Solver Guide for the MATLAB solid-core-fiber pulse propagation with transient gain

Yi-Hao Chen
Applied and Engineering Physics, Cornell University

January 27, 2025


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 single-mode scalar (linearly-polarized) unidirectional pulse propagation equation (UPPE) with transient gain:

$$
\begin{aligned}
&\partial_z A_s^{\pm}(z,\Omega_s) \\
&= \left(\hat{\mathcal{D}} + \hat{\mathcal{G}} + \hat{\mathcal{N}}\right) A_s^{\pm}(z,\Omega_s) \\
&= i\left[\beta(\omega) - \left(\beta_{(0)} + \beta_{(1)}\Omega\right)\right] A_s^{\pm}(z,\Omega_s) \\
&\quad + \left(\partial_z A_s^{\pm}(z,\Omega_s)\right)_{\hat{\mathcal{G}}} \\
&\quad + i\frac{\omega n_2}{cA_{\text{eff}}}\Bigg\{ (1-f_R)\,\mathfrak{F}_s\left[\left|A^{\pm}\right|^2 A^{\pm} + \tilde{\Gamma}^{K,\pm}\right] \\
&\qquad\qquad + f_R \mathfrak{F}_s\left[\left[\left(f_a h_a(t) + f_b h_b(t)\right) * \left(\left|A^{\pm}\right|^2 + \tilde{\Gamma}^{R,\pm}\right)\right] A^{\pm}\right]\Bigg\},
\end{aligned}
\tag{1.1}
$$

where $A^{\pm}(z,T)$ is the electric field envelope ($\sqrt{\text{W}}$), whose scaled Fourier transform is $A_s^{\pm}(z,\Omega_s) = \mathfrak{F}_s[A^{\pm}(z,T)]$. z is the propagation distance. $\hat{\mathcal{D}}$, $\hat{\mathcal{G}}$, and $\hat{\mathcal{N}}$ are dispersion, gain, and nonlinear operators. The scaled Fourier transform is applied with respect to angular frequency $\Omega_s = \frac{\Omega}{|c_s|} = \frac{\omega-\omega_0}{|c_s|}$, where ω_0 is the center angular frequency of the numerical frequency window and c_s is the scaling factor of the scaled Fourier transform. β is the propagation constant. $\beta_{(0)}$ and $\beta_{(1)}$ are to reduce the propagating global-phase increment, thus allowing a larger step size to facilitate simulations, in which $\beta_{(1)}$ is the inverse group velocity of the moving reference frame that introduces the delayed time $T = t - \beta_{(1)}z$. $(\cdot)_{\hat{\mathcal{G}}}$ represents that this derivative comes from the gain evolution. n_2 is the nonlinear refractive index. 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 ($f_a + f_b = 1$); $h_a(t)$ and $h_b(t)$ are isotropic and anisotropic Raman response functions [1, 2]. $\tilde{\Gamma}^{R,\pm}$ and $\tilde{\Gamma}^{K,\pm}$ are to model the spontaneous Raman and noise-seeded

Kerr contributions [3], following

$$\tilde{\Gamma}^K(T) = 2A^{\text{noise}}|A|^2 + A^2\left(A^{\text{noise}}\right)^* \tag{1.2a}$$

$$\tilde{\Gamma}^R(T) = A\left(A^{\text{noise}}\right)^* + A^{\text{noise}}A^*. \tag{1.2b}$$

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 GNLSE.

The gain term $\hat{\mathcal{G}}A_s^{\pm}(z,\Omega_s)$ is solved with the following coupled equations:

$$\frac{\partial N_n}{\partial t}(z,T) = \sum_{j(n)} C_j^{(1)} R_j(z,T) N_{(j)}(z,T) + \mathcal{O}_n(z,T), \tag{1.3a}$$

$$\text{with } R_j(z,T) = \sum_{\pm}\left\{\frac{\sigma_j(\nu_{\mathbf{p}})}{h\nu_{\mathbf{p}}}\frac{P_{\mathbf{p}}^{\pm}(z,T)}{A_{\text{cladding}}} + \frac{\Gamma}{A_{\text{core}}}\left|\mathfrak{F}_s^{-1}\left[\sqrt{\frac{\sigma_j(\nu)}{h\nu}}A_s^{\pm}(z,\Omega_s)\right]\right|^2\right\} \tag{1.3b}$$

$$\frac{\partial P_{\mathbf{p}}^{\pm}}{\partial z}(z,T) = \Gamma_{\mathbf{p}} P_{\mathbf{p}}^{\pm}(z,T)\sum_j C_j^{(2)}\sigma_j(\nu_{\mathbf{p}})N_{(j)}(z,T) \tag{1.3c}$$

$$\frac{\partial A^{\pm}}{\partial z}(z,T) = \sum_j C_j^{(2)}\frac{\Gamma N_{(j)}(z,T)}{2}\mathfrak{F}_s^{-1}\left[\sigma_j(\nu)A_s^{\pm}(z,\Omega_s)\right] + \sum_{j_+}\frac{\mathrm{d}A_{\text{SE},j_+}^{\pm}(z,T)}{\mathrm{d}z}, \tag{1.3d}$$

$$\text{with } \frac{\mathrm{d}A_{\text{SE},j_+}^{\pm}}{\mathrm{d}z}(z,T) = \sqrt{\Gamma N_{(j_+)}(z,T)}\,\frac{\mathrm{d}a_{\text{SE},j_+}^{\pm}}{\mathrm{d}z}(z,T) \tag{1.3e}$$

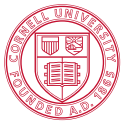
where the overlap integrals of signal/ASE and pump are $\Gamma = \int_{A_{\text{core}}}\left|F(\vec{r}_\perp)\right|^2\mathrm{d}^2x = 1 - \mathrm{e}^{-2a^2/w_0^2}$ with a the core radius and w_0 the beam waist [4], and $\Gamma_{\mathbf{p}} = \frac{A_{\text{core}}}{A_{\text{cladding}}}$, respectively. $j(n)$ represents specific summation terms for the population evolution $\frac{\partial N_n}{\partial t}$ (with $n=0$ being the ground level). $C_j^{(1)}$ is 1 if $R_jN_{(j)}$ contributes to the addition of N_n or -1 otherwise. σ_j is the absorption/emission cross section (m^2) with $C_j^{(2)}$ equal to -1 for absorption and $+1$ for emission. $N_{(j)}$ is the population density of the energy level that corresponds to σ_j. $\mathcal{O}_n$ represents terms independent of optical intensity, such as Γ_1N_1 for multiphonon decay or $A_{10}N_1$ for spontaneous transitions from the upper level to the lower level [5–7].

It is worth noting that the field A is defined as

$$\begin{aligned}\vec{\mathbb{E}}(\vec{x},t) &= \frac{1}{2}\left[\vec{\mathcal{E}}(\vec{x},t) + \text{c.c.}\right] \quad ,\vec{\mathcal{E}} \text{ is the analytic signal of } \vec{\mathbb{E}} \\ &= \int \mathrm{d}\omega\frac{1}{2}\left\{\frac{\vec{F}(x,y,\omega)}{N_1(\omega)}A(z,\omega)\mathrm{e}^{i\left[\beta(\omega)z-\omega t\right]} + \text{c.c.}\right\}.\end{aligned} \tag{1.4}$$

This makes MATLAB "ifft" become the Fourier Transform and "fft" become the *inverse* Fourier Transform. The convolution theorem also becomes different with different conventions. In our package, we follow this convention [Eq. (1.4)]. For example, to see the spectrum, please use

```
c = 299792.458; % nm/ps
wavelength = c./f; % nm
```



```
Nt = size(field,1);
dt = t(2)-t(1); % ps
factor_correct_unit = (Nt*dt)^2/1e3; % to make the spectrum of the correct unit
    "nJ/THz"
                                     % "/1e3" is to make pJ into nJ
spectrum = abs(fftshift(ifft(field),1)).^2*factor_correct_unit; % in frequency
    domain
```

Use "fftshift" to shift the spectrum from small frequency to large frequency. Note that it is not "ifftshift". They differ when the number of points is odd. To understand this, think about what the first data point is in "ifft(field):" it is the zero-frequency component, so we need to use "fftshift" with the frequency defining as

```
f = f0+(-Nt/2:Nt/2-1)'/(Nt*dt); % THz
```

For detailed discussion of the Fourier transform, especially in the field of ultrafast optics, please see the tutorial I recently published [8].

1.2 High-level understanding of this package

This package is designed to solve for nonlinear pulse propagation with transient gain in a single-mode fiber. 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. The package uses "RK4IP" (Runge-Kutta in the interaction picture) [9, 10].

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 Transient_gain_UPPE_propagate() MATLAB function.

Below is how to call this function in general.

```
prop_output = Transient_gain_UPPE_propagate(fiber, ...
                                            initial_condition, ...
                                            sim, ...
                                            gain_rate_eqn)
```

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

fiber
: It contains the information of the fiber, such as the index profile.

initial_condition
: It contains the information of the input.

sim
: It contains a multitude of information about the simulation, such as the algorithm to use and the center wavelength, etc.

gain_rate_eqn
: It contains the information required for the transient-rate-equation gain model, such as the pump power, the doped ion density, etc.


CORNELL UNIVERSITY
FOUNDED A.D. 1865

Chapter 3

Input arguments

Below, I use N_t as the number of time/frequency sampling points, N_ℓ as the number of energy levels to consider in population rate equations. N_z is the number of saved fields after the propagation. N_w is the number of windows. Overall, the dimension of simulation arrays follows $N_t \times N_\ell \times N_z \times N_w$.

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.

Some parameters are required only when you enable some settings. Below I labeled 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 has the unit of ps^n/m.

It's a column vector of

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

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 a $N_t \times 1$ array. It's a function of frequency, with an order from small to large.

n2

It's the nonlinear coefficient of the fiber. By default, Transient_gain_UPPE_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 initially left empty.

L0

This is the fiber length. The unit is meter.

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

3.2 initial_condition

This is the $1 \times N_w$ structure array containing the information of fields in different windows.

There are a few restrictions:

1. Fields in different windows should have the same number of sampling points N_t.
2. Coherent-field window should have the same temporal sampling period $\triangle t$, but $\triangle t$ have no restrictions and can be different among incoherent-field windows.

For $i = 1 \cdots$ num_windows,

initial_condition(i).dt
This is the temporal sampling period $\triangle t$ with a unit of ps in the ith window.

initial_condition(i).fields.forward
This is the temporal amplitude of the input forward fields (with the unit $\sqrt{\mathrm{W}}$). Its size is $N_t \times 1$.

initial_condition(i).fields.backward
This is the temporal amplitude of the input backward fields (with the unit $\sqrt{\mathrm{W}}$). Its size is $N_t \times 1$.

3.3 sim

Below are the most basic parameters for a simulation.

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 "betas" is a 2×1 column vector.

$$\begin{bmatrix} \beta_{(0)} \\ \beta_{(1)} \end{bmatrix} \tag{3.2}$$

By default, under the narrowband case where fiber.betas is a column vector of the Taylor series coefficient of $\beta(\omega)$, Transient_gain_UPPE_propagate() simply takes its parameters as "sim.betas:"

$$\begin{bmatrix} \beta_{(0)} \\ \beta_{(1)} \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}. \tag{3.3}$$

Cornell University

In the broadband case where β is a function of frequency, the code will compute the $\beta_{(0)}$ and $\beta_{(1)}$ at the center frequency of input signal pulse, and use it as the "sim.betas" here. Since there are multiple windows, only the window with the highest peak power is considered in this computation to select "sim.betas."

f0

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

dz

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.

save_period

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

Be aware that this number needs to be a divisor of the fiber length, fiber.L0; otherwise, Transient_gain_UPPE_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 Adaptive-step method

Here are the parameters if the simulation uses adaptive-step method. All parameters are contained within a "sim.adaptive_dz" 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_dz.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_dz.max_dz

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

3.3.2 Scaled Fourier transform

Narrowband transformation that duplicates the effect of scaled Fourier transform is implemented in both the incoherent and coherent fields.

For incoherent fields, the controlled parameters are saved in the **incoherent** structure:

incoherent.f0

The center frequency of incoherent-field windows in THz.

incoherent.dt

The temporal sampling period of incoherent-field windows in ps. This determines the scaling factor of the narrowband transformation.

This **incoherent** input is required only when ASE is considered.

For coherent fields, it is simply set with

cs

The scaling factor for the narrowband transformation of coherent-field windows. Unless it's a CPA simulation, this parameter should be 1 and the narrowband transformation is turned off. Narrowband transformation for coherent fields is useful only in CPA simulations.

3.3.3 Algorithm to use

gpu_yes

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

include_Raman

false(0) ignore Raman effect
true(1) Raman model including the anisotropic contribution ("Ch. 2.3, p. 43" and "Ch. 8.5, p. 340," Nonlinear Fiber Optics (5th), Agrawal)

Typically, only isotropic Raman is considered, which is based on a single vibrational Raman mode of molecules (Ch. 2.3, p.42, Nonlinear Fiber Optics (5th), Agrawal). Here, we include the anisotropic part if there is an existing model, such as the one in silica ("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.

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

cuda_dir_path

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

gpuDevice.Index

The GPU to use. It's typically 1 if the computer has only one GPU. MATLAB starts the index with 1. By starting multiple MATLAB sessions, we can run simulations on different GPUs simultaneously if different sessions are set with different GPU indices here.

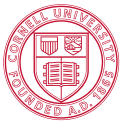
Here are the parameters for the progress bar used in the simulation. It's useful in general to see how a simulation progresses.

progress_bar

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

progress_bar_name

The name of the UPPE 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

This is the $1 \times N_w$ structure array containing the information of output fields in different windows.

For $i = 1 \cdots$ num_windows, the output, which we use "prop_output" as the variable name here, contains

prop_output(i).dt
This is the temporal sampling period $\triangle t$ (ps) in the ith window.

prop_output(i).fields.forward
This is the temporal amplitude of the output forward fields ($\sqrt{\mathrm{W}}$). Its size is $N_t \times 1 \times N_z$.

prop_output(i).fields.backward
This is the temporal amplitude of the output backward fields ($\sqrt{\mathrm{W}}$). Its size is $N_t \times 1 \times N_z$.

prop_output(i).Power.pump.forward
The forward pump power (W) along the fiber. Its size is $N_t \times 1 \times N_z$.

prop_output(i).Power.pump.backward
The backward pump power (W) along the fiber. Its size is $N_t \times 1 \times N_z$.

prop_output(i).population
The doped ion density (m^{-3}) of upper levels. Its size is $N_t \times N_\ell \times N_z$.

Above are specific information for each window. The rest are common parameters that are the same for all windows.

z
This is the positions of each saved field (m).

dz
The z-step size (m).

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 (ps) of the pulse at each saved point due to pulse centering.

seconds
The time spent for this simulation (s).

Chapter 5

load_default_UPPE_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_UPPE_propagate(input_fiber,...
                                          input_sim])
```

input_fiber and input_sim are user-defined parameters. Below are some examples.

```
% User-defined parameters
fiber.L0 = 3; % m

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

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

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

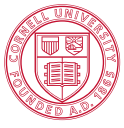
Besides loading the default values, this function gives a user more options to obtain several parameters. This function transforms them into the allowed parameters of UPPE_propagate(). I list them below. If both equivalence are specified unfortunately, the allowed UPPE_propagate() input has the higher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

Description	Allowed UPPE_propagate()'s input	Equivalent input arguments for this function
center frequency/wavelength	sim.f0 (THz)	sim.lambda0 (m)

Below is the process flow of this "load_default_UPPE_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) 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.material = 'silica';
fiber.n2 = 2.3e-20;

sim.dz = 1000e-6;
sim.save_period = 0;

sim.adaptive_dz.threshold = 1e-6;

sim.gpu_yes = true;
sim.pulse_centering = true;
sim.include_Raman = true;
sim.gpuDevice.Index = 1;
sim.progress_bar = true;
sim.progress_bar_name = '';
sim.cuda_dir_path = 'Transient_rate_gain/cuda';

%<-- Correlated parameters are loaded based on the input or default -->

%   Assume 920 nm for positive dispersion if lambda0 < 1000 nm,
fiber.betas = [9.8810e6; 4.8872e3; 0.0315; 2.0457e-5; 1.2737e-9];
fiber.MFD = 4; % um; 1030nm from IXblue's IXF-2CF-PAS-PM-4-80-0.16-P
%   Assume 1030 nm for positive dispersion if 1000 nm < lambda0 < 1300 nm (~ZDW
    for a silica fiber),
fiber.betas = [8.8268e6; 4.8821e3; 0.0209; 32.9e-6; -26.7e-9];
fiber.MFD = 5.95; % um; from Thorlabs 1060XP
%   Assume 1550 nm for negative dispersion if lambda0 > 1300 nm (~ZDW for a
    silica fiber),
fiber.betas = [5.8339e6; 4.8775e3; -0.0123; 0.1049e-6; -378.3e-9];
fiber.MFD = 8.09; % um; from Thorlabs 1060XP

(input SR precedes over input MFD)
fiber.SR = (1) input SR, if there's input SR
           (2) 1/Aeff, if (a) there's input MFD
                          (b) MFD is taken from the default one and there's no
                              input MFD

fiber.L0 = (1) input L0
           (2) 2 (m)
```



Chapter 6

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.

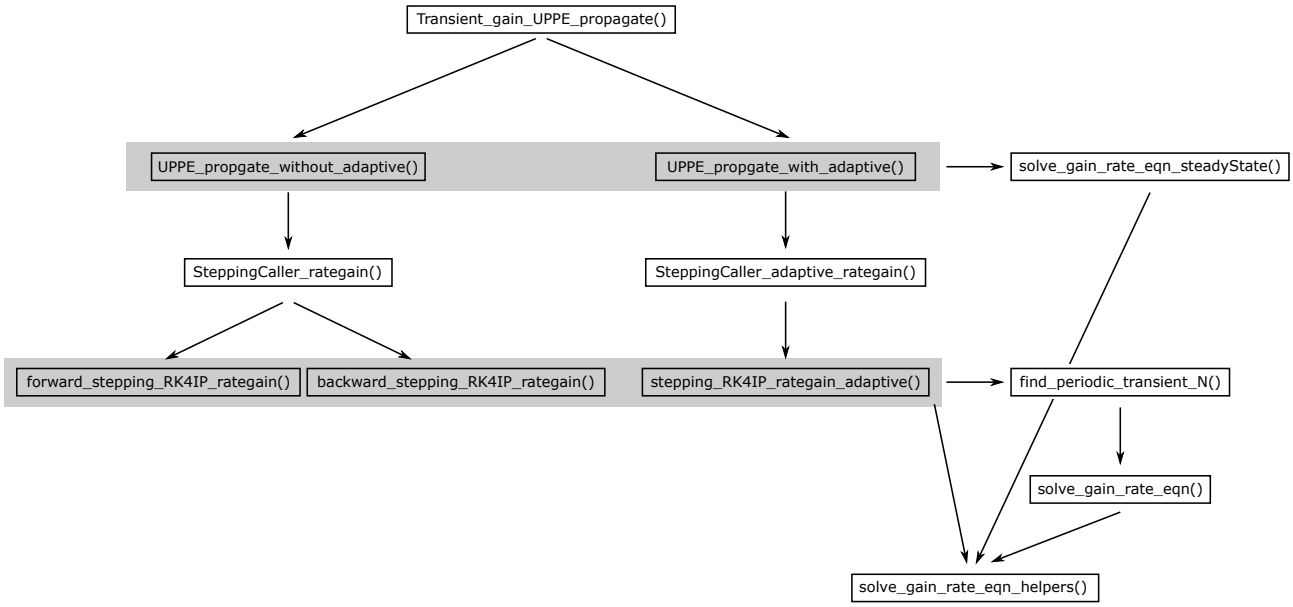
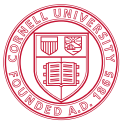


Figure 6.1: Diagram of the calling sequence.

The code uses "RK4IP" (Runge-Kutta in the interaction picture) [9, 10] with an adaptive step-size control.



Bibliography

[1] R. H. Stolen, J. P. Gordon, W. J. Tomlinson, and H. A. Haus, "Raman response function of silica-core fibers", J. Opt. Soc. Am. B **6**, 1159–1166 (1989).

[2] Q. Lin and G. P. Agrawal, "Raman response function for silica fibers", Opt. Lett. **31**, 3086–3088 (2006).

[3] Y.-H. Chen and F. Wise, "A simple accurate way to model noise-seeded ultrafast nonlinear processes", arXiv preprint arXiv: 2410.20567 (2024).

[4] T. J. Whitley and R. Wyatt, "Alternative Gaussian spot size polynomial for use with doped fiber amplifiers", IEEE Photon. Technol. Lett. **5**, 1325–1327 (1993).

[5] F. C. Maciuc, C. I. Stere, and A.-R. P. Sterian, "Rate equations for an erbium laser system: a numerical approach", in Romopto 2000: sixth conference on optics, Vol. 4430, edited by V. I. Vlad (International Society for Optics and Photonics, 2001), pp. 136–146.

[6] C. Huang, Y. Tang, S. Wang, R. Zhang, J. Zheng, and J. Xu, "Theoretical Modeling of Ho-Doped Fiber Lasers Pumped by Laser-Diodes Around 1.125 μm", J. Light. Technol. **30**, 3235–3240 (2012).

[7] C. D. Boley, J. W. Dawson, L. S. Kiani, and P. H. Pax, "E-band neodymium-doped fiber amplifier: model and application", Appl. Opt. **58**, 2320–2327 (2019).

[8] Y.-H. Chen, "Tutorial of Fourier and Hankel transforms for ultrafast optics", arXiv preprint arXiv: 2412.20698 (2025).

[9] A. M. Heidt, "Efficient Adaptive Step Size Method for the Simulation of Supercontinuum Generation in Optical Fibers", J. Light. Technol. **27**, 3984–3991 (2009).

[10] S. Balac and F. Mahé, "Embedded Runge-Kutta scheme for step-size control in the interaction picture method", Comput. Phys. Commun. **184**, 1211–1219 (2013).