Solver Guide for the MATLAB solid-core-fiber pulse propagation

Yi-Hao Chen
Applied and Engineering Physics, Cornell University

January 1, 2024


CORNELL UNIVERSITY
FOUNDED A.D. 1865

Contents


CORNELL UNIVERSITY
FOUNDED A.D. 1865

Chapter 1

Overview

1.1 Mathematical background

This package aims to solve the multimode unidirectional pulse propagation equation (MM-UPPE) in a solid-core fiber:

$$\begin{aligned}\partial_z A_p(z,\Omega) = {} & i\left[\beta_p(\omega) - \left(\beta_{(0)} + \beta_{(1)}\Omega\right)\right] A_p(z,\Omega) + g_p(z,\Omega)A_p(z,\Omega) \\ & + i\sum_{\ell} Q_{p\ell} A_\ell(z,\Omega) \\ & + \frac{i\omega}{4}\epsilon_0^2 n_{\text{eff}}^2 c n_2 \sum_{\ell m n}\Bigg\{ (1-f_R)\, Q^K_{p\ell mn}\mathfrak{F}[A_\ell A_m A_n^*] \\ & \qquad + f_R\Bigg\{ f_a Q^{R_a}_{p\ell mn}\mathfrak{F}\left[A_\ell\left[h_a * (A_m A_n^*)\right]\right] + \\ & \qquad\quad f_b Q^{R_b}_{p\ell mn}\mathfrak{F}\left[A_\ell\left[h_b * (A_m A_n^*)\right]\right]\Bigg\}\Bigg\}, \end{aligned} \tag{1.1}$$

where z is the propagation distance, $\Omega = \omega - \omega_0$, $\beta_{(1)}$ is the inverse group velocity of the moving frame, $g_p(z,\Omega)$ is the gain (or loss), n_2 is the nonlinear refractive index (m^2/W; refractive index change from nonlinearity $\triangle n = n_2 I$ where I is the light intensity), c is the speed of light; f_R is the Raman fraction representing the contribution of the Raman response of all nonlinearities where f_a and f_b are Raman fractions of the total Raman response for isotropic and anisotropic Raman responses, respectively ($f_a + f_b = 1$); h'_a and h'_b are isotropic and anisotropic Raman response functions; p, ℓ, m, and n the eigenmode indices. Q^K, Q^{R_a}, and Q^{R_b} are overlap integrals:

$$Q^K_{p\ell mn} = \frac{2}{3}Q^{R_a}_{p\ell mn} + \frac{1}{3}Q^k_{p\ell mn} \qquad , Q^k_{p\ell mn} = \frac{\int\left(\vec{F}_p^* \cdot \vec{F}_n^*\right)\left(\vec{F}_\ell \cdot \vec{F}_m\right)\mathrm{d}x\,\mathrm{d}y}{N_p N_\ell N_m N_n} \tag{1.2a}$$

$$Q^{R_a}_{p\ell mn} = \frac{\int\left(\vec{F}_p^* \cdot \vec{F}_\ell\right)\left(\vec{F}_m \cdot \vec{F}_n^*\right)\mathrm{d}x\,\mathrm{d}y}{N_p N_\ell N_m N_n} \tag{1.2b}$$

$$Q^{R_b}_{p\ell mn} = \frac{1}{2}\left[\frac{\int\left(\vec{F}_p^* \cdot \vec{F}_m\right)\left(\vec{F}_\ell \cdot \vec{F}_n^*\right)\mathrm{d}x\,\mathrm{d}y}{N_p N_\ell N_m N_n} + Q^k_{p\ell mn}\right] = \frac{1}{2}\left(Q^{r_b}_{p\ell mn} + Q^k_{p\ell mn}\right), \tag{1.2c}$$

where $\vec{F}_p$ is the p-th spatial eigenmode, n_{eff} and $n_{i,\text{eff}}$ in each N_i ($i \in \{p, \ell, m, n\}$) is often taken as the refractive index of silica such that

$$\frac{\epsilon_0^2 n_{\text{eff}}^2 c^2}{N_p N_\ell N_m N_n} = 4. \tag{1.3}$$

With Eq. (1.3), Eq. (1.1) is simplified to

$$\begin{aligned}\partial_z A_p(z,\Omega) = {} & i\left[\beta_p(\omega) - \left(\beta_{(0)} + \beta_{(1)}\Omega\right)\right] A_p(z,\Omega) + g_p(z,\Omega) A_p(z,\Omega) \\ & + i\sum_\ell Q_{p\ell} A_\ell(z,\Omega) \\ & + \frac{i\omega n_2}{c}\sum_{\ell mn}\Bigg\{ (1-f_R)\, S^K_{p\ell mn}\mathfrak{F}[A_\ell A_m A_n^*] + \\ & \qquad f_R\Bigg\{ f_a S^{R_a}_{p\ell mn}\mathfrak{F}\left[A_\ell\left[h_a * (A_m A_n^*)\right]\right] + \\ & \qquad\quad f_b S^{R_b}_{p\ell mn}\mathfrak{F}\left[A_\ell\left[h_b * (A_m A_n^*)\right]\right]\Bigg\}\Bigg\},\end{aligned} \tag{1.4}$$

with modified overlap integrals:

$$S^K_{p\ell mn} = \frac{2}{3}S^{R_a}_{p\ell mn} + \frac{1}{3}S^k_{p\ell mn} \qquad , S^k_{p\ell mn} = \int \left(\vec{F}_p^* \cdot \vec{F}_n^*\right)\left(\vec{F}_\ell \cdot \vec{F}_m\right) \mathrm{d}x\,\mathrm{d}y \tag{1.5a}$$

$$S^{R_a}_{p\ell mn} = \int \left(\vec{F}_p^* \cdot \vec{F}_\ell\right)\left(\vec{F}_m \cdot \vec{F}_n^*\right) \mathrm{d}x\,\mathrm{d}y \tag{1.5b}$$

$$S^{R_b}_{p\ell mn} = \frac{1}{2}\left[\int \left(\vec{F}_p^* \cdot \vec{F}_m\right)\left(\vec{F}_\ell \cdot \vec{F}_n^*\right) \mathrm{d}x\,\mathrm{d}y + S^k_{p\ell mn}\right] = \frac{1}{2}\left(S^{r_b}_{p\ell mn} + S^k_{p\ell mn}\right). \tag{1.5c}$$

However, in silica, it is sometimes overkill to run with UPPE due to mostly narrowband scenarios. In this case, $\beta(\omega)$ is obtained from its Taylor-series coefficients $\beta_{(0)} + \beta_{(1)}\Omega + \frac{\beta_2}{2}\Omega^2 + \frac{\beta_3}{3!}\Omega^3 + \cdots$, which is, in fact, equivalent to a more-commonly-used GMMNLSE.

1.2 High-level understanding of this package

This package is designed for both single-spatial(transverse) mode or multi-spatial modes. Not only scalar but also polarized fields can be simulated, as well as Raman scattering and the gain. The package exhibits an adaptive control of the step size, except for situations with amplified stimulated emission (ASE). In addition to CPU, highly parallelized cuda computation with a Nvidia GPU is implemented, which is strongly recommended for running with multimodes. In single-mode simulations, for sampling numbers less than approximately 2^{25}, they can still run faster with CPU than with GPU.

The fastest way to learn how to use this code is to start with the example codes in the package.

Chapter 2

Before I go deeply into details

2.1 Introduction

This document describes how to use the GMMNLSE_propagate() MATLAB function.

Below is how to call this function in general.

```
prop_output = GMMNLSE_propagate(fiber, ...
                                initial_condition, ...
                                sim[, ...
                                gain_rate_eqn])
```

prop_output

It contains the information of the output field after propagating through the fiber, such as the field amplitudes and the positions of each saved field, etc.

fiber

It contains the information of the fiber, such as β_2, S^R, and the MFD, etc.

initial_condition

It contains the information of the input.
Typically it's the input field amplitude. If you run it not only with the rate-equation gain model but considering ASE, it also contains the forward ASE at the input and backward ASE at the output.

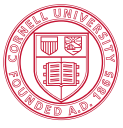
Figure 2.1: Initial conditions

sim

It contains a multitude of information about the simulation, such as the algorithm to use, if running with an adaptive-step method, and the center wavelength, etc.

[**gain_rate_eqn**]

This is required only for the rate-equation gain model.
It contains the information required for the rate-equation gain model, such as the pump power, the doped ion density, etc.



Chapter 3

Input arguments

Below, I use N_t as the number of time/frequency sampling points, N_m as the number of modes, and N_{sm} as the number of spatial modes. If there is no polarized mode, $N_m = N_{sm}$; otherwise, $N_m = 2N_{sm}$.

I recommend to use the information below as a reference guide if you're confused. Start with an example script is always better than reading this first. I have provided some example scripts in Chap.**??**.

Some parameters are required only when you enable some settings. Below I labelled in blue the parameters required all the time.

3.1 fiber

betas

Typically, this is the variable that saves the $\beta_0, \beta_1, \beta_2 \ldots$. It's has the unit of ps^n/m.

It's a column vector of

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \end{bmatrix} \tag{3.1}$$

if this is a single-mode simulation.

For multimode, it becomes

$$\begin{bmatrix} \beta_{0|1} & \beta_{0|2} & \cdots \\ \beta_{1|1} & \beta_{1|2} & \cdots \\ \beta_{2|1} & \beta_{2|2} & \cdots \\ \beta_{3|1} & \beta_{3|2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, \tag{3.2}$$

where $\beta_{i|m}$ is the β_i for the m-th mode.

Besides the narrowband Taylor series expansion of $\beta(\omega)$, I've also implemented the broadband version whose **betas** are column vectors of $\beta(\omega)$, that is, it becomes

$$\begin{bmatrix} \beta_{\cdot|1} & \beta_{\cdot|2} & \cdots \end{bmatrix}, \tag{3.3}$$

where each $\beta_{\cdot|m}$ is a column vector of the propagation constant of the m-th mode. It's a function of frequency, with an order from small to large. If the simulation is run with N_t time/frequency points, this $\beta_{\cdot|m}$ should have the length of N_t as well.

n2

It's the nonlinear coefficient of the fiber. By default, GMMNLSE_propagate() uses $2.3 \times 10^{-20}\,\mathrm{m}^2/\mathrm{W}$ assuming we use a silica fiber around 1 µm if fiber.n2 is left empty.

SR

It's the overlap integral S^R in scalar GMMNLSE. It's loaded in a dimension of N_{sm}^4.

For scalar GMMNLSE, S^K is S^R if the field is linearly polarized or $\frac{2}{3}S^R$ is it's circularly polarized. The unit is m^{-2}.

For polarized fields, their S^R and S^K can be calculated from the scalar S^R. This will be done by GMMNLSE_propagate() automatically if "sim.scalar=false." For details, check Chap.5.

L0

This is the fiber length. The unit is meter.

fiber_type

This specifies the material of the fiber. It's 'silica' by default. It's used only for specifying which Raman model to use. It's either 'silica', 'chalcogenide', or 'ZBLAN'.

If we use the Gaussian-gain model, the following parameters are required.

dB_gain

The small-signal gain amplification of the pulse energy in dB. This is used to calculate the gain_coeff (default to 30).

gain_coeff

This is the small-signal gain coefficient, the g in

$$A(z) = \mathrm{e}^{gz/2}A(0). \tag{3.4}$$

It's a scalar with the unit of m^{-1}.

gain_fwhm

This is the gain bandwidth. A typical number for Yb-doped gain fibers is 40 nm. This parameter has the unit of meter.

gain_doped_diameter

The diameter of the doped core to compute the overlap integral between mode fields and the doped core and accurately find their gain.

saturation_intensity

This is for multimode simulations. It has the unit of J/m^2.

saturation_energy

This is for single-mode simulations. It has the unit of nJ.

3.2 initial_condition

dt

This is the time sampling step $\triangle t$ with a unit of ps.

fields

This is the input field amplitude under the time domain. It has the unit of $\sqrt{\mathrm{W}}$. Its size is $N_t \times N_m$.

If its size is $N_t \times N_m \times N_z$, only the last N_z is taken as the input field.

The two parameters above are required all the time.

If we run the simulation with ASE (and also of course with the rate-equantion gain model), two extra parameters are required. Typically they are both all-zero $N_t \times N_m$ column vectors.

Power.ASE.forward

This is the forward ASE at the input ($z = 0$). Its array size is the same "fields" above.

Power.ASE.backward

This is the backward ASE at the output ($z = L_0$) because backward ASE starts from the output end of the fiber.

3.3 sim

Below are the most basic parameters for a simualtion.

betas

In UPPE, we not only create a moving frame that follows the pulse with the inverse velocity $\beta_{(1)}$ but extract out the reference propagation constant $\beta_{(0)}$. The benefit of extracting $\beta_{(0)}$ is that it reduces the rate of global phase increment such that the simulation can run with a larger step. This is similar to the limitation of multimode simulations that different spatial modes have different propagation constants that generate beating. To resolve the multimode beating, the size of the z-step cannot be too large.

This "betas" is a 2×1 column vector.

$$\begin{bmatrix} \beta_{(0)} \\ \beta_{(1)} \end{bmatrix} \tag{3.5}$$

By default, under the narrowband case where fiber.betas is a column vector of the Taylor series coefficient of $\beta(\omega)$, GMMNLSE_propagate() uses the 1st mode as the reference, that is,

$$\begin{bmatrix} \beta_{(0)} \\ \beta_{(1)} \end{bmatrix} = \begin{bmatrix} \beta_{0\vert 1} \\ \beta_{1\vert 1} \end{bmatrix}. \tag{3.6}$$

f0

The center frequency (THz). It's a scalar.

deltaZ

The z-step size (m). This is required only for non-adaptive-step method. For an adaptive-step method, this parameter varies during computation; setting it here is meaningless.

This may need to be 1–50 µm to account for intermodal beating, even if the nonlinear length is large.

save_period

The length between saved fields (m). If it's zero, it's equaivalent to save_period=fiber.L0 that saves only the input and output fields.

If the simulation doesn't use an adaptive-step method, be aware that this number needs to be a divisor of the fiber length, fiber.L0; otherwise, GMMNLSE_propagate() will throw an error. For an adaptive-step method, I have the maximum step size set as the $\frac{1}{10}$ of the save_period and the position of the saved fields will be chosen as the one that first passes through each saved point.

3.3.1 MPA

Here are the parameters if the simulation uses MPA step method. All parameters are contained within a "sim.MPA" structure.

MPA.M

This is the parallel extent for MPA. 1 is no parallelization. 5–20 is recommended; there are strongly diminishing returns after 5–10. 10 is recommended.

MPA.n_tot_max

The maximum number of iterations for MPA. This doesn't really matter because if the step size is too large, the algorithm will diverge after a few iterations. 20 is a typical number for this.

MPA.n_tot_min

The minimum number of iterations for MPA. 2 is recommended.

MPA.tol

The tolerance of convergence for MPA, which is related to the values of the average NRMSE between consecutive itertaions in MPA at which the step is considered converged. 10^{-6} is recommended.

3.3.2 Random linear mode coupling

Here are the parameters if the simulation uses random mode coupling. All parameters are contained within a "sim.rmc" structure. To run with random mode coupling, random-coupling matrices need to be created beforehand by calling

```
save_points=int32(fiber.L0/sim.deltaZ);
sim.rmc.matrices = create_rmc_matrices(fiber,sim,num_modes,save_points);
```

"sim.deltaZ" is required since there is no adaptive step-size control for computations with random mode coupling.

rmc.model

false (0)	includes random mode coupling
true (1)	don't include random mode coupling

rmc.varn

The variations of refractive index of the fiber. It is used to control the strength of random mode coupling.

rmc.stdQ_polarizedmode

Similar to "rmc.varn", it is used control the strength of random polarization-mode coupling.

rmc.lambda0

It lets the code know the wavelength of the eigenmode fields to load for random mode coupling to compute the coupling strengths.

rmc.downsampling_factor

To compute the couplng strengths among spatial modes, loading mode profiles is required. This downsampling factor determines the downsampled factor after loading to improve the performance of the random-mode-coupling matrices. Since it won't affect the latter nonlinear pulse propagation, I typically just set it to 1.

3.3.3 Polarization modes

Here are the parameters if the simulation includes polarization modes.

scalar

false (0)	includes polarization-mode coupling
true (1)	don't include polarization-mode coupling

If the simulation is solved with "sim.scalar=true," the input field takes only the scalar fields, e.g.,

$$\begin{bmatrix}\text{mode 1} & \text{mode 2} & \text{mode 3} & \dots\end{bmatrix}.$$

Otherwise, the input field of each polarized mode needs to be specified in the order of

$$\begin{bmatrix}\text{mode } 1_+ & \text{mode } 1_- & \text{mode } 2_+ & \text{mode } 2_- & \dots\end{bmatrix},$$

where (+,-) can be (x,y), (right-handed circular, left-handed circular), or any orthogonally polarized modes.

Based on whether to include polarization-mode coupling, S^R and S^K are automatically calculated to its polarized version by GMMNLSE_propagate().

ellipticity

The ellipticity of the polarization modes. Please refer to "Nonlinear Fiber Optics, eq (6.1.18) Agrawal" for the equations.

0	linear polarization	(+,-)=(x,y)
1	circular polarization	(+,-)=(right,left)

3.3.4 Adaptive-step method

Here are the parameters if the simulation uses adaptive-step method. All parameters are contained within a "sim.adaptive_deltaZ" structure. The user doesn't need to specify whether to use adaptive-step method or not; the code determines itself. With the adaptive-step method, the initial step size is set to a small 10^{-6} m.

adaptive_deltaZ.threshold
The threshold of the adaptive-step method. It controls the accuracy of the simulation and determines whether to increase or decrease the step size. I typically use 10^{-6}.

adaptive_deltaZ.max_deltaZ
The maximum z-step size (m) of the adaptive-step method. It's 1/10 the save_period by default.

3.3.5 Algorithm to use

gpu_yes

true (1)	use GPU
false (0)	don't use GPU

Raman_model

0 ignore Raman effect
1 Raman model approximated analytically by a single vibrational frequency of molecules (Ch. 2.3, p.42, Nonlinear Fiber Optics (5th), Agrawal)
2 Raman model including the anisotropic contribution ("Ch. 2.3, p.43" and "Ch. 8.5, p.340," Nonlinear Fiber Optics (5th), Agrawal)

For more details about anisotropic Raman, please read "Raman response function for silica fibers," by Q. Lin and Govind P. Agrawal (2006). Besides silica, chalcogenide and ZBLAN are also included.

gain_model
Except for the rate-equation gain model, all the other gain models use a Gaussian gain; thus, the gain_coeff, gain_fwhm, and gain saturation intensity or energy need to be specified in "fiber."

0 no gain
1 Gaussian gain
2 rate-equation gain: see Chap.6 for details

pulse_centering
Because the pulse will evolve in the fiber, it's hard to have the moving frame always move with the same speed as the pulse. As a result, the pulse will go out of the time window and come back from the other side due to the use of periodic assumption of discrete Fourier Transform. The shift in time is saved in "prop_output.t_delay" so that you don't lose the information

When enabling pulse_centering, the pulse will be centered to the center of the time window based on the moment of the field intensity ($|A|^2$).

true (1) center the pulse according to the time window
false (0) don't center the pulse

include_sponRS

true (1) include spontaneous Raman term
false (0) don't include spontaneous Raman term

include_noise

For a Raman or four-wave-mixing process, it's important to include the shot noise to generate a reasonably strong signal. The strength of noise is controlled by "num_photon_noise_per_band".

true (1) include photon noise
false (0) don't include the photon noise

num_photon_noise_per_bin

This controls the number of photon noise to include per spectral discretization bin. It's default to 1.

$$P_{\text{noise}} = hf/\left(N\triangle t\right), \tag{3.7}$$

whose relation depends on the convention of Fourier Transform. For convention of the laser field, Fourier Transform is defined as MATLAB's "ifft," which leads to the relation here. With a different convention, the multiplication factor might be different from $1/\left(N\triangle t\right)$.

cuda_dir_path

The path to the cuda directory into which ptx files will be compiled and stored. This is "/GMMNLSE/cuda/."

gpuDevice.Index

The GPU to use. It's typically 1 if the computer has only one GPU. MATLAB starts the index with 1.

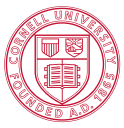
Here are the parameters for the progress bar used in the simulation. It's useful in general to see how a simulation progresses.

progress_bar

true (1) show progress bar
false (0) don't show progress bar

progress_bar_name

The name of the GMMNLSE shown on the progress bar. If not set (no "sim.progress_bar_name"), it uses a default empty string, ''.



Cornell University


CORNELL UNIVERSITY
FOUNDED A.D. 1865

Chapter 4

Output arguments

fields
The $N_t \times N_m \times N_z$ output fields.

dt
This is the time sampling step $\triangle t$ with a unit of ps.

z
This is the positions of each saved field.

deltaZ
The z-step size (m).

For an adaptive-step method, this contains the step size at each saved point. You can see how the step size evolves through the propagation with this parameter.

betas
The "sim.betas,"$[\beta_{(0)}; \beta_{(1)}]$, used in this propagation.

t_delay
The time delay of the pulse at each saved point due to pulse centering.

seconds
The time spent for this simulation.

4.1 For rate-equation gain model

4.1.1 Power

Here saves the pump and ASE power. They are saved in the "prop_output.Power" structure.

Power.pump.forward
The forward pump power along the fiber. Its size is $1 \times 1 \times N_z$.

Power.pump.backward
The backward pump power along the fiber. Its size is $1 \times 1 \times N_z$.

If ASE is considered,

Power.ASE.forward
The forward ASE power along the fiber. Its size is $N_t \times N_m \times N_z$ if run with multimode and $1 \times 1 \times N_z$ if run with single mode.

Power.ASE.backward
The backward ASE power along the fiber. Its size is $N_t \times N_m \times N_z$ if run with multimode and $1 \times 1 \times N_z$ if run with single mode.

4.1.2 Others

If the population inversion, N2, is exported,

N2
The doped ion density of the upper-state population. Its size is $N_x \times N_x \times N_z$ if run with multimode and $1 \times 1 \times N_z$ if run with single mode. N_x is the number of cross-sectional spatial sampling.

The following output occrus only when gain_rate_eqn.reuse_data=true.

saved_data
This is used for an oscillator to converge faster. There's no need for a user to read this. It'll be sent to GMMNSLE_propagate() in the next roundtrip.

Chapter 5

Polarization modes

If the simulation is solved with "sim.scalar=true," the input field takes only the scalar fields, e.g.,

$$\begin{bmatrix}\text{mode 1} & \text{mode 2} & \text{mode 3} & \dots\end{bmatrix}.$$

Otherwise, the input field of each polarized mode needs to be specified in the order of

$$\begin{bmatrix}\text{mode } 1_+ & \text{mode } 1_- & \text{mode } 2_+ & \text{mode } 2_- & \dots\end{bmatrix},$$

where (+,-) can be (x,y), (right-handed circular, left-handed circular), or any orthogonally polarized modes.

If the input β has a dimension of only the number of spatial modes, N_{sm}, I assume there's no significant influence from birefringence; thus, it's expanded into $2N_{sm}$ dimension with each i and j (polarization) modes being degenerate by GMMNLSE_propagate(). For polarized fields,

$$S^R_{plmn} = \frac{\int \mathrm{d}x\,\mathrm{d}y\left[\mathbf{F}^*_p\cdot\mathbf{F}_l\right]\left[\mathbf{F}^*_m\cdot\mathbf{F}_n\right]}{\left[\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_p|^2\right)\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_l|^2\right)\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_m|^2\right)\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_n|^2\right)\right]^{1/2}} \tag{5.1}$$

$$S^K_{plmn} = \frac{2}{3}S^R_{plmn} + \frac{1}{3}\frac{\int \mathrm{d}x\,\mathrm{d}y\left[\mathbf{F}^*_p\cdot\mathbf{F}^*_n\right]\left[\mathbf{F}_m\cdot\mathbf{F}_l\right]}{\left[\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_p|^2\right)\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_l|^2\right)\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_m|^2\right)\left(\int \mathrm{d}x\,\mathrm{d}y|\mathbf{F}_n|^2\right)\right]^{1/2}} \tag{5.2}$$

Therefore, S^R_{plmn} isn't zero as (p,l) and (m,n) both have the same polarization, and we get four possibilities for (p,l,m,n), (0,0,0,0), (0,0,1,1), (1,1,0,0), and (1,1,1,1), with their values directly derived from the scalar S^R_{plmm}. For S^K_{plmn}, in addition to the permutations of S^R_{plmn}, we need to consider those from the fraction above which isn't zero as (p,l,m,n) is (0,0,0,0), (0,1,1,0), (1,0,0,1), and (1,1,1,1). Notice that some of them can add up with S^R_{plmn} while some of them can't, so the value has a prefactor of 1, $\frac{2}{3}$, $\frac{1}{3}$.

The above generalization of the scalar S^R to polarized S^R, S^K needs each $\mathbf{F}_p$ to be either parallel or orthogonal to one another, so (i,j) has to be an orthogonal group in 2D, e.g., (x, y) or (σ_+, σ_-).

Chapter 6

Rate-equation gain model

To run with rate-equation gain model, you need to run "gain_info()" first. This precomputes the required information for this model and saves the computational time. For example, it computes the doped ion density based on the absorption and the cladding area. It also loads the multimode spatial profiles for multimode gain evolution.

Below is the code sequence of how to run the rate-equation gain model:

```
f = ifftshift( (-N/2:N/2-1)/N/dt + sim.f0 ); % in the order of "omegas" in
    GMMNLSE_propagate()
c = 299792.458; % nm/ps
lambda = c./f; % nm

% First call gain_info():
gain_rate_eqn = gain_info( fiber,sim,gain_rate_eqn,lambda );

% And then send it to GMMNLSE_propagate():
output_field = GMMNLSE_propagate( fiber,input_field,sim,gain_rate_eqn );
```

6.1 Input arguments

Most of the important parameters are contained in the gain_rate_eqn structure. Some parameters are used only with multimode simualtions. I labelled in blue those required all the time whether it's single-mode or multimode. I put (SM) if it's only for single-mode simulations and (MM) if it's only for multimode ones.

6.1.1 gain_rate_eqn

Multimode mode-profile folder

MM_folder$^{(MM)}$

A string; where the betas.mat and S_tensor_?modes.mat are. This is used only for multimode simulations which need to load their betas and SR values in the mat files from the mode solver.

Oscillator info

reuse_data

True (1) or false (0).

For a ring or linear cavity, the pulse will enter a steady state eventually. If reusing the pump and ASE data from the previous roundtrip, the convergence can be much faster, especially for counterpumping.

linear_oscillator

True (1) or false (0), about whether the simulation is for a linear oscillator.

For a linear oscillator, there are pulses from both directions simultaneously, which will both contribute to saturating the gain; therefore , the backward-propagating pulses need to be taken into account.

For a linear oscillator, gain_rate_eqn.reuse_data must be "true" to consider the backward-propagating pulses. If gain_rate_eqn.reuse_data is "false", gain_info() will correct it to "true".

How to use it:

```
% previous_rate_gain_saved_data comes from the GMMNLSE_propagate() output
    from the previous roundtrip
gain_rate_eqn.saved_data = previous_rate_gain_saved_data;
prop_output = GMMNLSE_propagate(fiber,...
                                input_field,...
                                sim,...
                                gain_rate_eqn);
(next)previous_rate_gain_saved_data = prop_output.saved_data;
```

Gain-fiber info

core_diameter

For double-clad fibers, this is where the doped ion is and the pulse propagates in (μm).

cladding_diameter

The cladding diameter (μm).

core_NA$^{(SM)}$

The core numerical aperture of the gain fiber. This is used only for single-mode simulations to calculate the MFD and further the overlap factor between the signal pulse and the doped ion. For multimode, the overlap factor is obtained from loaded spatial profiles; therefore, it doesn't need core_NA.

Doped-ion info

absorption_wavelength_to_get_N_total

The wavelength specified by the manufacturer which they use to measure the absorption of the gain fiber (nm).

absorption_to_get_N_total

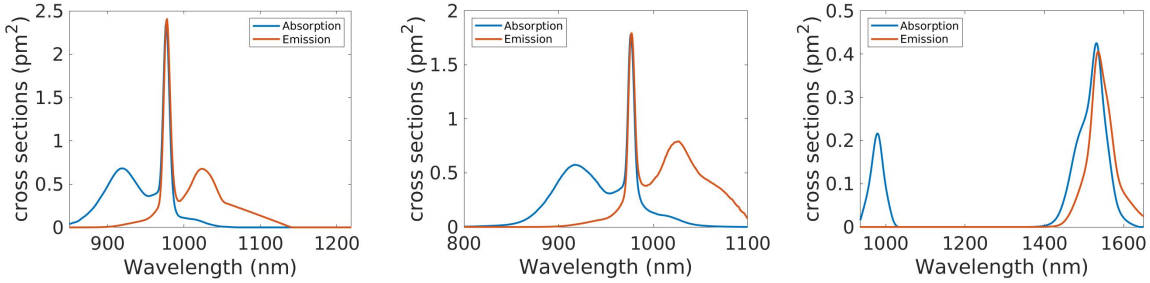
The absorption measured with the wavelength specified above (dB/m).

cross_section_filename

The filename of the data of the doped-ion cross section. It contains both the emisson and absorption cross sections.

Currently I have '**Liekki Yb_AV_20160530.txt**' and '**Yb_Gen_VIII_Cross_Section (Nufern).txt**' for Yb and '**optiwave Er.txt**' for Er.

Er data is from the optiwave website. Nufern (currently bought by Coherent) Yb data is from their spreadsheet.



(a) Yb (Liekki) (b) Yb (Nufern) (c) Er (Optiwave)

Figure 6.1: Absorption and emission cross sections.

Pump info

pump_wavelength

The pump wavelength (nm).

copump_power

This is the pump sending in from the input end of the fiber (W). It co-propagates with the signal pulse. If it's counterpumping, set this to zero.

counterpump_power

This is the pump sending in from the output end of the fiber (W). It counter-propagates with the signal pulse. If it's copumping, set this to zero.

Mode profiles

If "mode_profiles" is ignored in the input arguments of gain_info(), it will read the mode profiles from gain_rate_eqn.MM_folder. Some extra operations on these mode profiles are done with the following parameters.

downsampling_factor$^{(MM)}$

This is the factor of downsampling that reduces the size of mode profiles for multimode. Typically the loaded mode profiles from the mode solver is 800×800, so downsampling it down to 100×100, or even smaller, speeds up a simulation a lot. Under this circumstances, gain_rate_eqn.downsampling_factor=8.

Computational info

tau

The lifetime of the upper states, which is used in spontaneous emission (s).

1. lifetime of Yb in $F_{5/2}$ state = 840 µs (Paschotta et al., "Lifetme quenching in Yb-doped fibers")
2. lifetime of Er in $^4I_{13/2}$ state = 8-10 ms

t_rep

The roundtrip time (1/repetition rate) of the pulse, which is used to calculate the power of the "signal" pulse (s).

This rate-equation gain model assumes a high repetition rate such that the gain is able to reach a steady state. Therefore, the information of the repetition rate of the pulse is necessary. If the repetition is too low, gain_info() will throw a warning.

ASE info

ignore_ASE

True (1) or false (0).

sponASE_spatial_modes

The number of available spatial modes for ASE.

In principle, the number of spatial modes for ASE should be the same as those for signal fields. However, due to computational simplicity, a smaller number of spatial modes for signal fields might be considered. In such a situation, ASE spatial modes should still be considered to correctly approximate the amount of generated ASE. For example, in large-mode-area fibers, the number of ASE modes can be larger than that of the signal field, where usually, only the fundamental mode is considered for the signal field due to fiber coiling. If this value is left empty like "[ ]", it is "length(sim.midx)," the number of signal spatial modes. I use "sponASE" because ASE grows from spontaneous emission. The number of spatial modes manifests itself as more spontaneous emission generated.

Algorithm info

export_N2

True (1) or false (0). Whether to export N2, the ion density of the upper state, or not.

max_iterations

The maximum number of iterations.

If having iterations is required, you can see that more iterations are needed for a longer fiber to converge. Play around this value to get the result to converge.

tol

The tolerance of this iteration. If the difference of pulse energy or ASE power between the last two results is smaller than this tolerance, it's done.

verbose
Show the information (final pulse energy) during iterations.

memory_limit
The memory limit for the simulation. This can be found by default. It's important only when gain_rate_eqn.reuse_data=true for an oscillator.

With GPU computing, it is "sim.gpuDevice.Device.AvailableMemory/2". Otherwise, it looks for the available RAM for MATLAB and becomes "userview.MemUsedMATLAB/2 · 2^{20}" for windows. This is supported only under windows and linux, not iOS because I have no experience in iOS. The command between linux and iOS shouldn't differ too much, it's possible to implement it.

6.1.2 lambda and mode_profiles

lambda
The wavelengths of the simulation (nm). It's ordered as right after taking "ifft." Check the code of how to call gain_info() at the beginning of this chapter.

mode profiles

If this isn't specified or left empty, gain_info() will load mode profiles from gain_rate_eqn.MM_folder.

mode_profiles
The eigenmode profiles of field amplitudes. It'll be normalized into the unit of 1/µm in gain_info().

mode_profiles_x
The x-position of mode profiles (µm). It's an array of length N_x.

6.2 Output arguments

Precomputing overlap_factor, FmFnN, and GammaN is the main reason of running this function before GMMNLSE_propgate() with the rate-equation gain. For multimode, it saves a huge amount of time.

gain_rate_eqn.cross_sections_pump
The absorption and emission cross sections at the pump wavelength (µm^2).

gain_rate_eqn.cross_sections
The absorption and emission cross sections over the frequency domain (µm^2). This is used for signal pulse and ASE.

gain_rate_eqn.overlap_factor
The overlap factors of both the pump and the signal pulse. It contains overlap_factor.pump and overlap_factor.signal and determines the overlap between each mode and the doped ion. It has no unit for single-mode but has the unit of 1/µm^2 for multimode.

gain_rate_eqn.N_total

The doped ion density ($1/\mu m^3$). For single-mode, it's a scalar; while for multimode, its size is $N_x \times N_x$.

gain_rate_eqn.FmFnN

It precomputes $\int_{A_{\text{core}}} F_{m_i} F_{n_i}^* N_T \, \mathrm{d}^2x$, the integral2(overlap_factor*N_total), for the signal and ASE.

gain_rate_eqn.GammaN

It precomputes $\int_{A_{\text{core}}} \frac{N_T}{A_{\text{cladding}}} \, \mathrm{d}^2x$, the integral2(overlap_factor*N_total), for the pump.

Chapter 7

load_default_GMMNLSE_propagate()

Because of the overwhelming parameters of input arguments, I've created a function that loads the default value for each parameter. If a user has specified the value already, the user's value precedes over the default one.

Here is a typical way of calling this function.

```
[fiber,sim] = load_default_GMMNLSE_propagate(input_fiber,...
                                             input_sim[,type_of_mode])
```

input_fiber and input_sim are user-defined parameters. type_of_mode is either 'single-mode' or 'multimode'; if it's ignored, 'single-mode' is assumed by default. Below are some examples.

```
% User-defined parameters
fiber.betas = [0 0 0.02 0];
fiber.L0 = 3;

% Incorporate default settings
[fiber,sim] = load_default_GMMNLSE_propagate(fiber,[]); % single_mode

% If there are "sim" settings
sim.gpu_yes = false;
[fiber,sim] =  load_default_GMMNLSE_propagate(fiber,sim); % single_mode

% Use only user-defined "sim", not "fiber"
[fiber,sim] = load_default_GMMNLSE_propagate([],sim); % single_mode

% For multimode, you must add the string 'multimode' as the last argument.
[fiber,sim] = load_default_GMMNLSE_propagate(fiber,sim,'multimode');
```

Besides loading the default values, this function gives a user more options to obtain several parameters. This function transforms them into the allowed parameters of GMMNLSE_propagate(). I list them below. If both equavilence are specified unfortunately, the allowed GMMNLSE_propagate() input has the higher priority.

Description	Allowed GMMNLSE_propagate()'s input	Equivalent input arguments for this function
center frequency/wavelength	sim.f0 (THz)	sim.lambda0 (m)
nonlinear coefficient	fiber.SR (m^{-2})	fiber.MFD (μm)

Several other input arguments are

midx

An array of the mode indices. It helps select only those modes we want to use in the simulation. For example, if I want only mode 2 and mode 4 in simulations,

```
sim.midx = [2,4];
```

This function will read "betas" and "SR" with

```
betas = betas_mat_file(:,midx);
SR = SR_mat_file(midx,midx,midx,midx);
```

To load multimode mode profiles, use the following three parameters.

MM_folder

This specifies the folder where betas and SRSK mat files are stored; only used in multimode.

betas_filename

The filename of the mat file that stores betas.

Note that the input unit of betas in GMMNLSE_propagate() is ps^n/m while the one from the mode solver is fs^n/m. Besides loading the betas data, this function helps transform into the unit GMMNLSE_propagate() needs after loading. If the user provides their own betas, they need to make sure the unit is correct; this function assumes the user's input has the correct unit and won't modify it.

S_tensors_filename

The filename of the mat file that stores S^R tensors.

A few values about the gain are used only in this file to calculate the gain saturation intensity or energy. They are labelled with an asterisk $\star$. If you provide the saturation intensity or energy directly, you don't need to worry about these parameters.

Below is the process flow of this "load_default_GMMNLSE_propagate()" function. Read this if you're not sure whether your input will be used or overwritten. Because user-defined parameters take precedence, overwritten should happen only for (f0,lambda0) and (SR,MFD) mentioned above.

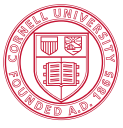
```
% <-- Uncorrelated parameters are loaded directly -->

sim.f0 - depend on input f0 or lambda0
         If no input f0 or lambda0, f0=3e5/1030e-9 (THz)

% If there's a user-defined one, use user's instead for the parameters below.
% Below I list the default values -->
fiber.fiber_type = 'silica';
fiber.n2 = 2.3e-20;

sim.deltaZ = 1000e-6;
sim.save_period = 0;
sim.ellipticity = 0; % linear polarization

sim.MPA.M = 10;
sim.MPA.n_tot_max = 20;
sim.MPA.n_tot_min = 2;
sim.MPA.tol = 1e-6;
```



```
19
20 sim.rmc.model = false;
21 sim.rmc.varn = 0;
22 sim.rmc.stdQ_polarizedmode = 0;
23 sim.rmc.lambda0 = default_sim.lambda0;
24 sim.rmc.downsampling_factor = 1;
25
26 sim.scalar = true;
27
28 sim.adaptive_deltaZ.threshold = (1e-6 if RK4IP or 1e-3 if MPA);
29
30 sim.single_yes = true;
31 sim.gpu_yes = true;
32 sim.Raman_model = 1;
33 sim.gain_model = 0;
34
35 sim.pulse_centering = true;
36 sim.include_sponRS = true;
37 sim.num_photon_noise_per_band = 0;
38 sim.include_noise = false;
39 sim.gpuDevice.Index = 1;
40 sim.progress_bar = true;
41 sim.progress_bar_name = '';
42 sim.verbose = false;
43 sim.cuda_dir_path = 'GMMNLSE/cuda';
44
45 % <-- Correlated parameters are loaded based on the input or default -->
46
47 % single-mode -- >
48
49 sim.midx = 1;
50
51 %   Assume 1030 nm for positive dispersion if lambda0 < 1300 nm (~ZDW for a
      silica fiber),
52 fiber.betas = [8.8268e6; 4.8821e3; 0.0209; 32.9e-6; -26.7e-9];
53 fiber.MFD = 5.95; % um; 1030nm from Thorlabs 1060XP
54 %   Assume 1550 nm for negative dispersion if lambda0 > 1300 nm (~ZDW for a
      silica fiber),
55 fiber.betas = [5.8339e6; 4.8775e3; -0.0123; 0.1049e-6; -378.3e-9];
56 fiber.MFD = 8.09; % um; 1030nm from Thorlabs 1060XP
57
58 (input SR precedes over input MFD)
59 fiber.SR = (1) input SR, if there's input SR
60           (2) 1/Aeff, if (a) there's input MFD
61                          (b) MFD is taken from the default one and there's no
                               input MFD
62
63 *fiber.gain_Aeff = 1/fiber.SR (taken from above)
64 *fiber.gain_doped_diameter = fiber.MFD;
65
66 % multimode -->
67
68 fiber.MFD = [] (not used)
69
70 sim.midx = (1) input midx
71           (2) 1:num_modes (num_modes is determined by loading "betas.mat")
```

```
72
73 fiber.betas = (1) input betas
74               (2) loaded from betars_filename in fiber.MM_folder (loaded modes
                     are based on the above midx)
75 fiber.SR - (1) input SR
76           (2) loaded from S_tensors_filename in fiber.MM_folder (loaded modes
                 are based on the above midx)
77 *fiber.gain_Aeff = (1) pi*( (input gain_doped_diameter) *1e-6/2)^2;
78                    (2) 1/fiber.SR(1,1,1,1) (taken from above)
79 *fiber.gain_doped_diameter = (1) input gain_doped_diameter
80                             (2) fiber.MFD;
81
82 % For both single-mode and multimode -->
83
84 fiber.L0 = (1) input L0
85            (2) 2 (m)
86
87 fiber.dB_gain = (1) input gain under dB/m
88                 (2) 30 (dB/m)
89 % overlap_factor is obtained from gain_doped_core. It's the overlap between the
       mode fields and the doped core.
90 fiber.gain_coeff = (1) (input gain_coeff)*overlap_factor
91                    (2) fiber.dB_gain*log(10)/(10*fiber.L0)*overlap_factor; % m
                           ^-1, from db/m
92 fiber.gain_fwhm = (1) input gain_fwhm
93                   (2) 40e-9; % m
94
95 *fiber.gain_tau = (1) input gain_tau
96                   (2) 840e-6; % s; 840 us is the lifetime of Yb ions
97 *fiber.t_rep = (1) input t_rep
98                (2) 1/15e6; % s; assume 15 MHz repetition rate
99 *fiber.gain_cross_section = (1)input gain_cross_section
100                           (2) 6.43e-25 + 4.53e-26; % m^2; the total cross
                                  section of Yb ions at 1030 nm
101
102 fiber.saturation_intensity = (1) input saturation_intensity
103                            (2) calculated according to gain_tau, t_rep, and
                                   gain_cross_section above
104 fiber.saturation_energy = (1) input saturation energy
105                          (2) calculated according to saturation_intensity and
                                 gain_Aeff above
```

Chapter 8

Diagram of the calling sequence

It's not necessary to know how or when each function is called. I keep it here for documentation or in case someone wants to modify the code.

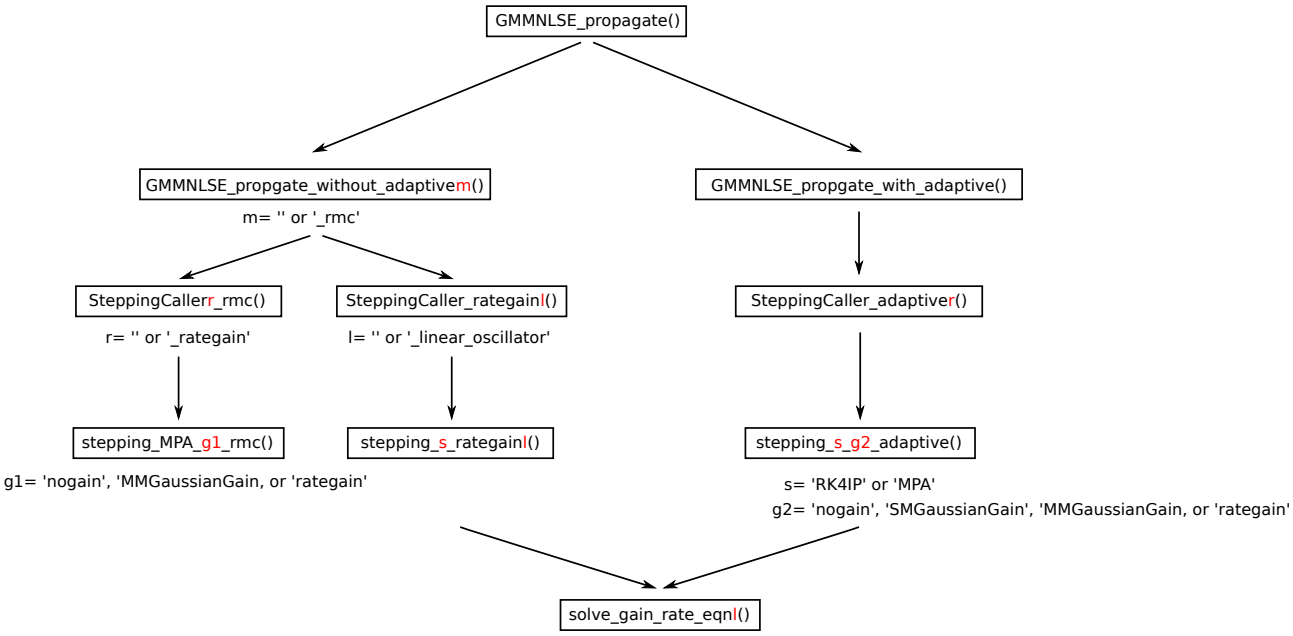


Figure 8.1: Diagram of the calling sequence.

The code uses "RK4IP" (Runge-Kutta in the interaction picture) for single mode and "MPA" (Massively Parallel Algorithm) for multimode.