there is no curvature in *y* direction. (An infinite radius of curvature is specified as zero.)
- **bp_set_grid(x_max, N_x, y_max, N_y, z_max, N_z, N_s)** defines the numerical grid of an optical channel. It covers the *x* range from **–x_max to +x_max** with **N_x** steps. The step size is **2 x_max / N_x** (note the factor 2), and the number of steps must be a power of 2. Analogous rules apply to the *y* direction. The *z* range is from 0 to **z_max** with **N_z** steps, so that the *z* step size is **dz = z_max / N_z**. Each **dz** step is numerically sub-divided into **N_s** sub-steps. (The value of **N_s** can also be changed automatically during the simulation – see the function **bp_set_N_s()**.)
- **bp_set_reflection('x')** sets reflective boundaries of the grid in the *x* direction only). Similarly, use **bp_set_reflection('y')** or **bp_set_reflection('xy')** (for reflections at all boundaries).

  bp_set_reflection('(x/a_x)^2+(y/a_y)^2 >= 1') (as an example for a boundary within the grid) defines an elliptical reflecting boundary: the reflecting part is the part where the expression evaluates to a non-zero value.
- **bp_set_n2('n2(x,y)')** sets the nonlinear index of the material as an expression which can depend on **x** and **y**.
- **bp_set_XPM(r)** sets the relative strength of cross-phase modulation (XPM). The default value of the argument is 2, but for cross-polarized channels one may use 2/3.

Caution: do not confuse the mentioned functions with the functions **set_n2()** and **set_XPM()**, which apply to ultrashort pulse propagation models (see section 5.15).

- **bp_set_N_dop('Yb', 'N_dop(x,y)')** defines the type of laser-active ions in a fiber and their concentration (in units of 1 / m^3) as a function of **x** and **y**.
- **bp_set_I_p('I_p(x,y,z)', lambda_p)** defines a pump wave with a certain wavelength (first argument) and a given intensity distribution (second argument).
- **bp_set_dt(dt)** defines a time interval for dynamic simulations. It determines for how long the fields act on the laser-active ions after each propagation step.
- **bp_do_pumping(t)** simulates the influence of the pump wave only for a given time. If there is no pump wave, it only simulates the decay of excitation.
- **bp_set_excitation(x, y, z, n)** sets the degree of excitation of laser-active ions (between 0 and 1) for some position.
- **bp_excitation(x, y, z)** retrieves the degree of excitation of laser-active ions (between 0 and 1) at some position.
- **bp_set_alpha(alpha)** defines the linewidth enhancement factor, i.e., a parameter of the amplitude/phase coupling. The phase change in one *z* step is the linewidth enhancement factor times the amplitude gain.
- **bp_define_channel(lambda)** defines an optical channel for beam propagation, where **lambda** is the vacuum wavelength of the channel, and the function result is the number of the channel (e.g., 2 for the second defined channel).
- **bp_set_n(func$)** defines the refractive index profile. The argument is a string which defines a mathematical expression depending on *x* and *y*. For example, **'n_f(sqrt(x^2+y^2))'** would refer to a previously defined function of the radial coordinate.
- **bp_set_n_z(func$, test$)** defines a a refractive index function which may also depend on *z*. The second argument is a function which is used to determine whether the refractive indices for a certain *z* position must be recalculated: they are recalculated only if the value of the test function changes.
- **bp_set_n_smooth(r)** causes the calculated refractive index profile to be smoothed; **r** can be between 0 (no smoothing) and 10 (strong smoothing). 1 means that mostly only the directly neighbored points are used for smoothing.
- **bp_set_N_s('N_s(z)', u)** assigns a *z*-dependent expression (first argument) for recalculating **N_s** after each *z* step. The value must be between 0 (effectively taken as 1) and 1000. If the second parameter is not zero, the spatial factors (resulting from refractive index and curvature radii) will be recalculated in each sub-step.
- **bp_set_loss(func$)** defines a linear loss profile. The argument is a function of *x* and *y*, and its values indicate the local propagation losses in units of dB/m. For example, with **bp_set_loss('((x^2+y^2)/(10 um)^2)^4')** one would define a loss which sharply rises at the edges of the grid.
- **bp_set_A_factors(z, 'f%(x,y)')** defines a complex factor, which will be applied to the complex field amplitudes in the spatial domain when a certain *z* position is reached.

(Such factors can be defined for multiple *z* positions.) If the given *z* value is negative, these amplitudes will be applied after *every z* step.

- **bp_set_A_f_factors(z, 'f%(f_x,f_y)')** does the same as the function **bp_set_A_factors()**, but in the spatial Fourier domain.
- **bp_set_A0(func$)** defines the initial amplitude profile of the current channel as a function of *x* and *y*. For example, **'sqrt(P/(0.5*pi*w0^2)) * exp(-(x^2+y^2)/w0^2)'** would define a Gaussian function with power **P**, Gaussian radius **w0** and zero complex phase. Amplitudes are scaled such that their modulus squared is the optical intensity (power per unit area).
- **bp_set_A(z, func$)** defines an amplitude profile just as **bp_set_A0(func$)**, but at an arbitrary z position.
- **bp_copy_A(d,ch,z)** copies a field distribution from some beam propagation device **d** and channel **ch** at position **z** to the initial field distribution of the current propagation channel. (The involved optical channels must have the same wavelength.) One may, for example, copy the field at the end of the same propagation channel in order to start a further propagation without requiring more memory. If the two fields have different numerical grids (in terms of size or resolution), quadratic interpolation is used for calculating the new field values.
- **bp_set_color(j)** defines the color setting of the channel in the profile viewer window (see section 4.10). The argument 1 corresponds to the first possible setting, 2 to the second, etc.
- **bp_set_label(s$)** sets a text label for the current optical channel, which can be displayed in the beam profile viewer.
- **bp_remove_channel(j)** removes an optical channel, e.g. in order to save memory.
- **bp_A%(x, y, z)** delivers the complex amplitude for the given position and the currently selected optical channel.
- **bp_I(x, y, z)** is the optical intensity (the squared modulus of the amplitude).
- **bp_I_max(z)** is the maximum optical intensity of a profile.
- **bp_P(z)** is the optical power, i.e., the intensity integrated over the full area.
- **bp_x_m(z)** is the mean *x* position of the beam profile for a given *z* coordinate. It is calculated as the “center of gravity” of the intensity distribution. In an analogous way, **bp_y_m(z)** is the mean *y* position.
- **bp_w_x(z)** and **bp_w_y(z)** calculate the beam radius in *x* and *y* direction, respectively, based on the second moment of the intensity profile.
- **bp_A_eff(z)** calculates the effective mode area as $A_{\mathrm{eff}} = \dfrac{\left(\int I \; \mathrm{d}A\right)^2}{\int I^2 \; \mathrm{d}A}$. Note that this is *not* directly related to the beam radii based on the second moment of the intensity profile; the ratio of these different radii depends on the shape of the intensity profile.

- `bp_A_f%(fx, fy, z)` delivers the amplitude distribution in Fourier space. (A two-dimensional Fourier transform in the *x*-*y* plane is used.)
- `bp_theta_x(z)` and `bp_theta_y(z)` calculate the half-angle divergence of the beam, if the beam profile at that location were released to vacuum.
- `bp_M2_x(z)` and `bp_M2_y(z)` calculate the beam quality M^2 factors in *x* and *y* direction, respectively. If the wavefronts are already known to be flat, as is the case e.g. for modes in lossless fibers, one can use the faster versions `bp_M2_x_col(z)` and `bp_M2_y_col(z)`.
- `bp_set_interpol(1)` leads to linear interpolations in the *x*-y plane for later function calls e.g. to `bp_I(x, y, z)`. With the argument 2 instead of 1, one obtains quadratic interpolation, and the argument 0 switches off the interpolation.
- `bp_set_storage(x_max, N_x, y_max, N_y, N)` defines a store region, having a numerical grid with given extensions and number of points in *x* and *y* direction. After that function call, beam profiles can be stored with an index between 0 and `N`.
- `bp_store_profile(z, j)` stores the current beam profile at a given `z` position in the storage region, using the storage place `j`.
- `bp_get_stored()` makes the stored beam profiles accessible with functions like `bp_I(x, y, z)` or `bp_P(z)`.
- `bpv_show(r)` shows (for non-zero `r`) or hides the beam profile viewer.
- `bpv_set_device(j)` sets a certain beam propagation device as the source for the viewer.
- `bpv_set_channel(j)` set a certain beam propagation channel.
- `bpv_set_domain(s$)` sets the spatial domain with the argument `'real space'` or `'Fourier space'`.
- `bpv_set_interpolation(s$)` sets the interpolation with the argument `'none'`, `'linear'` or `'quadratic'`
- `bpv_set_resolution(s$)` sets the resolution with the argument `'coarse'`, `'middle'` or `'fine'`.
- `bpv_set_color(s$)` sets the color with the argument `'default'`, `'red'`, `'green'`, `'blue'` or `'user'`.
- `bpv_set_plot(s$)` sets the plot with the argument `'intensity'`, `'amplitude'` or `'function'`.
- `bpv_set_function(s$)` sets a plot function.
- `bpv_set_log_scaling(r)` switches the logarithmic scaling on (`r` ≠ 0) or off (`r` = 0).
- `bpv_set_remove_fast_phase(r)` switches the removal of the fast phase variation in *z* direction on (`r` ≠ 0) or off (`r` = 0).
- `bpv_set_plane(s$)` sets the display plane with the argument `'xy'`, `'xz'` or `'yz'`.
- `bpv_set_show_parameters(r)` switches the display of parameters on or off.

- **`bpv_set_show_grid_lines(r)`** switches the display of grid lines on or off.
- **`bpv_set_scaling(j)`** sets the scaling of color values to some number of decibels between –10 and +10.
- **`bpv_set_zoom(j)`** sets the zoom level. The *x* and *y* coordinate ranges are reduced by the factor 2^j.
- **`bpv_set_position(r)`** sets the *x*, *y* or *z* position (depending on the display plane) to a certain value (in meters, or in 1/m if the Fourier space is set for the *yz* or *xz* plane).

6.1.5 Functions Used Specifically for Dynamic Simulations

(see section 5.12)

- **`set_P_in_dyn(ch, Filename$)`**: defines a time-dependent input power of channel **`ch`**
- **`set_R_dyn(ch, R1$, R2$)`**: defines time-dependent reflectivities at both fiber ends for a given channel
- **`set_loss_int_dyn(signal,'l1(t)','l2(t)')`**: defines time-dependent functions for the localized losses (between 0 and 1) at the left and right side of the fiber.
- **`set_delays(T1, T2)`**: defines additional time delays (in seconds), occurring on the left and right side of the active fiber, respectively (for dynamic laser simulations)
- **`calc_P_dyn_z(ch,1)`**: indicates that position-dependent dynamic powers have to be stored for that optical channel
- **`calc_n_dyn_z(1)`**: indicates that position-dependent dynamic excitations have to be stored for that optical channel
- **`calc_dyn(t1,t2,dt)`**: starts a dynamical simulation
- **`calc_dyn_cond(t1,t2,dt,cond$)`**: starts a dynamical simulation, which will done only as long as a condition is fulfilled.
- **`P_out_dyn(ch,t)`**: output power from a channel in a dynamic simulation
- **`P_dyn(ch,z,t)`**: power at position **`z`** of a channel in a dynamic simulation
- **`P_out_dyn_max(ch)`**: maximum output power of a channel
- **`t_peak(ch)`**: temporal position of the maximum output power
- **`n_av_dyn(l,t)`**: time-dependent average fractional excitation of the level **`l`** of laser-active ions (averaging over the whole fiber in the longitudinal and transverse direction)
- **`sp_gain_dyn(ch,t)`**: time-dependent single-pass power gain of a channel

6.1.6 Functions Used for Ultrashort Pulse Simulations

(see also section 5.15)

Defining the Numerical Grid:

- **`set_pulse_grid(T,N,lambda)`** defines the numerical grid for the time domain. **`T`** is the width of the temporal range, **`N`** is the number of numerical samples (which has to be a power of 2), and **`lambda`** is the center wavelength.

Definition of the Initial Pulse:

- **startpulse_G(E,tau,chirp)** defines a Gaussian-shaped initial pulse where **E** is the pulse energy, **tau** the pulse duration (full width at half maximum) and **chirp** the chirp parameter (rate with which the optical frequency changes, in units of Hz/s). The spectral maximum is assumed to be at the center wavelength of the chosen optical channel.
- **startpulse_s(E,tau,chirp)** defines a sech^2-shaped pulse. That pulse shape (with no chirp) is typical for fundamental soliton pulses.
- **startpulse_t("A0%")** defines an arbitrary initial pulse based on a temporal pulse profile. The argument is the name of a previously defined function for the pulse profile, given by a complex envelope function **A0%(t)**.
- **startpulse_f("A0%")** defines a spectral profile for the initial pulse, defined by a function **A0%(f)** where **f** is the absolute optical absolute frequency.

Definition of the Fiber Properties and Fiber-specific Simulation Parameters:

- **set_GVD(GVD)** defines a constant group velocity dispersion (in units of s^2/m) of a fiber.
- **set_GVD_l("GVD_fiber(l)")** defines a wavelength-dependent group velocity dispersion of the fiber. The argument can be any expression depending on the wavelength **l**.
- **set_n_eff("n_eff(l)")** defines a wavelength-dependent effective refractive index to the fiber. It is possible to use the mode solver for obtaining the wavelength-dependent effective index, see the function **n_eff_lm()**.
- In case that the function **pp_fiber_2p()** is used for sending two pulses through a fiber, the second pulse will by default see the same dispersion as the first one. However, one can call the function **set_GVD2()**, **set_GVD_l2()** or **set_n_eff2()** (in analogy to the functions described above, and after calling these) to define a different dispersion for the second pulse.
- **set_n2(n2)** sets the nonlinear refractive index of the fiber. The argument must be the value in units of m^2/W – for example, 2.6e–20 for silica. Together with the effective mode area of the optical channel, this determines the strength of the nonlinearity.
- **set_ss(s)** defines a coefficient for nonlinear self-steepening. If the given value is 1, the normal magnitude of the self-steepening effect is taken. This is appropriate if the nonlinear index and the fiber's effective mode area are not wavelength-dependent. One may use larger values for simulating the influence of the wavelength dependence of the nonlinear index and the effective mode area.
- **set_dnr("h_R(t)",f_R)** defines the parameters of a delayed nonlinear response, which leads to stimulated Raman scattering. The first argument is the (previously defined) Raman response function. **f_R** is a parameter between 0 and 1, which specifies the relative strength of the delayed nonlinear response (see section 2.9.3).
- If the function **pp_fiber_2p()** is used for sending two pulses through a fiber, one can set the strength of the nonlinear interaction (essentially, cross-phase modulation) with the function **set_XPM(x)**. Here, **x** is a dimensionless parameter, the default value of which is 2. That means that the phase change of pulse 2 per watt of power of pulse 1 is 2 times that resulting from self-phase modulation. This is the situation when two pulses

propagate in the same optical channel with the same polarization direction. If the polarization directions are linear but mutually orthogonal, and the medium is isotropic (e.g., a glass), the correct value of **`x`** would be 2 / 3. That value could also be modified to take into account a reduced spatial overlap, for example.

- **`set_gain_sat(r)`** defines a factor which artificially modifies the strength of gain saturation (but only for ultrashort pulse simulations). For example, gain saturation is ignored if the argument is 0.

  For many passes of weak pulses through an amplifier, one may do one of 100 propagation simulations after **`set_gain_sat(100)`**, and all the others after **`set_gain_sat(0)`**. That way, gain saturation has to be simulated only once, which saves some computation time.

- **`set_pulse_tolerance(tol)`** sets a tolerance for the accuracy of the ultrashort pulse simulations. The default value of the argument would be 10^{-2}. This means that the square root of the time- or frequency-integrated amplitude errors is of the order of 10^{-2} times the square root of the pulse energy. If a higher accuracy is required, the computation time may be longer, and in some cases the desired accuracy might not be achieved. Normally, that condition triggers a numerical error, aborting the calculation. That abortion can be suppressed, however, by calling the function with a *negative* argument. The tolerance is then the modulus of that argument.

- **`set_auto_center(1)`** activates automatic centering of the pulse in the time domain after each propagation step. This means that the pulse is prevented from "walking away". That setting can be switched off again by calling the function with the argument 0.

Propagating a Pulse through an Optical Element:

A number of functions can be used to propagate the current pulse through various kinds of optical elements:

- **`pp_fiber(fn,ch)`** propagates the pulse through the fiber **`fn`**, using the optical channel **`ch`**. The fiber number **`fn`** is usually 1, but can be different if different fibers have been defined using **`set_device()`**. The assigned optical channel (see section 5.6) determines the propagation direction (forward or backward), the intensity profile (which is relevant for nonlinear effects) and the background losses for the pulses. The resulting pulse will be the pulse as it exits the fiber at the other end.

  Note that before **`pp_fiber(ch)`** is called, one has to define the fiber details as explained above.

  With the function **`get_pulse(z)`**, one can later obtain the pulse at a particular position within the fiber. (Note that the next pulse propagation will start with that pulse, so you may have to call **`get_pulse(L_f)`** later on to ensure further propagation with the correct pulse, assuming propagation in forward direction.)

- **`pp_fiber_2p(fn,ch1,ch2,p2_in,p2_out)`** propagates two different pulses through a fiber where they can interact via nonlinearities. The optical channels **`ch1`** and **`ch2`** of the two pulses can be the same, but do not have to be the same. While the first pulse is taken from the current pulse, the second pulse is taken to be the stored pulse (see the function **`store_pulse()`**) with index **`p2_in`**, and the second pulse after the interaction will be stored with the index **`p2_out`**. The second pulse at a position within the fiber can be obtained with **`get_pulse2()`**.

Note that two pulses could in principle also be combined in a single time or frequency trace and propagated via **pp_fiber()**. However, **pp_fiber_2p()** is better suited if the pulses have very different center wavelengths. In that situation, the frequency trace would have to be extremely wide when using **pp_fiber()**.

- **pp_dispersion(D2,D3,D4)** applies chromatic dispersion of second, third and fourth order. The units of these parameters are s^2, s^3 and s^4, respectively. (For dispersion of arbitrary orders, use **pp_multiply_f()**, see below.)
- **pp_compress(order)** calculates the dispersive compression of a pulse. The argument of the function call determines the details of the compression:
  - If it is zero, a perfect compressor is assumed, which set the spectral phase at all frequencies to zero.
  - If it is between 2 and 6, chromatic dispersion up to the corresponding order is applied such that the peak power of the compressed pulse is maximized.
  - If it is between -2 and -6, chromatic dispersion up to the corresponding order (modulus of the argument) is applied such that the duration of the compressed pulse is minimized.

  The optimization for maximum peak power is usually preferable. Optimizing for minimum pulse duration can lead to slightly different results.
- **pp_loss(fl)** simulates a wavelength-independent linear loss. The argument is the fractional loss of power (e.g, 0.10 for 10% loss). For wavelength-dependent losses, use the function **pp_multiply_f()** (see below).
- **pp_multiply_t(f_t%[])** multiplies the whole time trace (i.e., the temporal amplitudes) with the contents of an array with the given name. The array needs to have an appropriate index range; normally, it covers the whole time trace, although this is not strictly required. The step size in the array does not have to coincide with that of the time trace; linear interpolation will be applied.

  That function can be used, for example, to model the effect of an active mode locker (with time-dependent transmission or phase modulation).
- **pp_multiply_f(f_f%[])** works like **pp_multiply_t()**, but in the frequency domain. That function can be used, for example, for simulating spectral filters or for chromatic dispersion with arbitrary wavelength dependence.
- **pp_SPM(gamma)** applies a nonlinear optical element where self-phase modulation (SPM) occurs. The argument is a nonlinear coefficient with units of rad/W.
- **pp_sat_abs(loss,tau,E_sat)** applies a saturable absorber with some initial power loss (e.g., 0.10 for 10%), a recovery time **tau** and the saturation energy **E_sat**. If the recovery time is zero, one has a fast saturable absorber (adjusting the losses instantaneously to the optical power), and the last argument is the saturation power (not energy).
- **pp_TPA(beta)** applies an element with two-photon absorption (TPA). The optical power in the time domain is attenuated according to $P_{\mathrm{out}}(t) = \exp(-\beta P(t)) \cdot P$ where $P(t)$ is the incident power.

- `pp_noise(p)` adds white (frequency-independent) random noise to a pulse. The argument directly determines the noise power, if it is positive. If it is negative, its modulus defines a noise power relative to the quantum noise limit. This noise (with `p` = -1) can be used, for example, for adding quantum noise to a start pulse, such that not only stimulated Raman scattering, but also spontaneous Raman scattering in a fiber can be simulated: the quantum noise acts as a seed for Raman amplification.

- `pp_center(w)` centers the pulse in the time domain, i.e., it shifts the time trace such that the maximum power occurs at $t = 0$. If the argument is non-zero, amplitudes at the edges of the time trace can "wrap around". With the function `dt_c()` one can obtain the total temporal shift applied by all calls of the `pp_center()` function.

- `pp_shift_t(dt,wrap)` shifts the temporal pulse envelope (without affecting the phase) by `dt`. If `wrap` is not zero, the amplitudes can wrap around at the edges of the time trace.

- `pp_shift_phi(dphi)` changes the optical phase of the whole pulse by `dphi`.

- `pp_add_pulse(j,f%)` adds the amplitudes of the stored pulse with index `j`, multiplied with `f%`, to the current pulse. This may be used for modeling interferometers, for example.

After each such function call, one can obtain the pulse properties with the functions explained in the next section. One can apply any sequence of optical elements.

Getting Temporal Pulse Properties

- `P_t(t)` returns the time-dependent optical power in units of W = J/s. (Do not confuse this with the function `P(ch,z)`.)

- `A%(t)` is the complex pulse envelope function, normalized such that its squared modulus is the optical power.

- `E(t)` is the electric field strength, defined as $E(t) = \mathrm{Re}(A\%(t)\, e^{-i\omega t})$ where ω is the angular frequency corresponding to the center wavelength. It is normalized such that its squared modulus is the optical power.

- `Phi_t(t)` is the temporal phase (i.e., the phase of `A%(t)`), projected to the range $-\pi$ to $+\pi$. There is also the function `Phi_t_c(t)` which is a continuous phase.

- `df_i(t)` is the instantaneous frequency deviation from the reference frequency, calculated as the derivative of the temporal phase divided by 2π.

- `P_p()` is the peak power.

  Note that a quadratic interpolation around the pulse maximum is used – the value can therefore be a little higher than the highest point in the time trace.

- `t_p()` is the temporal position of the pulse maximum (using quadratic interpolation around the pulse maximum), and `t_m()` is the mean position of the pulse.

- `tau_p()` is the pulse duration (FWHM = full width at half maximum).

- **tau_p2()** is the pulse duration calculated as twice the second moment of power (which is more appropriate for complicated pulse shapes):

$$\text{tau\_p2} \equiv \sqrt{\frac{\int t^2 P(t)\,\mathrm{d}t}{\int P(t)\,\mathrm{d}t}}$$

- **tau_p_th(th)** is the pulse width, where the threshold **th** specifies how far below the peak we measure the width. For example, **th = 0.5** gives the FWHM (full width at half maximum).
- The functions **t_min()**, **t_max()** and **ts()** return the range and step size of the time trace.

Getting Spectral Pulse Properties

- **P_f(f)** returns the power spectral density with respect to the absolute optical frequency in units of J/Hz.
- **P_l(l)** returns the power spectral density with respect to the wavelength in units of J/m.
- **A_f%(f)** is the frequency-dependent complex amplitude (with **f** being the absolute frequency value, not relative to the reference frequency).
- **Phi_f(f)** is the spectral phase (i.e., the phase of **A_f%(f)**), projected to the range $-\pi$ to $+\pi$. There is also the function **Phi_f_c(f)** which is a continuous phase. Similarly, there are the functions **Phi_l(l)** and **Phi_l_c(l)** with wavelength arguments.
- **T_g(f)** is the group delay of the spectral component with absolute frequency **f**.
- **f_p()** is the frequency where the spectral power density **P_f(f)** has its peak (using quadratic interpolation around the maximum). **l_p()** is the wavelength where **P_l(l)** reaches its maximum.
- **P_f_max()** is the maximum value of **P_f(f)**, and **P_l_max()** is the maximum value of **P_l(l)**.
- **f_m()** is the mean frequency, and **l_m()** the corresponding wavelength.
- **df_p()** is the pulse bandwidth (FWHM) in Hz, **dl_p()** the pulse bandwidth (FWHM) in the wavelength domain (in meters).
- **df_p2()** is the pulse bandwidth calculated as twice the second moment of the power spectral density in the frequency domain.
- **df_p_th(th)** and **dl_p_th(th)** calculate the width in the frequency and wavelength domain with a threshold condition analogous to that for **tau_p_th(th)**.
- The functions **f_min()**, **f_max()** and **fs()** return the range of the frequency trace (with absolute frequencies, not values relative to the reference frequency).
- **I_tf(t,tau,f)** is the spectral intensity around the time **t**. It is obtained by multiplying the temporal amplitudes with a Gaussian function with FWHM **tau** and then taking the spectrum of this. The result is the power density in the frequency spectrum (in W/Hz) at the absolute frequency **f**. This function can be used e.g. to draw spectrograms.

- **`I_tl(t,tau,l)`** is analogous to **`I_tf(t,tau,f)`** but works with a wavelength argument.
- **`I_tf_max(t,tau)`** gives the maximum value of **`I_tf(t,tau,f)`** with variable **`f`**.

Getting Global Pulse Properties

- **`E_p()`** returns the pulse energy.
- **`TBP()`** is the time–bandwidth product, calculated from the FWHM values of pulse width and bandwidth. It is 0.315 for an unchirped soliton pulse, for example.
- **`TBP2()`** is a similar quantity, calculated from the second moments of the pulse power in the time and frequency domain, and normalized so that it becomes 1 for the ideal case of an unchirped Gaussian pulse. In contrast to **`TBP()`**, the latter quantity takes into account satellite pulses.
- **`pulse_error$()`** returns an error string for the current pulse, if it is not valid (otherwise an empty string). This can be used to find out the reason of a problem if a pulse has not been calculated successfully.

Storing and Recalling Pulses

- A pulse can be saved in a file with **`save_pulse_t(n$)`**, where the function parameter is the filename. This will save the time-domain information. With **`save_pulse_f(n$)`**, one can save the frequency-domain information. In any case, the other domain will be recalculated automatically after loading if needed.
- A pulse can be loaded from a file with **`load_pulse(n$)`**, where the parameter is the filename.

  The save and load functions return the value 1 if successful, otherwise 0.
- One can store up to 10 000 pulses in the main memory. These pulses can be distinguished with a storage index between 0 and 9 999. One can store the current pulse with **`store_pulse(j)`** and recall it later with **`recall_pulse(j)`**. The latter function returns 1 if successful, otherwise 0.

Changing Settings of the Pulse Display Window

The following functions can be used for automatically modifying or retrieving settings of the pulse display window (see section 4.10):

- **`pdw_show(x)`** displays the pulse display window (just as via the menu with **Window | Pulse display window**), if the function argument is non-zero.
- **`pdw_display_t("phase")`** determines what quantity is displayed in the time domain. The argument can be any value in the corresponding drop-down box. **`pdw_display_f("phase")`** does the same for the frequency domain.
- **`pdw_autoscale_t(x)`** and **`pdw_autoscale_f(x)`** turn on the automatic scaling for the time and frequency domain, if the function argument is non-zero.
- **`pdw_statistics(x)`** turns on the statistics display, if **`x`** is not zero.
- **`pdw_highres(x)`** turns on the higher resolution for the statistical data, if **`x`** is not zero.
- **`pdw_grid(x)`** turns on the display of grids in the coordinate systems, if **`x`** is not zero.

- **pdw_steps(x)** turns on the display of the numerical steps, if **x** is not zero.
- **pdw_set_position_mode(x)** turns on the display of stored pulses (see function **store_pulse()**), if **x** is non-zero, or otherwise returns to the display of pulses within the fiber.
- **pdw_set_position(p)** sets the position. In the mode displaying stored pulses, the position is a pulse index. Otherwise, it is a *z* value in the fiber.
- **pdw_get_position(p)** gets the position from the form. In the mode displaying stored pulses, the position is a pulse index. Otherwise, it is a *z* position in the fiber.
- **pdw_lastcalc(x)** turns on the display of the last calculated pulse, if the function argument is non-zero.

7 How to Achieve Certain Things

7.1 How to Get Started with RP Fiber Power

As a beginner with that software, you may get started in the following way:

- Print out the manual file **RP Fiber Power.pdf** and have a look at this.
- Install the software on your PC.
- Make a copy of the Demo folder in your working folder. (Do not edit the originals.) Start the software by double-clicking on one of these script files in the Windows Explorer. In the software, you may open more files with File | Open. Read the files and execute them by pressing F8 (**Execute | Graphics**).
- Call the interactive input forms (section 4.4), which allow you to generate scripts simply filling out some fields and pressing the Execute button on these forms.

7.2 How to Import Spectroscopic Data

It is rather easy to utilize fiber data files which are already obtained in the form of **.inc** files from RP Photonics. They are stored in a certain folder (see section 3), and if you get any new files of this kind, just put them into that folder. (If you have problems with access rights, you may use that folder to a different location, see **Options | Change location of fiber data folder**, section 4.5.5).

If you get spectroscopic data from other sources, you first need to bring them into the right format. The software requires functions to define wavelength-dependent effective transition cross sections. (See e.g. the page on defining simple laser-active ions.) If you require transition cross sections only for a few wavelengths, you may simply define functions like

```
s_a_Yb(lambda):=
  if lambda = 975e-9
  then 2.6e-24
  else if lambda = 1030e-9
  then 48e-27

s_e_Yb(lambda):=
  if lambda = 975e-9
  then 2.5e-24
  else if lambda = 1030e-9
  then 635e-27
```

However, it is often convenient to have a function which uses tabulated data for many different wavelength values. Ideally, these data are available as a plain text file in CSV (comma-separated values) format. As an example, let us assume that your data file named **`Yb cross sections.dat`** look as follows:

```
850, 22, 0.014
852, 27.54, 0.02107
854, 32.8, 0.02864
...
```

where the first column is the wavelength in nanometers (running e.g. from 850 to 1150 in steps of 2), and the second and third column gives the absorption and emission cross sections, respectively, in units of 1e–27 m^2. You may then use the following commands in your script:

```
defarray s12[850,1150,2]
defarray s21[850,1150,2]

readlist l, s12[l], s21[l]
  from "Yb cross sections.dat"

s_a_Yb(l):=if 850e-9 <= l <= 1150e-9
  then 1e-27*s12~~[1e9*l]
s_e_Yb(l):=if 850e-9 <= l <= 1150e-9
  then 1e-27*s21~~[1e9*l]
```

The first two lines defines arrays. (The third parameter is the wavelength step size.) The **`readlist`** command loads the data from the file into the two arrays. Finally, the two functions access these arrays. The if-then structure avoids accessing array components outside the index range, and returns 0 in such cases. (It is important that cross-section functions return 0 in such spectral regions.)

More sophisticated operations may be necessary e.g. when your data points are not equidistant. You may use technical support from RP Photonics in such cases.

7.3 How to Choose Appropriate Numerical Parameters

For amplifier and laser simulations **without ultrashort pulses**, there are two types of numerical parameters for which reasonable values must be chosen:

- The number of steps N_z along the fiber (see section 2.1) should be so small that within no step there is a strong change of optical powers, except for optical channels having a very low power, so that saturation effects are weak. In a script, that number of steps is given as the second parameter of the **`set_fiber()`** function (section 5.4). In case of doubt, increase that value until the calculated output powers or similar parameters do not change significantly.
- The transverse resolution should also be high enough, i.e., the definition of doped rings should be sufficient fine. As explained in section 5.4, it can be advisable to subdivided a homogeneously doped region in order to accurately take into account that the optical intensities can substantially vary within these regions. Again, it is advisable to check whether an increased resolution has a significant effect on the calculated results.

For the simulation of **ultrashort pulse propagation**, the following additional aspects are important to know:

- It is *not* mandatory to increase the number of steps N_z along the fiber even if the fiber nonlinearity, for example, leads to strong changes of a pulse in a single *z* step. The used algorithm automatically does the required number of smaller sub-steps to maintain the required numerical accuracy. It is only that pulse properties can later only be retrieved for pulses on the predefined numerical grid, not at intermediate positions according to the automatically done sub-steps.
- The numerical grid parameters must be chosen appropriately (see also section 7.3):
  - The temporal width **`T`** must be large enough to fully contain the temporal pulse profile – not only for the initial pulse, but for the whole propagation.

    Note: if some part of a pulse reaches an end of the time trace, it will not disappear, but rather come back on the other end.

    Note that for the simulation of stimulated Raman scattering, a high temporal resolution is required. For silica fibers, for example, the temporal resolution should be better than 10 fs, ideally even better than 5 fs.
  - The frequency resolution will be the inverse of the temporal width of the grid. Therefore, a longer temporal grid may also be required if spectrally narrow features of the dispersion profile are relevant.
  - In the frequency domain, the resolution will be **`1/T`** (i.e., the inverse of the temporal width of the grid), and the total covered spectral range will be **`N/T`**. That range must be large enough to fully contain the spectral pulse profile at all positions within the fiber.
  - The number of grid points needs to be large if one needs to cover a wide temporal range and also have a high temporal resolution (i.e., a wide frequency range). In simple case, 2^8 = 256 points or even less are sufficient, while in some other cases one needs 2^{12} = 4096 points or even more.

  It is the user's responsibility to correctly choose these parameters. A test can be to check whether the results change significantly if the numerical resolution is doubled, either by doubling **`T`** and **`N`** (using a doubled temporal range but the same spectral range) or by doubling only **`N`** (using twice the spectral range).

7.4 How to Export Calculated Data in Text Form

You may mostly use the graphical output of the software, but in some cases it is desirable to write calculated data to a text file. For example, you may need such a file for importing data into some other software. A frequently used format is the CSV (comma-separated values) format with information on one data point per line.

For that purpose, you can use a few lines of script code as described below. Note that this doesn't force you to work on the script level for everything: you can still use the interactive forms (section 4.4) and insert the script lines on the page with additional definitions (section 4.4.5). These lines will then be inserted into the generated script.

Typically, one uses a loop structure to write information on multiple data points into an output file. In a first example, we export position-dependent forward and backward-propagating signal power values into a file:

```
write "; position, forward power, backward power", >"Test.dat"
for z:=0 to L_f step L_f/No_z_steps do
 write [z:f1, ", ",
        P(signal1_fw,z):d4, ", ",
        P(signal1_bw,z):d4],
       >>"Test.dat"
```

Here, the first script line writes a one-line header into the output file. Then, the for loop is used to cycle the variable **z** through a range of values. For each value, the **write** command (section 5.18) is used to write three values into one line of the output file: the **z** position and the forward and backward power. The **>>** option indicates that the generated line is appended to the existing file.

In case that a header line at the beginning is not required, one can also use the following version:

```
write [z:f1, ", ",
       P(signal1_fw,z):d4, ", ",
       P(signal1_bw,z):d4],
      for z:=0 to L_f step L_f/No_z_steps,
      >"Test.dat"
```

Here, a single **write** command is used, having the for option for writing multiple lines.

In other cases, you need to modify certain model inputs within the loop. The powers then have to be recalculated accordingly. For example, we may vary the signal input power of an amplifier and write it together with the corresponding signal output power to a file:

```
write [P_in:f2, ", ",
       (set_P_in(signal1_fw,P_in); P_out(signal1_fw)):d4],
      for P_in:=0 to 0.5 step 0.05,
      >"Test.dat"
```

Here, the composite expression in the second line first sets the input power of the channel **signal1_fw**, which causes a recalculation of all optical powers and excitations, and then retrieves the new output power.

7.5 How to Trace Errors in a Script

It is unavoidable that occasional errors occur when you develop a script. However, this software gives you some means to trace down within a reasonable time. Here are some hints.

If a script cannot be executed because the script interpreter experiences a serious error (e.g. a syntax error), try to get some clues by reading the content of the log area (section 4.7) in the lower left part.

More difficult situations can arise if a script executes, but doesn't do what you expected. The following actions may be helpful:

- Within your script, place some **show** commands (section 5.17) to display some quantities in the Output area. This may give you clues for what might have gone wrong.

- If you want to try out some simplified versions of script code, you may temporarily "comment out" some lines of codes by inserting one line with `(*` before them and one line with `*)` after them.
- Consult the manual or the online help about the functions you used. Make sure you correctly understood what they are doing.

As a general recommendation, introduce more sophisticated features only step by step, and do some tests after each step. Otherwise, it might become difficult to find out which step was wrong.

In difficult cases, consider to call technical support (section 1.5).

7.6 How to Model Double-clad Fiber Devices

In an active fiber with a double-clad design (see http://www.rp-photonics.com/double_clad_fibers.html), there is a fiber core for a signal or laser wave, which is usually single-mode or supports only few propagation mode, whereas the pump light is injected into a substantially larger inner cladding. Only the core is doped with laser-active ions. Due to some overlap of the intensity distribution of the pump light with the fiber core, some pump light is absorbed, and this can be utilized for the laser process in the core.

The easiest way of modeling a cladding-pumped device with this software is to define a pump channel with a large intensity distribution according to the area of the inner cladding. Typically, one assumes a top-hat intensity profile filling the whole cladding and core region. Note, however, that the model then rests on the important assumption that the intensity distribution in the inner cladding remains unchanged during propagation in the fiber. Particularly for fibers with a simple design, having a centered core in a circular inner cladding, this assumption may be violated, because the power of some modes of the inner cladding is preferentially absorbed, and after some distance the mode distribution is such that the absorption becomes substantially weaker. The intensity distribution then exhibits a dip in the core region.

There are various methods to mitigate that problem. One may, for example, using a cladding shape with reduced symmetry (e.g. a “D” shape), or strongly bend the fiber to enhance mode mixing. Such methods may not be totally effective, i.e., the pump absorption may still be lower than assumed in the model. Unfortunately, it would be very difficult (for various reasons) to model such situations accurately.

More accurate modeling is possible, however, in the case of a radially symmetric fiber design, provided that the number of cladding modes is not too large. This is done in the demo file described in section 8.14. Here, one uses the mode solver to calculate all cladding modes. The question arises then how the input pump power is distributed over the cladding modes; this depends on the beam properties of the pump source and the coupling optics, of course also the alignment. A first guess is an equal distribution of the power over all cladding modes, although this will generally not correspond to a smooth intensity distribution.

7.7 How to Model Mode Locking of a Fiber Laser

Mode locking of fiber lasers is an advanced and sophisticated scientific topic, as there are many different operation modes of such lasers, and these often exhibit sophisticated details, partly arising from the strong nonlinear interactions. With **RP Fiber Power**, one can simulate the pulse evolution in a fiber laser. Basically, one will proceed as follows in the used script:

- Define the parameters of all optical components, i.e., all the involved fibers and possible other components such as filters, saturable absorbers, etc.
- After defining the pulse grid, set a start pulse with parameters roughly resembling those of the expected intracavity pulse. It is convenient to choose the position just before the output coupler as the start position in the resonator, as one can then easily obtain the output pulse from the intracavity pulse at that position.
- Define a function which subsequently transmits the current pulse through all resonator components, so that a complete round trip is performed. One may then call that function many times (e.g., 100 times), hoping that the pulse parameters converge. (Convergence may not be achieved if the resonator properties and the initial pulse properties are not chosen correctly.)
- If gain saturation is modeled realistically, it takes a huge number of round trips (possibly many millions) until the pulse energy has reached the steady state. If the gain dynamics are not of interest, it is much more convenient to recalculate the laser gain after each round trip according to the steady state which would be reached for the given input pulse energy. It is then usually sufficient to simulate only 100 round trips.
- One can define diagrams showing not only the final pulse, but also the evolution of pulse parameters in many round trips.
- It is convenient to store the pulse after each round trip, so that the pulses can be conveniently inspected later on with the interactive pulse display window.

There is a demo file for an all-normal-dispersion picosecond fiber laser, see section 8.39.

8 Demo Files

The RP Fiber Power software package contains various example scripts demonstrating the features of the software. They are contained in the folder named Demo. That folder is a subfolder in the program folder; depending on details of your Windows installation, it will be something like **`C:\Program Files\RP Software\RP Fiber Power\Demo`**. Before working with any demo files, it is recommended to make a copy of the whole demo folder in your user folder and edit only files in that copy.

The following subsections explain the demo files, which also contain a lot of comments.

8.1 Calculating Fiber Modes

File: **Fiber modes.fpw**

(The form settings file **Fiber modes.fpi** allows one to do similar things in the forms mode.)

For simplicity, this script demonstrates only the calculation of fiber modes with the integrated mode solver. The use of such modes in a fiber amplifier model is demonstrated in section 8.11.

The script works with a refractive index profile defined with tabulated values. With a few lines of script code, these are read into an array, and the index function **`n_f(r)`** uses interpolated values from that array.

It is shown then how to produce various diagrams displaying the modal properties of the fiber:

- A first diagram shows all radial functions. Different colors are used for different *l* values. The refractive index profile and the effective indices of the modes are also shown.

- The second diagram shows the intensity pattern of a selected mode.
- Diagram 3 shows the number of modes as a function of the wavelength. At 1.96 µm, the single-mode regime starts.
- Diagram 4 shows how the effective mode indices depend on the wavelength. One sees that all these indices reach the cladding index at their cut-off wavelength.
- Diagram 5 shows the fraction of power in the core for all modes versus wavelength.

8.2 Modes of a Germanosilicate Fiber

File: **Germanosilicate fiber.fpw**

(The form settings file **Germanosilicate fiber.fpi** allows one to do similar things in the forms mode.)

This script is a more sophisticated case for the calculation of the properties of fiber modes. We consider a germanosilicate multimode fiber, where a supergaussian transverse profile of the germanium concentration is assumed. The refractive index is interpolated between that of silica and germania according to the local germania content. The refractive indices of silica and germania are calculated from Sellmeier formulas, so that the wavelength dependence is included, which we need for calculating the chromatic dispersion.

The mode solver (section 2.5) provides functions which deliver effective refractive indices, group indices, group velocity dispersion, etc., for all modes.

8.3 Launching Light into a Multimode Fiber (Mode-based Simulation)

File: **Fiber launch.fpw**

This script simulates the results of launching light into a fiber with several guided modes. The input beam is a Gaussian beam, which can be offset from the core center and tilted against the fiber axis. The script can calculate the resulting amplitudes of all modes, from which the intensity profile at the fiber output can be calculated efficiently (without numerical beam propagation!). Apart from a numerical output of the resulting mode powers, the following diagrams can be made:

- Diagram 1 shows the powers of all the guided modes as functions of the input beam position.
- Diagram 2 shows the powers of all the guided modes as functions of the input beam tilt.
- Diagram 3 shows the output intensity profile for a given beam offset.
- Diagram 4 shows the output intensity profile for a given beam tilt.

8.4 Launching Light into a Single-mode Fiber (with Beam Propagation)

File: **Launching light into a single-mode fiber.fpw**

Here, we use numerical beam propagation to investigate what happens if light is launched into a single-mode fiber with imperfect launch conditions: the input beam profile is Gaussian (not perfectly fitting to the guided mode of the fiber) and can be offset in terms of position and incidence angle. Based on the calculated fiber mode, one can already calculate the launch efficiency, but only with numerical beam propagation one learns what exactly happens in the fiber.

- Diagram 1 shows the field amplitude profile in the yz plane. One can see how some of the launched light gets into the cladding.
- Diagram 2 shows the launch efficiency as a function of the initial beam radius.

8.5 Bent Large Mode Area Fiber

File: **Bent large mode area fiber.fpw**

Here, we investigate how bending influences the propagation of light in a large mode area fiber. We use a simple step-index design, but other index profiles could be investigated as well.

- Diagram 1 shows how light evolves if the bending becomes stronger and stronger along the fiber, until all light is lost into cladding modes.
- Diagram 2 plots the propagation losses as a function of curvature radius for different modes. This calculation takes quite a few minutes, but the diagram delivers a lot of information.

8.6 Tapered Fiber

File: **Tapered fiber.fpw**

We assume that light is into a guided mode of a fiber. After some distance of propagation, the fiber core gets continuously smaller due to tapering of the fiber. The simulation shows how the mode field adapts to the core size, unless the change of core size is too fast.

- Diagram 1 shows the field amplitudes in the yz plane.
- Diagram 2 shows how the beam parameters evolve along the fiber: the optical power, beam radius, mode area, etc.

8.7 Fiber Coupler

File: **Fiber coupler.fpw**

Numerical beam propagation can be done for essentially arbitrary refractive index profiles. Here, we assume the structure of a directional fiber coupler based on evanescent wave coupling. Two fiber cores come relatively close over some distance, so that light can leak from one core to the other. We launch light into one of the two inputs and can investigate what happens for different waveguide distances, lengths of the coupling region, wavelengths, etc.

- Diagram 1 shows the refractive index profile. This is useful to check that the coupler structure is as we intend it to be.
- Diagram 2 shows the field distribution in the yz plane. One sees how some of the optical power couples over into the second waveguide.
- Diagram 3 shows the beam profile at one of the output ports.
- Diagram 4 shows the coupling strength as a function of the distance between the two cores in the coupling region.
- Diagram 5 shows the coupling strength as a function of wavelength. The coupling is weak for short wavelengths, where the evanescent fields are weak, and becomes stronger for long wavelengths, until it is reduced again because the light couples back to the original waveguide, and because the bend loss increases.

Note that the script defines separate wavelength channels for the simulations with different wavelengths, so that one can later inspect the detailed beam profiles with the interactive beam profile viewer.

8.8 Pump Absorption in a Double-clad Fiber

File: **Pump absorption in double-clad fiber.fpw**

We investigate how pump light is absorbed in a double-clad fiber. We can have a simple geometry with a centered doped core in a circular cladding, but also a D-shaped cladding, where a smaller section of its cross-section is cut off. The simple approach has the disadvantage that many modes exhibit quite poor pump absorption.

- Diagram 1 shows the field amplitudes in the yz plane. For the simple geometry, one recognizes that the beam profile develops a pronounced “hole” in the core region.
- Diagram 2 displays the final intensity profile.

8.9 Nonlinear Self-focusing

File: **Self-focusing.fpw**

Here, we first calculate how the fundamental mode of a large mode area fiber shrinks as the effect of nonlinear self-focusing.

The mode solver actually ignores nonlinear effects. However, with a few lines of script code we can store the refractive index profile including its nonlinear changes and then recalculate the fiber modes. This is repeated until we get a self-consistent solution.

The script also demonstrates the application of numerical beam propagation. It can be simulated how the beam profile evolves at high optical powers. One can, e.g. inject a superposition of the (low-power) LP_{01} and LP_{11} modes and study how the fiber nonlinearity influences the propagation. One finds e.g. that even if only the LP_{11} mode is excited originally, at high power levels the situation becomes unstable, and much of the power is transferred into the LP_{01} mode.

The following diagrams are made:

- Diagram 1 shows the mode profile for a given optical power (not far below the power for catastrophic self-focusing), and the corresponding refractive index profile. One sees that the index profile is substantially modified by the nonlinear effect.
- Diagram 2 shows the mode area versus optical power. The mode area shrinks dramatically as the critical power is approached.
- Diagram 3 shows the maximum power as a function of the core radius. For each core radius, one has to calculate the optical power for which the on-axis intensity reaches the damage threshold. Of course, the modes need to be recalculated for each power value.
- Diagram 4 shows the evolution of the beam profile, simulated with numerical beam propagation.
- Diagram 5 shows how the optical powers in the two lowest-order modes evolve.

8.10 Simple Ytterbium-doped Amplifier

File: **Yb amplifier, simple model.fpw**

(The form settings file **Yb amplifier, simple model.fpi** allows one to do similar things in the forms mode.)

(Remark: this file and the following demo files do not work, if you only have the passive edition of the software.)

This script is best suited for learning some principles of the software. It defines a simple kind of an ytterbium-doped fiber amplifier.

Both pump and signal wave propagate in a single-mode core. Each wave is represented by one optical channel. The mode profiles are defined in the script as Gaussian functions with given radii.

Amplified spontaneous emission is not considered in this model. Therefore, the model will become invalid if you reduce the input signal power such that the single-pass gain becomes too high.

The script also defines some diagrams for plotting the following:

- the optical powers versus position in the fiber
- the signal output power as a function of the pump power, or the signal input power, or the fiber length
- the transverse (radial) dependencies of doping and intensity profiles

8.11 Ytterbium-doped Amplifier with Calculated Mode Profiles

File: **Yb amplifier, calculated modes.fpw**

(The form settings file **Yb amplifier, calculated modes.fpi** allows one to do similar things in the forms mode.)

This script is the same as the last one, except that the mode profiles are calculated from the refractive index profile, given as a function **`n_f(r)`** which in that case defines a simple step-index profile. The definition of the pump and signal channel uses the calculated modes: the mode functions are given as **`'I_lm(0,1)'`**, which indicates LP_{01} modes. These are the only modes in that case, as the fiber is operated in the single-mode regime.

8.12 Multimode Fiber Amplifier

File: **Yb amplifier, multimode fiber**.**fpw**

Here, a multimode fiber is considered, where a refractive index profile and a Yb doping profile is given. The script first calculates all the guided signal modes. Then it defines the pump channel (where the pump power is assumed to be concentrated in the LP01 mode) and one signal channel for each guided mode at the signal wavelength (disregarding different polarization states).

The first diagram shows the radial mode functions and also displays the gain for each mode. The second diagram shows the power evolution of all channels.

8.13 Cladding-pumped Fiber Amplifier

File: **Yb amplifier, cladding-pumped.fpw**

(The form settings file **Yb amplifier, cladding-pumped.fpi** allows one to do similar things in the forms mode.)

This is a modified version of the demo script for a simple single-mode amplifier. Here, the pump wave is assumed to propagate in the multimode inner cladding of a double-clad fiber, whereas the signal again propagates in a single-mode core. The pump power is increased to 10 W. As the pump absorption is strongly reduced, a longer fiber is required, the pump wavelength is chosen to be 975 nm, and the doping concentration has also been increased.

It is implicitly assumed in this model that the shape of the transverse pump intensity distribution remains constant during propagation. This requires strong mode mixing in the fiber, which may to be given for a fiber with radially symmetric design.

Amplified spontaneous emission is not considered in this model. Therefore, the model will become invalid if you reduce the input signal power such that the single-pass gain becomes too high.

8.14 Cladding-pumped Fiber Amplifier, with Calculated Cladding Modes

File: **Yb amplifier, cladding-pumped, with cladding modes.fpw**

This is another demo script for an amplifier with a double-clad fiber. In contrast to the previous case, we now consider all cladding modes, which are calculated with the built-in mode solver. The modes are calculated based on the simple refractive profile with given numerical apertures of the core and pump cladding. For simplicity, it is assumed that the power is equally distributed over all pump modes. Amplified spontaneous emission is not considered.

The last diagram (diagram 5) shows the transverse intensity profiles of the pump input and output, apart from the signal output. One recognizes that the residual pump light forms a ring around the core. This is because only those pump modes remain which have a low overlap with the core region and thus experience only weak absorption. This demonstrates the commonly experienced problem with such fibers that a significant part of the pump power is not absorbed even after a fiber length which would allow for efficient absorption according to a simple model, disregarding the change in pump intensity profile. This non-absorbed pump increases if the numerical aperture of the pump cladding is increased, because then the highest-order pump modes can better avoid the core region. (Of course, one might still launch the pump power predominantly into lower-order modes, thus mitigating the problem.)

8.15 Fiber Amplifier with Dip in Doping Profile

File: **Yb amplifier, dip profile.fpw**

(The form settings file **Yb amplifier, dip profile.fpi** allows one to do similar things in the forms mode.)

This is a modified version of the demo script for a simple single-mode amplifier. Here, the doping concentration of the laser-active ytterbium ions is assumed to exhibit a dip on the fiber axis. This phenomenon can occur for some fiber fabrication methods.

On the script level, the required modification is simple: instead of a single call of the **`add_ring()`** function, there are three such function calls, defining three rings with different doping concentrations.

8.16 Two-stage Fiber Amplifier Models

Files **Yb amplifier with two stages.fpw**, **Yb amplifier with two stages, with ASE.fpw** and **`Yb amplifier with two stages for pulses.fpw`**

These files demonstrate how to model a two-stage amplifier.

The first file is a simpler version without ASE. The two stages are considered as two different devices, and function calls like **`set_device(2)`** allow one to switch between them. A function **`connect_powers()`** is defined which takes the output signal power of the first stage as an input for the second one. In diagram 3, the script varies the pump power of the first stage along the *x* axis, and plots the output power of the second stage as a function of this. At each point in the graph, it needs to switch to stage 1 in order to modify the pump power and calculate the signal output power, and then it switches to stage 2 in order to calculate the output of the second stage with the next signal input.

The second file is a more sophisticated version with ASE. Here, it is possible that backward ASE from the second stage influences the gain in the first stage. Therefore, the function **`connect_powers()`** needs to iterate until a self-consistent solution is found, even though for the parameters chosen the back-action is not very strong.

The third file also simulates the amplification of nanosecond pulses. The continuous-wave simulation is first done without any signal input power in order to obtain the small-signal gain. Then we inject a seed pulse with super-Gaussian shape. Before that happens, we can either have the amplifier in the state after long pumping or in the steady state for a given pulse repetition rate. The simulation also displays the ASE spectrum (calculating the time average over one pumping cycle) and makes a diagram where the pulse repetition rate is varied.

8.17 Fiber Amplifier with ASE

File: **Yb amplifier with ASE.fpw**

(The form settings file **Yb amplifier with ASE.fpi** allows one to do similar things in the forms mode.)

This is a modified version of the demo script for a simple single-mode amplifier. Here, amplified spontaneous emission (ASE) is considered in addition to the pump and signal wave.

In order to model the full ASE spectrum, containing different wavelength components with substantially different optical gain, ASE is described by an array of forward and backward-propagating ASE channels, rather than by only two channels:

```
l1_ASE:=950 nm
l2_ASE:=1100 nm
dl_ASE:=2 nm

defarray c_ASE_fw[l1_ASE, l2_ASE, dl_ASE]
defarray c_ASE_bw[l1_ASE, l2_ASE, dl_ASE]
For convenience, this script also defines the functions
P_ASE_fw(x):=
  sum(l:=l1_ASE to l2_ASE step dl_ASE,
      P(c_ASE_fw[l],x))
P_ASE_bw(x):=
  sum(l:=l1_ASE to l2_ASE step dl_ASE,
      P(c_ASE_bw[l],x))
```

which sum up the ASE powers (at some position x) over all wavelengths.

In this script, the ASE channels have all the same bandwidth, and their center wavelengths are equidistant. Accepting somewhat more sophisticated book keeping, this could be generalized, so that a smaller number of ASE channels is required, as a high spectral resolution is required only around the 975-nm peak of the transition cross sections.

8.18 Fiber Amplifier with ASE (with html Output)

File: **Yb amplifier with ASE (with html output).fpw**

This is an extended version of the previous demo file, where results are written to an html file, so that they can be displayed in a web browser. For that purpose, some functions are defined, which e.g. write parts of html tables.

8.19 Ytterbium-doped Fiber Laser

File: **Yb fiber laser.fpw**

(The form settings file **Yb fiber laser.fpi** allows one to do similar things in the forms mode.)

This is a simple model for an ytterbium-doped fiber laser. Both pump and signal (laser) wave are assumed to propagate in a single-mode core.

On the script level, the amplifier model is simply converted to a laser model by inserting a call of the **`set_R()`** function in the model definition. It is assumed that the left fiber end is totally reflecting for the signal (laser) wave (e.g. due to a fiber Bragg grating), whereas the output fiber end has a reflectivity of 4% (arising from the Fresnel reflection at a bare fiber end).

Note that the laser wavelength has simply been defined in the model. If there is no optical component defining that wavelength, the laser may actually chose to operate on some other wavelength, where there is more gain. There is a more sophisticated demo script which automatically determines the right laser wavelength.

8.20 Ytterbium-doped Fiber Laser, Automatic Wavelength Calculation

File: **Yb fiber laser, automatic lambda.fpw**

This is a model for an ytterbium-doped fiber laser, where the laser wavelength is calculated automatically. For that purpose, multiple signal channels with a wavelength spacing of 5 nm are defined, and the software will find out which of the channels is lasing under the given conditions. (A problem can arise in a situation where two channels have very similar gain.)

The script defines a user-defined function named **`laser_wavelength()`** to find the lasing channel – simply the channel with the highest output power.

It is interesting to look at the third graph, where the fiber length is varied. The laser wavelength is recalculated for every point, and indeed it varies substantially. For short fiber lengths, lasing occurs at 975 nm, where the emission cross section is highest. For longer lengths, however, the laser suddenly jumps to 1030 nm and then to longer wavelengths, because reabsorption of laser light at 975 nm (where there is also a high absorption cross section) becomes more important. Such behavior is characteristic for quasi-three-level laser systems.

8.21 Erbium-doped Fiber Amplifier with ASE

File: **Er amplifier with ASE.fpw**

This is similar to the demo script for an ytterbium-doped amplifier with ASE, but it works with erbium instead of ytterbium. Data for an aluminosilicate fiber are used. As these data do not include the pump absorption around 980 nm, a model for pumping at 1470 nm has been made.

In this script, ideal performance of the erbium ions has been assumed. This means that no quenching or energy transfer processes occur. There is a more sophisticated model taking account such effects.

8.22 Erbium-doped Fiber Amplifier with Quenching

File: **Er amplifier with quenching.fpw**

This is similar to the demo script for an erbium-doped amplifier with ASE, but it uses a more sophisticated model for the erbium ions, which also includes an upconversion effect. Here, two ions in the upper level of the laser transition can interact such that one ions gets to the ground state while the other ion gets transferred into a higher-lying state, which however immediately decays back to the original level. In effect, one excitation is destroyed. The overall performance of the amplifier is somewhat diminished.

8.23 Erbium-doped Fiber Amplifier for Multiple Signals

File: **Er amplifier with quenching.fpw**

Here, multiple signals at equidistant wavelength are injected into an erbium-doped fiber amplifier. These signals experience somewhat different gain values, so that the output powers are different. Also, the noise figures are displayed (see diagram 2). These are better (lower) for the longer wavelengths, where reabsorption effects are weaker.

8.24 Erbium-ytterbium-doped Fiber Laser

File: **Er-Yb fiber laser.fpw**

This is a model for a short erbium-ytterbium-doped fiber laser, where a pump beam at 975 nm excites both ytterbium and erbium ions. The excitation energy of the ytterbium ions can be transferred to erbium ions.

This kind of laser would also work without the ytterbium, as you can verify by setting the ytterbium concentration to zero. However, the pump absorption would then be very incomplete, leading to a low output power. (This is due to the short fiber length and the limited possible erbium doping concentration.) With the additional ytterbium, which can be much more highly concentrated, the laser becomes more efficient. At high pump powers, however, the limited rate of the energy transfer becomes a bottleneck, limiting the performance.

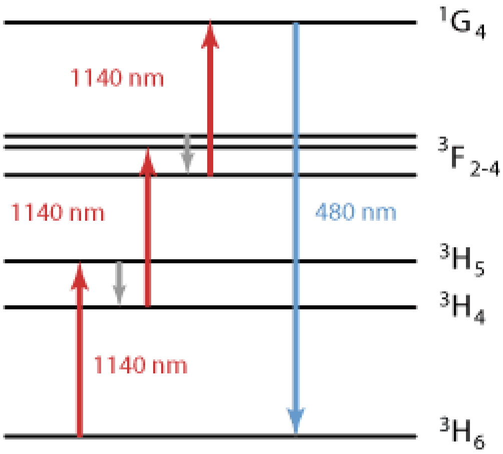
8.25 Thulium-doped Upconversion Fiber Laser

File: **Tm upconversion fiber laser.fpw**

This model illustrates how lasers or amplifiers with sophisticated level schemes can be modeled in **RP Fiber Power**.

We consider a fiber laser with the following properties:

- The fiber is made of ZBLAN glass and doped with thulium (Tm^{3+}) ions. Due to the low phonon energies of the ZBLAN glass, the levels $^3H_4$ and $^3F_2$ are metastable (not quenched by multi-phonon transitions).
- The thulium ions can be excited to a high-lying electronic level ($^1G_4$) by subsequent absorption of



three pump photons at 1140 nm. From that level, stimulated emission to the ground state generates blue laser light at 480 nm.

- The left fiber end has a totally reflecting mirror, while the other fiber end has an output coupler mirror with 60% reflectivity.

The model uses spectroscopic data as published in R. Paschotta et al., “Characterization and modeling of thulium:ZBLAN blue upconversion fiber lasers”, J. Opt. Soc. Am. B 14 (5), 1213 (1997).

8.26 Dynamic Amplifier Simulation

File: **Dynamic amplifier simulation.fpw**

(The form settings file **Dynamic amplifier simulation.fpi** allows one to do similar things in the forms mode.)

This model illustrates how a dynamic simulation for a fiber amplifier with time-dependent input powers can be done with **RP Fiber Power**.

We consider a simple model of an ytterbium-doped fiber amplifier. As a starting point for the state of the fiber, we take the steady state with given pump and signal input powers. For the dynamic simulation, we then assume a signal pulse with super-Gaussian shape, which extracts most of the stored energy. Because the gain sharply drops during amplification, the output pulse shape is substantially distorted.

See also section 8.16 for two-stage amplifier models; the third one works with pulses.

8.27 ASE Source with Automatic Spectral Optimization

File: **ASE source.fpw**

This model shows how the reflectivities in an ASE source are automatically optimized so as to achieve a flat output spectrum. An iterative procedure is implemented with a few lines of script code. Each iteration step includes the following:

- Calculate the total ASE power in forward direction, and from this the targeted ASE power per spectral channel.
- For each spectral channel, rescale the reflectivity at the left fiber end according to the ratio of target power and actual power. Rescale the whole reflectivity curve such that the maximum is 1 (= 100%).

A few iteration steps lead to the desired result.

8.28 Actively Q-switched Fiber Laser

File: **Actively Q-switched fiber laser.fpw**

This model shows how to simulate the pulse generation in Q-switched fiber lasers.

We consider a laser based on a double-clad fiber, which is pumped from both sides. The output coupler is a partially reflecting fiber Bragg grating on the right side of the active fiber, and a modulator is inserted on the left side.

As it is of interest to simulate the generation of multiple pulses and compare their properties, it is convenient to define two functions simulating the opening time and the closing time of the Q-switch, respectively. This makes it easy to simulate multiple Q-switching cycles for some graphical diagrams.

8.29 Passively Q-switched Fiber Laser

File: **Passively Q-switched fiber laser.fpw**

This model shows how to simulate passive Q switching of a fiber laser. This is somewhat tricky. For a reasonable speed, the pump phase needs to be with relatively course temporal steps, while for the pulse build-up we need very fine steps and need to take into account the propagation times. Therefore, we proceed as follows:

- We begin with a simulation of the pump phase in a simplified model, where we remove the optical feedback and thus suppress lasing. We just use this to find out at what time the round-trip gain becomes positive.
- We then do that simulation once again only for that time, so that the fiber is in the correct initial state for the next step:
- We then simulate the pulse build-up with the full model, containing the optical feedback.
- The saturable absorber, acting as a passive Q switch, is simulated with a function which returns the reflectivity based on the currently stored excitation energy. That energy is updated using the laser power of the previous temporal step.

The model allows one to study many aspects, e.g. the influence of the absorber parameters (unsaturated absorption, saturation energy, recovery time).

8.30 Actively Q-switched Nd:YAG Laser

File: **Nd-YAG laser, actively Q-switched.fpw**

This model demonstrates the application of RP Fiber Power to a bulk laser. This is no problem as long as the laser crystal is end-pumped and the beam radii remain approximately constant throughout the crystal. In the taken situation, the Rayleigh length of a Gaussian beam with beam radius of 200 µm is 215 mm, more than ten times the crystal length, so the assumption is very well fulfilled.

As the pump beam is assumed to be slightly misaligned, the radial symmetry is broken. Therefore, we use a rectangular grid rather than the usual ring structure. The resolution is chosen such that it is sufficient for properly sampling the Gaussian beams.

A dynamic simulation is done in two steps. First, the pump phase is simulated, where the Q-switch prevents lasing. This can be done with a large temporal step size, so that only a fraction of a second is required. Second, the pulse formation with open Q-switch is simulated. Here, we need a very small temporal step size (which is automatically chosen), and the calculation requires a few seconds on an ordinary PC.

Note that a finite switching time of the Q-switch is implemented. For infinitely fast switching, the temporal profile of the pulse would exhibit a significant modulation on the time scale of the round-trip time of the resonator. (This implies multiple longitudinal modes, of course.)

The following diagrams are shown:

- Diagram 1: build-up of stored energy and signal gain in the pump phase. The slope reduces more and more since an increasing amount of fluorescence radiates away energy.
- Diagram 2: transverse profile of the neodymium excitation in the crystal after the pump phase. This reflects the shape of the pump beam.

- Diagram 3: transverse profile of the neodymium excitation after pulse generation. As the signal beam is slightly smaller than the pump beam, and the pump beam is misaligned, some excitation remains mainly on the right side.
- Diagram 4: evolution of output power and gain during the pulse formation. The pulse energy and peak power, among other data, are displayed in the diagram.
- Diagram 5: evolution of output power and gain, shown on a logarithmic scale. One can see how the laser power grows exponentially over several orders of magnitude, starting with a very low level from spontaneous emission.

See also the following demo file, which treats a similar kind of laser.

8.31 Actively Q-switched Nd:YAG Laser, with Beam Propagation

File: **Nd-YAG laser, actively Q-switched, with beam propagation.fpw**

This model simulates a similar laser as the previous one. An important difference, however, is that it does not rest on the assumption that the laser beam radius stays constant. Numerical beam propagation is used to check that. Indeed it turns out that gain guiding has some effect – which is usually not that strong, but can become in a certain situation where resonant mode coupling can occur.

The following diagrams are shown:

- Diagram 1: temporal evolution of power and beam radius. Various data are stored after each round-trip. (That diagram must be made for the other diagrams to work.)
- Diagram 2: spatial profile of the Nd excitation after the pulse.
- Diagram 3: evolution of beam profile during the pulse.

If one sets the variable L_air (distance between the crystal and the end mirror) to 25 mm (instead of 20 mm or 30 mm), gain guiding effects due to resonant mode coupling become important and change the results substantially.

For more explanations, see our web page for a case study:
http://www.rp-photonics.com/fiberpower_qs_yag_bp.html.

8.32 Passively Q-switched Nd:YAG Laser

File: **Nd-YAG laser, passively Q-switched.fpw**

This model is similar to the previous one, but instead of an active Q switch, a Cr:YAG crystal is used for passive Q switching. As a dynamic model can contain only a single “fiber”, this has been treated like a single fiber containing both Nd^{3+} and Cr^{4+} ions, spread over the whole resonator length. This means that we don’t use the usual simple energy level scheme, but a user-defined scheme with two ions, each having two relevant electronic levels. (Short-lived higher levels can be ignored, as their populations are negligible.)

The following diagrams are made:

- Diagram 1 simulates only the pump phase, using relatively course temporal steps. The temporal propagation is done until the net gain per resonator round trip becomes positive.
- Diagram 2 just shows the transverse profile of the Nd^{3+} excitation just before pulse emission.

- Diagram 3 then simulates the pulse generation. Here, a full dynamic simulation taking into account propagation times is needed. This is in principle slower, but it has to span only a smaller range of time. Therefore, we also get that done within a few seconds. The evolution of output power is shown on a logarithmic scale.
- Diagram 4 shows the same with a linear scale for the power and a smaller temporal range.
- Diagram 5 shows the transverse profiles of the Nd^{3+} and Cr^{4+} excitations after the pulse.
- Diagram 6 shows the pulse repetition rate, energy and duration as functions of the pump power. One sees that a higher pump power essentially only increases the pulse repetition rate but does not change the pulse parameter substantially.

8.33 Higher-order Soliton Pulses

File: **Higher-order solitons.fpw**

(The form settings file **Higher-order solitons.fpi** allows one to do similar things in the forms mode.)

Here, it is shown how higher-soliton propagation in a germanosilicate fiber is simulated:

- The mode properties are calculated from the given concentration profile of germania. The parameters were chosen such that single-mode guidance is obtained.
- The full chromatic dispersion profile, or alternatively only the second-order dispersion, can be used for the ultrashort pulse simulation.
- The initial pulse is taken to be an unchirped sech^2-shaped pulse, but with an energy corresponding to a higher-order soliton with a given order (e.g., 2 or 4).

It turns out that for short soliton pulse durations (1 ps or shorter), the higher-order dispersion substantially affects the pulse propagation, such that the pulses are no more exactly reproduced after one soliton period.

The following diagrams can be shown:

- Diagram 1: temporal evolution: optical power vs. time for different propagation length.
- Diagram 2: same information displayed with a color diagram
- Diagram 3: spectral evolution: power spectral density vs. wavelength for different propagation length.
- Diagram 4: spectral evolution displayed with a color diagram
- Diagram 5: chromatic dispersion profile of the fiber, showing that there is substantially third-order dispersion
- Diagram 6: producing spectrograms, which are stored in an animated GIF file

8.34 Soliton Self-Frequency Shift

File: **Soliton self-frequency shift.fpw**

(The form settings file **Soliton self-frequency shift.fpi** allows one to do similar things in the forms mode.)

Here, it is shown how the center wavelength of a fundamental soliton pulse in a fiber drifts to larger values due to stimulated Raman scattering.

The initial pulse is taken to be an unchirped sech^2-shaped pulse. For the Raman scattering in the fiber, a simplified response function is used.

The following diagrams can be shown:

- Diagram 1: evolution of frequency shift.
- Diagram 2: frequency shift vs. pulse duration. The results are compared with analytical results for a variable soliton pulse width.

8.35 Parabolic Pulses in an Ytterbium-doped Fiber Amplifier

File: **Parabolic pulses in Yb amplifier.fpw**

(The form settings file **Parabolic pulses in Yb amplifier.fpi** allows one to do similar things in the forms mode.)

Here, it is shown how ultrashort pulses are amplified in an ytterbium-doped fiber amplifier, where the fiber's chromatic dispersion is in the normal dispersion regime:

The initial pulse is taken to be an unchirped Gaussian pulse. During propagation in the fiber, the pulses develop an up-chirp and an increasing bandwidth, while the pulse duration also increases. The temporal pulse becomes roughly parabolic; this is the origin of the term *parabolic pulses*. The pulses approximately involve into *similaritons* (pulses which keep their temporal and spectral shape apart from an increasing width), but there are deviations due to the limited gain bandwidth and the non-constant amplifier gain (partly due to gain saturation).

Stimulated Raman scattering, which results from the delayed nonlinear response, is taken into account, but has no profound effect for the used parameters.

The following diagrams can be shown:

- Diagram 1: final pulse in the time domain
- Diagram 2: final pulse in the frequency domain
- Diagram 3: pulse energy, bandwidth and duration vs. position
- Diagram 4: This shows how the peak power and duration (FWHM) of the final pulse are affected by dispersive compression, depending on the amount of second-order dispersion. It is also displayed (in text form) what results would be achieved with a compressor which can compensate all dispersion up to the fourth order.

8.36 Stimulated Raman Scattering in Yb-doped Fiber Amplifier

File: **Stimulated Raman scattering in Yb amplifier.fpw**

This simulates how an ultrashort pulse is amplified in an Yb-doped amplifier. As the fiber nonlinearity is quite strong, stimulated Raman scattering becomes relevant: at the end of the fiber, a significant portion of the pulse energy is Raman-shifted to lower frequencies (longer wavelengths).

The following diagrams can be shown:

- Diagram 1: evolution of pump power
- Diagram 2: final pulse in the time domain

- Diagram 3: final pulse in the frequency domain
- Diagram 4: spectrogram of the final pulse
- Diagram 5: evolution of spectrum in the fiber
- Diagram 6: pulse parameters vs. position

8.37 Nonlinear Loop Mirror

File: **Nonlinear fiber loop mirror.fpw**

Here we investigate the function of a nonlinear loop mirror, consisting of a short active fiber, a longer passive fiber, and a fiber coupling connecting them to a ring. We inject pulses into an input port of the fiber coupler, obtaining two counterpropagating pulses in the ring. As one of these pulses is amplified *before* going through the passive fiber, while the other one is amplified *after* that, their nonlinear phase shifts are different. Therefore, when they meet again at the fiber coupler, the interference conditions are affected such that *not* all power goes back to the input port: a power-dependent fraction goes to the other port.

The ultrashort pulse propagation needs to be considered under the influences of Kerr nonlinearity, chromatic dispersion and amplification.

The following diagrams are made:

- Diagram 1: average powers along the active fiber. Here one see that some significant part of the pump power remains unabsorbed.
- Diagram 2: output pulses in the time domain. One can set different input pulses energies, e.g., and study how the coupling works.
- Diagram 3: coupling ratio vs. input pulse energy. Precisely speaking, the fraction of the total power getting to the output port is plotted as a function of input pulse energy. For every pulse energy, the steady state of the amplifier is calculated.

8.38 Regenerative Amplifier

File: **Regenerative amplifier.fpw**

This simulates the performance of a regenerative amplifier. Here, a pulse circulates many times in a resonator containing an amplifier until it is ejected when the pulse energy has reached a high value.

Such devices are very limited in performance when made with a fiber, because the nonlinear effects are very strong. Therefore, it is assumed that a bulk piece of doped glass is used, where the length is only 3 mm and the signal beam radius of 150 µm is relatively large.

The following diagrams can be shown:

- Diagram 1: evolution of pulse parameters in 60 subsequent resonator round-trips. The pulse energy rises roughly exponentially. When the pulse energy becomes substantial, the pulse bandwidth is increased substantially by self-phase modulation, resulting from the Kerr nonlinearity of the gain medium. Also, the pulse duration rises due to chromatic dispersion – with an increased speed once the spectral bandwidth has increased.
- Diagram 2 shows the final pulse (after 60 round trips) in the time domain. The roughly parabolic pulse profile results from the interaction of chromatic dispersion and the Kerr nonlinearity.

- Diagram 3 shows the optical spectrum of the final pulse. It is strongly broadened and structured as a result of the Kerr nonlinearity.
- Diagram 4 shows how the final pulse would react to variable amount of second-order dispersion. Optimum compression would be achieved for roughly −48 000 fs^2, where the compressed pulse duration is about 85 fs – much shorter than the initial pulse duration of 400 fs.
- Diagram 5 shows how various pulse parameters evolve in repetitive operation, i.e., when pulse amplification is done at a constant repetition rate of 4 kHz and the gain medium is pumped continuously. One finds that the pulse energy drops somewhat during the first pulses, as the time between two pulse amplification cycles is too short to fully replenish the energy stored in the gain medium.
- In diagram 6, we vary the number of resonator round-trips per amplification cycle. The repetition rate is kept fixed at 4 kHz. If the number of round-trips is low, e.g. 50, the pulse energy is low, because there is simply not enough gain to fully amplify the pulses. Above 70 round-trips, we get into a regime where the pulse energy oscillates between two values. For larger number of round-trips, there can be 4 or even more different pulse energies; one observes period doubling, and for certain parameters one could also get into a chaotic regime. This is an example for a deterministic chaos. That behavior depends on many parameters, such as the repetition rate, the number of round-trips for amplification, the initial pulse energy, the pump power, etc.

8.39 Mode-locked Fiber Lasers

Files: **Mode-locked fiber laser, ring resonator.fpw** and **Mode-locked fiber laser, linear resonator.fpw**

Here, we consider a mode-locked fiber laser with all normal dispersion. Comparing with a typical mode-locked bulk laser or a soliton fiber laser, the details of pulse formation in such a laser are rather sophisticated, because there is a complicated interplay of nonlinear effects, spectral filtering and a saturable absorber.

For the version with a ring resonator, in each resonator round trip the circulating pulse is transmitted through the following optical components:

- an output coupler with wavelength-independent transmission (for simplicity)
- a spectral bandpass filter
- a passive fiber
- an ytterbium-doped fiber
- another passive fiber
- a fast saturable absorber

The following diagrams can be shown:

- Diagram 1: time-dependent optical power (evolving over 100 resonator round trips) and the instantaneous frequency, showing a pronounced up-chirp
- Diagram 2: power spectral density and spectral phase with a frequency axis
- Diagram 3: power spectral density and spectral phase with a wavelength axis

- Diagram 4: evolution of intracavity pulse parameters during 100 round trips
- Diagram 5: pulse parameters in the fibers
- Diagram 6: power of the output pulse after dispersive compression with an automatically optimized second-order pulse compressor

The other file presents an example case with a linear resonator and is otherwise similar.

8.40 Chirped-pulse Amplifier System

File: **CPA system.fpw**

The system considered here consists of the following components:

- Ultrashort pulses at a high pulse repetition rate (41 MHz) are generated in a mode-locked fiber laser as discussed in the previous section. The output pulses have an energy of 1 nJ, a duration of 5.7 ps and an already broadened optical spectrum of ≈8 nm width (measured at the 10% level).
- A pulse picker transmits only one of 1000 pulses of the laser. The reduced average power allows for stronger amplification of the pulse energy later on.
- A 100 m long passive fiber acts as a dispersive pulse stretcher, increasing the pulse duration to 46 ps. The fiber nonlinearity is not negligible; it further broadens the spectrum to a width of 11.5 nm.
- An amplifier fiber with larger mode area amplifies the pulse energy from ≈1 nJ to 1.55 µJ. The spectrum is further broadened to 11.6 nm. In order to find the amplifier gain in the steady state, multiple pulse amplification cycles are simulated. (That iteration would actually be important only for lower repetition rates and correspondingly larger pulse energies.)
- A dispersive compressor with numerically optimized second- and third-order dispersion temporally compresses the pulses to 285 fs. The peak power rises up to 4.8 MW.

This simulation demonstrates that it is easy with **RP Fiber Power** to setup simulations even for rather complex amplifier systems.

8.41 Soliton Self-frequency Shift

File: **Soliton self-frequency shift.fpw**

We investigate how the mean wavelength of a soliton pulse slides towards larger values due to intra-pulse stimulated Raman scattering.

- Diagram 1 shows the evolution of the frequency shift in the fiber (usually a linear decrease).
- Diagram 2 shows the frequency shift as a function of pulse duration. (Shorter pulses exhibit much stronger frequency shifts.)

8.42 Supercontinuum Generation

File: **Supercontinuum generation.fpw**

Here, we simulate the complicated process of supercontinuum (strong spectral broadening) with ultrashort pulses in a fiber. The chromatic dispersion, which is an essential detail for supercontinuum generation, is taken from the mode solver. Nonlinearities resulting from the Kerr effect (with self-steepening) and stimulated Raman scattering are considered.

- Diagram 1 shows the calculated mode function. (The fiber is single-mode at the pump wavelength.)
- Diagram 2 shows the calculated chromatic dispersion versus wavelength.
- Diagram 3 shows the output pulse in the time domain.
- Diagram 4 shows the output pulse in the wavelength domain.
- Diagram 5 shows the spectral evolution in the fiber, i.e., the color-coded spectra versus position in the fiber.
- Diagram 6 shows the output light in the form of a spectrogram.

8.43 Transmission of Telecom Signal

File: **Data transmission.fpw**

We consider the transmission of a telecom signal through a germanosilicate fiber. The mode parameters of the fiber are calculated with the mode solver. A pseudorandom NRZ (not return to zero) signal is created and propagated through 5 km of the fiber. From this, an eye diagram is drawn.

The following diagrams can be shown:

- Diagram 1 shows the transmitted signal, and for comparison the input signal.
- Diagram 2 shows an eye diagram, demonstrating that for the chosen parameters the signal could still be recovered with a detector.

8.44 Fiber Amplifier with ASE, Output to html File

File: **Yb amplifier with ASE (with html output).fpw**

The simulation in this file is the same as that of another demo file (section 8.17). Here, we create an additional output file in html format (**Yb_amplifier_with_ASE.html**), which can be viewed with a web browser. If the html file and some corresponding files (see below) are uploaded to a web server, it can be displayed as a web page.

The include file "**html output.inc**" is used, which contains a few functions for writing html output. Also, a CSS file "**reports.css**" is delivered, which contains formatting details. The displayed images are stored in **.png** files when the script is executed with F8, and the html file refers to them.