Documentation for laser-propagation

Jeffrey M. Brown

May 21, 2019

Contents

1 Field propagation

1.1 Propagation equation

The propagation of the electric field is represented by the unidirectional pulse propagation equation written in the spectral domain as

$$\frac{\partial \mathcal{E}(z,k_\perp,\omega)}{\partial z} = ik_z(k_\perp,\omega)\mathcal{E}(z,k_\perp,\omega) + \frac{i\omega^2}{2\epsilon_0 c^2 k_{z,n}(k_\perp,\omega)}\left[\mathcal{P}(z,k_\perp,\omega) + i\frac{\mathcal{J}(z,k_\perp,\omega)}{\omega}\right] \tag{1}$$

where $k_z(k_\perp,\omega)$ describes how the spectral field amplitudes propagate linearly in the medium (both diffraction and linear absorption). The nonlinear properties of the medium are described by functions representing the nonlinear polarization P and/or nonlinear current J. The functional forms of k_z, P, and J are determine by the parameter file supplied by user at runtime. All of the available options are detailed in Section 2.

It's not ideal to solve Equation (1) directly since it still contains a fast oscillating term resulting from linear propagation. The is remedied by defining the spectral amplitude $A(k_\perp,\omega) = E(k_\perp,\omega)\exp[-ik_z z]$, whose evolution is the govern by the nonlinear sources only

$$\partial_z \mathcal{A}(z,k_\perp,\omega) = \frac{i\omega^2}{2\epsilon_0 c^2 k_z(k_\perp,\omega)} e^{-ik_z z}\left[\mathcal{P}(z,k_\perp,\omega) + i\frac{\mathcal{J}(z,k_\perp,\omega)}{\omega}\right] \tag{2}$$

The variables $A(k_\perp,\omega)$ are the native variables of the simulation. Their evolution equation is a large ODE system that is solved using GSL's ode solver library. This library provides many methods of solving ODE's and includes adaptive step control. The ODE solver only requires the RHS of Eq. 2 and some error tolerance values to control the adaptive stepping algorithm.

1.2 Algorithm

At each step in solving Eq. (2), the solver invokes a call to the RHS with a proposed step size forward in z. Since nonlinear sources P and J are typically defined in real space, the RHS function has the job of spectrally transformed $\mathcal{A}(k_\perp,\omega)$ to $E(r,t)$, calling each nonlinearity, and transforming back to spectral amplitudes. The RHS function returns the change in spectral amplitudes $\partial_z \mathcal{A}(z,k_\perp,\omega)$ to the solver, which determines whether the proposed step was small enough to evolve the amplitudes $\mathcal{A}$ within a given error tolerance.

Explicitly, during each step of the ODE solver, the RHS is calculated the following way:

1. Linear propagate $\mathcal{A}$ forward by Δz (ω/v_g term keeps the pulse centered in the computational box)

$$\mathcal{E}(k_\perp,\omega) = A(z,k_\perp,\omega)\exp\left[i\left(k_z\left(k_\perp,\omega\right) - \omega/v_g\right)\Delta z\right] \tag{3}$$

2. Transform to real space in radial coordinates using the Fourier-Hankel transform

$$E(r,t) = \text{Hankel}\{\mathcal{E}(k_\perp)\}\text{FT}\{\mathcal{E}(\omega)\} \tag{4}$$

3. Calculate nonlinearities

$$P(r,t) = \sum_i P_i[E(r,t)]$$

4. Transform to spectral space

$$P(k_\perp,\omega) = \text{Hankel}\{P(r)\}\text{FT}^{-1}\{P(t)\}$$

5. Undo the propagation in Step 1

$$\mathcal{P}(k_\perp,\omega) = P(k_\perp,\omega)\exp\left[-i(k_z(k_\perp,\omega) - \omega/v_g)\Delta z\right] \tag{5}$$

6. Finally, multiply the nonlinearity by the coupling factor and return to ODE solver

$$\partial_z \mathcal{A}(k_\perp,\omega) = \frac{i\omega^2}{2\epsilon_0 c^2 k_z}\mathcal{P}(k_\perp,\omega) \tag{6}$$

1.3 Computational Grid

The computational grids in the simulation are three 2D arrays representing the fields $\mathcal{A}(k_\perp,\omega)$ and $E(r,t)$, and also the electron density $\rho(r,t)$ in cylindrical coordinates. There are also some auxillary arrays that are used for computing the spectral transforms. The spectral transformation of this grid involves a Fourier transform between time t and frequency ω and a Hankel transform between r and $k_\perp$.

To perform the Fourier transform the library FFTW is used. A norm-preserving Hankel transform was introduced by H. Fisk Johnson in 1986. The radial grid points are not equally spaced, but are proportional to the zeros of the Bessel $J_0(\alpha_i) = 0$, and are defined as $r_i = R_{max}\alpha_i/\alpha_N$ where N is the number of radial grid points. The values of $k_{\perp,i} = \alpha_i/R_{max}$. With this coordinate transformation, the Hankel transform is simply a matrix multiplication with elements

$$H_{mn} = \frac{2J_0(\alpha_n\alpha_m/\alpha_N)}{J_1(\alpha_n)J_1(\alpha_m)\alpha_N}$$

The Hankel transform is its own inverse so multiplication twice is the identity: $HHf = f$. This multiplication is speed up by using the library Eigen, which generates SIMD instructions resulting in a speed up of 30-40% in freespace calculations.

The size of the spectral amplitudes $\mathcal{A}$ (number of points in $k_\perp$ and ω) is always smaller than the size of E, since only positive frequency components of $\mathcal{A}$ are propagated. This saves quite a bit of memory since the number of active frequencies ω are less than half the size of the number of time points in the grid. Using a capillary waveguide with only a few number of modes (the number of points in $k_\perp$ greatly reduces the size of $\mathcal{A}$ as well. For example, a capillary waveguide simulation with a real-space grid size of $Nr \times Nt = 100 \times 16384 = 1.6M$ points is represented by the spectral amplitude grid-size of $N\omega \times Nk_\perp = 2199 \times 10 = 22k$ points when using 10 capillary modes and only propagating the active frequencies corresponding to the wavelengths between 250 nm - 3000 nm.

It's good to keep in mind the algorithmic scaling of the two transformations: FFT and Hankel. The FFT is performed for each radial grid point and scales as $\mathcal{O}(NrNt\log_2 Nt)$. FFT runs most efficiently with Nt being of a power of two: 2^k. The Hankel transform has a different scaling for freespace and capillary calculations. For freespace calculations a full matrix multiplication is done for each active frequency ω and thus scales as $\mathcal{O}(N\omega Nr^2)$. Due to the reduced $Nk_\perp$ in capillary simulations this complexity can be reduced to $\mathcal{O}(N\omega Nk_\perp Nr)$. Since the number of capillary modes $Nk_\perp$ is usually around 5-10, the Hankel transform in the capillary simulations scale linearly with the number of radial grid points.

2 Parameter file

The type of simulation, medium model, choice of nonlinearities, and output are controlled by a parameter file that is passed to the executable as an argument on the command line. All values in the parameter file are assumed to be in SI units. There are some values that are required in order to run any type of simulation (e.g. setting up the computational grid, initial conditions, etc), but many values (e.g. selecting which nonlinearities to include) are optional and only need to be included in the parameter file if you wish to have them in the simulation.

2.1 Syntax

The syntax of the parameter file is similar to conf or ini styles, where key-value pairs are grouped into sections. Here's an example:

```
[section1]
key = value
x = y

# top level comment
[section2]
s = hello
a = 1.1
x = -2e-12   # inline comment
```

Whitespace surrounding the keys, values, sections, and = symbol are ignored. Comments can be placed basically anywhere since all text after a hash # symbol is ignored. The keys and values can be read and runtime to the program by the `Parameters` class defined in `parameters.h` & `parameters.cc`.

To read the parameter file `input`:

```
[time]
N = 1024
```

```
time_min = -10e-15
time_max = 50e-15
```

instantiate the `Parameters` class with the filename and extract the values

```
Parameters::Parameters params("input");
int Nt = params.get<int>("time/N");
double time_min = p.get<double>("time/time_min");
double time_max = p.get<double>("time/time_max");
```

Internally, all keys and values are stored as strings and keys have had their surrounding section prepended to them using a `/`. The `Parameters` class has a template function `get` that uses stringstreams to interpret the type of value. This means that the syntax of the values is limited to readable C++ types: e.g. `string`, `int`, `double`, etc.

2.2 Setting up the computational grid

2.2.1 [time] (required)

This section defines the properties of the temporal and spectral domains of the field. For the temporal domain, the keys to define are the number of temporal points `N`, and the minimum `time_min` and maximum `time_max` values of the temporal range. For efficiency, `N` should be a power of 2. Additionally, only certain positive frequency components of the spectral field are propagated. To set the range of active frequencies, define `wavelength_min` and `wavelength_max`.

An example of this section is:

```
[time]
N = 2048
time_min = -100e-15
time_max = 200e-15
wavelength_min = 150e-9
wavelength_max = 4e-6
```

This defines a temporal box of 2048 points extending from -100fs to 200fs. The active frequencies in the simulation are only the ones that fall between 150nm and 4 microns.

2.2.2 [filtering] (optional)

This section defines temporal and spectral filtering. Filtering is crude since it amounts to multiplication of the temporal and spectral fields by $\sin^2$ mask before the forward and inverse Fourier transformations. The values defined here along with the ones defined in sections `[time]` determine how fast $\sin^2$ ramps go from 0 to 1. Setting a filter min/max value to a value outside the limits in `[time]` disables filtering on that particular side.

Reasonable filter values for the time domain in previous section could be:

```
[filtering]
time_filter_min = -90e-15
time_filter_max = 150e-15
wavelength_filter_min = 250e-9
wavelength_filter_max = 3e-6
```

Note: there is no filtering in r or $k_\perp$ which leads to reflections if the field reaches the radial boundary.

If interested in THz generation, the filter value on the long wavelength side should be set to a large value:

```
[filtering]
time_filter_min = -90e-15
time_filter_max = 150e-15
wavelength_filter_min = 250e-9
wavelength_filter_max = 1
```

2.2.3 [space] (required)

This section defines the spatial properties of the radial domain: `N` sets the number of radial points, and `radius_max` sets the radial extent of the domain. Since the radial spectral transform is the Hankel transform, there are no special values for `N` that are more efficient, unlike there for the time domain (due to the FFT).

An example of this section is:

```
[space]
N = 173
radius_max = 10e-3
```

This defines a radial domain of 173 points with a maximum extent of 10 mm.

2.3 Propagtion and ODE step control

2.3.1 [propagation] (required)

This section sets the physical starting and ending distances (`starting_distance` and `ending_distance`) of the simulation, and when data is written to the disk. The key `num_reports_cheap` sets the number of cheap diagnostics (fast to calculate or small in written data size) to perform between the starting and ending distances, and `num_reports_expensive` sets the number of expensive diagnostics. Whether a diagnostic is cheap or expensive is defined by the how the Observer was added in `main.cc:initialize_observers` via the function `main.cc:conditionally_add`.

An example of this section is:

```
[propagation]
starting_distance = 0
ending_distance = 4
num_reports_cheap = 40
num_reports_expensive = 4
```

The laser propagates from 0 to 4 meters and during propagation 40 cheap and 4 expensive diagnostics are performed.

2.3.2 [ode] (required)

This section defines the parameters for the ODE solver. Adjustment of these values has a great impact on the runtime of the simulation, since they force the solver to take small enough steps to satisfy these criteria. The values below are passed directly to GSL's ODE step control function. They form a two-parameter heuristic value $D_i = \epsilon_{abs} + \epsilon_{rel}|A_i|$ that is compared to the error calculated by the RK45 method $|Aerr_i|$.

For fast, but perhaps not the most accurate simulations, try:

```
[ode]
absolute_error = 1e10 # or 1e8
relative_error = 1e-4
first_step = 1e-3
```

This sets the two error parameters and the first attempted step of the simulation to be 1 mm.

For production runs where the data should be reasonable converged, try:

```
[ode]
absolute_error = 1e6
relative_error = 1e-4
first_step = 1e-3
```

2.4 Linear properties

2.4.1 [medium] (required)

This section defines the linear properties of the medium in the simulation. The key `type` controls the functional form of the linear propagator (basically the formula for $k_z(k_\perp, \omega)$), and can be set to either `freespace`, `capillary`, or `diffractionless`.

Selecting `freespace` sets

$$k_z(k_\perp, \omega) = \sqrt{\left(\frac{n(\omega)\omega}{c}\right)^2 - k_\perp^2}$$

Selecting `capillary` sets

$$k_z(k_\perp, \omega) = \sqrt{\left(\frac{n(\omega)\omega}{c}\right)^2 - k_\perp^2} + i\alpha(k_\perp)$$

where the absorption α due to the gas cladding interface is

$$\alpha = \frac{1}{2R}\left(\frac{k_\perp c}{\omega}\right)^2 \frac{n_{clad}^2 + 1}{\sqrt{n_{clad}^2 - 1}}$$

where R is the capillary radius and n_{clad} is the index of the cladding. If `capillary` is chosen, then the input file must contain the section `[capillary]` as described in 2.4.2.

Selecting `diffractionless` sets

$$k_z(k_\perp, \omega) = \frac{n(\omega)\omega}{c}$$

which is useful for verifying that temporal dispersion is computed properly.

The key `index` defines the index of refraction function $n(\omega)$. Currently implemented functions are: `vacuum`, `air`, `argon`, and `ethanol`. The index can also be defined using tabulated data from a file that will be linearly interpolated onto the points ω of the spectral domain. To do this set `index` equal to a filename ending with the `.dat` extension and contains three space-separated columns: ω $\Re\{n(\omega)\}$ $\Im\{n(\omega)\}$.

Lastly, the key `pressure` is used to increase or decrease the index of refraction using the Lorentz-Lorenz relation.

An example of freespace propagation in air at a pressure of 1 atm is:

```
[medium]
type = freespace
index = air
pressure = 1
```

An example of capillary propagation in argon at 3.3 atm is:

```
[medium]
type = capillary
index = argon
pressure = 3.3
```

2.4.2 [capillary] (optional)

This section defines a capillary waveguide which sets the functional form of $k_z(k_\perp, \omega)$ (adding absorption from gas-cladding interface) and also the number of propagating capillary `modes`. The index of the cladding is `cladding`.

```
[capillary]
cladding = 1.45
modes = 5
```

2.5 Initial conditions

2.5.1 [laser] (required)

This section defines the initial laser pulse. The key `type` can be `gaussian`, `restart`, or `file`.

An example of a Gaussian beam is:

```
[laser]
type = gaussian
wavelength = 800e-9
length = 20e-15
waist = 4e-3
focus = 2
energy = 1e-3
chirp = 0
phase_deg = 0
delay = 0
```

Here the laser has a envelope shape of Gaussian in space and time. It's central wavelength is 800 nm, FWHM temporal length is 20 fs, 1/e^2 radius is 4 mm, is focused at 2 m, has energy of 1 mJ, and has zero chirp, phase and temporal delay. Note: the `starting_distance` in `[propagation]` can be used to linear propagate forward in z assuming Gaussian beam propagation in space and a Gaussian-like propagation in time through a dispersive media (gvd and chirp are taken into account). This is useful for only performing a nonlinear simulation near the focus, while propagating linearly through a medium beforehand.

An example of restarting from an old simulation:

```
[laser]
type = restart
filename = A004.dat
wavelength = 800e-9
```

The `filename` must point to a spectral field file that was generated from a previous simulation, and you must specify the central wavelength (used for calculating values, such as $n0$, gvd, P_{cr}). Note: you might want to set the `starting_distance` in `[propagation]` to match the physical distance of the spectral field file.

An example of starting from a binary space-time field file:

```
[laser]
type = file
filename = init_field.dat
wavelength = 800e-9
```

Note: the field $E(r,t)$ in `init_field.dat` must be stored in a binary file of doubles (8 bytes) where the real and imaginary values are stored as [real0 imag0 real1 imag1 ...]. The time axis t is the fast axis (convention of C, opposite of Fortran). The sampling of r and t must match the values are set in the sections `[time]` and `[space]`.

2.6 Output data

2.6.1 [output] (optional)

This section defines which data is written to which files during propagation. All of the entries in this section are optional. Commenting or removing any line below results in no data being written for that particular item.

```
[output]
# field and density filename patterns and their associated distances
temporal_field = E
spectral_field = A
electron_density = Rho
```

```
distance = distance.dat

# coordinates
time = time.dat
radius = radius.dat
omega = omega.dat
kperp = kperp.dat
wavelength = wavelength.dat

# values along propagation
energy = energy.dat
max_intensity = max_intensity.dat
max_density = max_density.dat
```

The keys are fixed by the program, but the values can be changed by the user. The only special values are the ones for the field and density files (`temporal_field = E`, `spectral_field = A`, and `electron_density = Rho`) where their filename values contain no extension (e.g. dat or txt). This is because their filenames will be enumerated (e.g. E000.dat, E001.dat, E002.dat, etc.) and the distances at which they are written out is contained in `distance`.

2.7 Nonlinear properties

2.7.1 [kerr] (optional)

Defining this section adds Kerr nonlinearity to the simulation using

$$P_{\mathrm{NL}}(r,t) = \epsilon_0 \chi^{(3)} E^3(r,t)$$

where $\chi^{(3)} = \frac{4}{3}\epsilon_0 c n_2$. The only value to set is the nonlinear coefficient n_2.

```
[kerr]
n2 = 20e-24
```

Note: The value of $\chi^{(3)}$ is multiplied by the value of `pressure` that is set in section `[medium]`.

2.7.2 [ionization] (optional)

Defining this section adds a rate of ionization which populates the electron density $\rho(r,t)$ at each step of the simulation according to

$$\frac{\partial \rho}{\partial t} = W(E(t))(\rho_{nt} - \rho)$$

where W is the rate of ionization and ρ_{nt} is the `density_of_neutrals`. This equation is solved using the exponential time differencing method

$$\rho(t+\Delta t) = \exp\left[-\int_t^{t+\Delta t} W(E(t'))dt'\right]\left\{\rho(t) + \frac{\Delta t}{2}\rho_{nt}W(E(t))\right\} + \frac{\Delta t}{2}\rho_{nt}W(E(t+\Delta t))$$

Defining `[ionization]` also adds nonlinear absorption

$$J_{\mathrm{NA}}(t) = \frac{W(E(t))}{I(t)} U_i \left(f\rho_{nt} - \rho(t)\right)\epsilon_0 c E(t)$$

where U_i is the `ionization_potential` and f is the fraction of neutrals that can be ionized (`ionizing_fraction`).

Defining `[ionization]` also adds effects due to plasma according to

$$\frac{\partial J_{\mathrm{PL}}}{\partial t} + \frac{J_{\mathrm{PL}}(t)}{\tau_c} = \frac{e^2}{2m_e}\rho(t)E(t)$$

where τ_c is the `collision_time`. This equation is solved using the exponential time differencing method:

$$J_{\mathrm{PL}}(t+\Delta t) = e^{-\Delta t/\tau_c}\left\{J_{\mathrm{PL}}(t) + \frac{\Delta t e^2}{2m_e}\rho(t)E(t)\right\} + \frac{\Delta t e^2}{2m_e}\rho(t+\Delta t)E(t+\Delta t)$$

```
[ionization]
filename = ../../data/ionization-argon.txt
ionizing_fraction = 1
density_of_neutrals = 2.5e25
collision_time = 190e-15
ionization_potential = 2.5250303e-18
```

2.7.3 [quantum] (optional)

Experimental, quantum argon based on Ken's length gauge model. When adding this section its necessary to compile the program with support for MPI.

3 Extending

3.1 Adding new linear medium

3.2 Adding new nonlinearities