MATLAB GMMNLSE Solver User Guide

Zachary Ziegler[1], Logan Wright[1] and Frank Wise[1]

[1]School of Applied and Engineering Physics, Cornell University, Ithaca, New York 14853, USA

December 20, 2017

I don't want to read this, how do I run a simulation?

Fair enough. Run the script **Examples/MM Propagation/GMMNLSE_driver_OM4_lowpower_cpu.m** or **Examples/MM Propagation/GMMNLSE_driver_OM4_lowpower_gpu.m**. All you need to run this is MATLAB, but if you want to use the GPU version (which is highly recommended), see the Setup section below.

1 Overview

This package contains all the code needed to simulate multimode pulse propagation in multimode optical fiber, based around the Massively Parallel Algorithm implemented efficiently on the GPU. The core of the package steps a multimode field through a number of small steps Δz in parallel; the rest of the code provides user-friendly functions to propagate over an arbitrary number of steps and to set up the simulations from a given index profile. While a modest speed-up is achieved through the use of the MPA algorithm, the use of the GPU alone gives a significant speed-up compared to the same calculation on the CPU. Using a GPU is not required, therefore, but it is strongly suggested if possible. Run on a modern GPU, this code enables realistic multimode propagation simulations.

Specifically, the code solves the Generalized Multimode Nonlinear Schrödinger equation (GMMNLSE) [1]:

$$\begin{aligned}\frac{\partial A_p}{\partial z} &= i(\beta_0^{(p)} - \Re[\beta_0^{(0)}])A_p - (\beta_1^{(p)} - \Re[\beta_1^{(0)}])\frac{\partial A_p}{\partial t} + i\sum_{n\geq 2}\frac{\beta_n^{(p)}}{n!}\left(i\frac{\partial}{\partial t}\right)^n A_p \\ &+ i\frac{n_2\omega_0}{c}\left(1+\frac{i}{\omega_0}\frac{\partial}{\partial t}\right)\sum_{l,m,n}\left\{(1-f_R)S^K_{plmn}A_lA_mA_n^* + f_R S^R_{plmn}A_l[h*(A_mA_n^*)]\right\}\end{aligned} \tag{1}$$

Where A_p is the field envelope of the pth mode, $\beta_n^{(p)}$ is the nth order dispersion coefficient for the pth mode, n_2 is the nonlinear index coefficient, ω_0 is the center angular frequency,

f_R is the fractional Raman contribution to the nonlinearity, S^K_{plmn} and S^R_{plmn} are the modal overlap factors for the Kerr term and the Raman term, respectively, h is the Raman response function, t is time in a reference frame moving with the fundamental mode at the central frequency, and z is the distance along the fiber.

2 Core files

The following is a list of the files included in the solver package, along with a short description of what they do. The various step methods are described first, followed by the propagation method, followed by the auxiliary functions. More details about the exact input and output can be found in comments at the top of each function.

2.1 Step methods

This package includes two step functions, one using the more traditional split step method and the other using the Massively Parallel Algorithm (MPA).

`GMMNLSE_ss_step`

Performs one small step Δz, using the split-step method. The dispersion is applied during the first half step exactly, then the nonlinearity is applied over the full time step using a 4th order Runge-Kutta method, and finally the dispersion is applied again over the last half step.

`GMMNLSE_MPA_step`

Performs M small steps of size Δz in parallel, using the Massively Parallel Algorithm. The dispersion is again accounted for exactly, but now the nonlinear phase accumulation is calculated for each small step in parallel and then integrated to calculate the full nonlinear phase accumulation. This process is iterated until a desired tolerance in the change of the solution is met. See the description by Pavel Lushnikov.

2.2 Propagation (multiple steps)

`GMMNLSE_propagate`

This function takes in parameters of the fiber, parameters of the simulation, and an initial condition, and propagates the field through a number of steps to the end of the fiber. It first performs some checks to ensure the input is valid, formats the SˆK and SˆR tensors to maximize efficiency and precomputes as much as possible (the dispersion term, and the Raman response). Then it enters the main loop, calling the appropriate step function each iteration.

2.3 Pre-propagation functions

`solve_for_modes`

Given an index profile function and the parameters of the profile, this function solves Maxwell's equations to find the modes of the fiber and their propagation constants. Specifically, it calculates the modes and propagation constants over a range of frequency, which is required to calculate the dispersion of each mode. The modes used in this simulation package are approximated as scalar modes.

`calc_dispersion`

After calculating the propagation constants for each mode with `solve_for_modes`, this function imports the results, approximates the dispersion with a polynomial, and saves the dispersion coefficients.

`calc_SRSK_tensors`

After calculating the modes with `solve_for_modes`, this function calculates the overlap tensors as in [1]. The polarization of the modes is assumed to be either linear or circular, in which case the SK and SR tensors are constant multiples of one another. In the more general case the code would have to be slightly modified for other polarization states.

3 Setup

Run on the CPU, the GMMNLSE solver only requires the base version of MATLAB without any extra packages. So far, the code has been tested on MATLAB 2016b.

This GMMNLSE solver has also been optimized to run efficiently on Nvidia GPUs (due to CUDA support). The code has been tested on 800-series, 900-series, and 1000-series GPUs, although it should work for older GPUs as well. In general the newer GPUs will give better performance, in addition to having more memory. The GPU memory primarily determines the number of modes; for a sense of scale, on the Titan X with M=10 and N=2^{13}, the GPU can support 125 modes. So almost all modern GPUs will have enough memory to simulate a reasonable number of modes .

Using a GPU with MATLAB requires the MATLAB Parallel Computing Toolbox. Furthermore, because the solver uses a small section of custom CUDA code to greatly accelerate the simulations, the Nvidia CUDA Toolkit is also required. To enable GPU simulations, the following steps should be taken:

1. Install the Nvidia drivers. If the GPU has been used before these will probably already be installed. If on Windows, install Nvidia's GeForce Experience software and follow the steps to download the drivers. If on Linux, you will get the best performance with Nvidia's proprietary drivers which can be installed either via most package managers (recommended) or directly from Nvidia's website.

2. Install a C++ compiler, if you do not have one already. The CUDA compiler driver `nvcc` relies on a pre-existing C++ compiler, and will not function without one. If on Windows, the free Visual Studio Community is recommended. If on Linux, `gcc` is recommended and should be installed already.

3. Install the Nvidia CUDA Toolkit. You can find the download page here, and the installation instruction guides here for Linux and here for Windows. This will install the Nvidia CUDA compiler driver `nvcc`.

4. Add the C++ compiler and `nvcc` to the PATH. On Windows `nvcc.exe` is located by default in `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\bin` or similar. If using Visual Studio the C++ compiler `cl.exe` may be located in `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin`, although the path may be different for different versions. These can be added to the PATH by adding them to the list located at Control Panel → System and Security → System → Advanced system settings → Environment Variables in Windows 10 and Windows 8. In Windows 7, right click on 'My Computer' then choose Properties → Advanced system settings → Environment Variables

3.1 Visual Studio 2017 and CUDA 9.0: compatibility issue

As of December 2017, the newest versions of Visual Studio (2017) do not support CUDA 9, the latest CUDA version. This requires a workaround, which we expect will not be necessary when a new version of CUDA is released.

For now, if the above steps don't work because of Visual Studio 2017, you can create a second legacy installation. Following the instructions here, install the "VC++ 2015.3 v140 toolset for desktop (x86,x64)" component. By installing this, you'll find a "Microsoft Visual Studio 14.0" folder under "Program File (x86)". You can then add this to the PATH as descibed in Step. 4 above.

4 Use

Some examples of how to use the tools included in this package to simulate pulses according to the GMMNLSE can be found in the Examples folder. Generally, the workflow is as follows:

1. Choose the type and parameters of fiber, and solve for the modes with `solve_for_modes.m`. The type of fiber is selected through the function used the generate the refractive index profile. Functions to build common fiber types such as step index fiber or GRIN fiber are included, and the specific fiber parameters can be modified. The modes can be solved for first at a single wavelength, to get an idea of the number of guided modes and shape, and then again over a range of wavelengths which is needed to calculate the dispersion.

2. Calculate the dispersion coefficients with `calc_dispersion.m`. This script takes propagation constants from the modes that have been previously calculated, fits them to a polynomial, and extracts the dispersion coefficients.

3. Calculate the mode overlap tensors with `calc_SRSK_tensors.m`. This script takes the spatial fields of the modes at a single wavelength and computes the modes overlap tensors according to [1].

4. Build the system to simulate in a file similar to any of the `GMMNLSE_driver.m` files. This is a good place to set the simulation parameters and then run the `GMMNLSE_propagate` function to do the actual propagation.

5 Fast Fourier Transform convention

As the code included here uses a spectral method to solve the GMMNLSE, the FFT plays an important role in the simulation code. The optics community tends to use the convention where the Fourier transform pair is defined as

$$\mathcal{F}_{optics}\{f(t)\} = \int f(t)e^{i\omega t}dt \tag{2}$$

$$\mathcal{F}^{-1}_{optics}\{f(\omega)\} = \frac{1}{2\pi}\int f(\omega)e^{-i\omega t}d\omega \tag{3}$$

whereas MATLAB follows the more traditional definition where the negative sign in the exponent is included in the forward transform. Ultimately as long as the convention is consistent it does not matter physically which convention is used, however for easy comparison with the rest of the optics community in the frequency domain the optics convention is used in this simulation package. To accomplish this in MATLAB, the forward transform is performed with `ifft` and the backward transform is performed with `fft`. This does not affect the results of the simulation, but when visualizing the output it is suggested to retain the convention as well.

6 Massively Parallel Algorithm (MPA)

The core simulation code in this package uses the MPA algorithm to take a number of small steps together in parallel. The GMMNLSE MPA algorithm was developed by Pavel Lushnikov, Department of Mathematics and Statistics, University of New Mexico, as an extension of a similar algorithm for the single mode NLSE. In short, the algorithm treats the linear dispersion term exactly in the Fourier domain and solves for the nonlinear phase accumulation over a number of small steps Δz in parallel, initially assuming the nonlinear phase accumulated over the whole step is small. This process is then iterated until a self-consistent solution with a desired level of accuracy is obtained. See the notes in the `Documentation` folder for more details.

7 Acknowledgements

The svmodes function is used from the MATLAB file exchange, described in A. B. Fallahkhair, K. S. Li and T. E. Murphy, "Vector Finite Difference Modesolver for Anisotropic

Dielectric Waveguides", J. Lightwave Technol. 26(11), 1423-1431, (2008).

8 Contact

Technical questions may be directed to Zachary Ziegler (zmz4@cornell.edu) or Logan Wright (lgw32@cornell.edu).

References

[1] Peter Horak and Francesco Poletti. Multimode Nonlinear Fibre Optics : Theory and Applications. In Moh Yasin, editor, *Recent Prog. Opt. Fiber Res.* InTech, 2012.