3+1D GNLSE Solver Documentation

This code is currently suitable for forward simulation of pulses and CW propagation. **While the code is autodifferentiable (written in JAX), and theoretically can be plugged into optimizers, this current distribution of the code contains no additional infrastructure for doing so.** This code solves a version of the Unidirectional Pulse Propagation Equation (UPPE), a (3+1)D Generalized Nonlinear Schrödinger Equation (GNLSE):

$$\begin{aligned}\frac{\partial A(x,y,z,\omega)}{\partial z} =& \frac{1}{2\beta_{eff}(\omega)}\nabla_\perp^2 A + \hat{D_\omega}(\omega)A + i\frac{\beta_{eff}(\omega)}{2}((\frac{n(x,y,\omega)}{n_{eff}(\omega)})^2-1)A \\ &+ \mathcal{F}_t\{i\frac{n_2\omega_0}{c}(1+\frac{i}{\omega_0}\frac{\partial}{\partial t})\{(1-f_R)|A|^2A + f_R A[h*|A|^2]\}\}\end{aligned}$$

The RHS terms describe, from left to right, *diffraction*, *dispersion*, *waveguiding*, and finally nonlinear effects including *Kerr nonlinearity* and *Raman scattering* (tuned between with the *Raman fraction* $f_R \in [0,1]$), and self-steepening. Other mechanisms implemented in code (but not indicated in the above equation) include *gain*, a *saturation intensity*, and perfectly matched layers (PMLs) that prevent undesired back-reflections at simulation boundaries.

Waveguiding, diffraction, dispersion, Kerr nonlinearity, and PMLs have been tested and appear to operate reliably; other effects (e.g., self-steepening, Raman scattering, gain, etc.) have not been thoroughly tested--use at your own risk (or else report anything fishy to me).

Quick Guide to Turning off Effects:

Setting arguments/flags to exactly these values not only ensures they have zero magnitude, but also signals to the code to completely skip those calculations (and speed up sim) where possible.

- Raman Scattering off: turn $f_R = 0.0$.

To construct and run a simulation requires specification of the following:

- Source: An initial field of the form $A(x,y,t)$, launched at the $z=0$ plane of the propagation medium.
- Medium: defined by the refractive index profile $n(x,y,\omega)$, nonlinear coefficient n_2, and material dispersion parameters (β_1, β_2). (Note: β_0 is determined by the source wavelength and the refractive index.) Additional effects might include Raman response (parameterized by t_1 and t_2, turned "off" by setting), self-steepening effects

gnlse_solver.py: Runs the simulation. The only user-facing function should be

- `GNLSE3D_propagate(args, A0, *, event_fn=None, event_payload=None, stop_on_event=True, event_check_every: int=1)` $\rightarrow$ `dict(field, dt, dx, seconds, **meta)`
  - `args` : a dictionary containing all physical information about the simulation, including:
    - `"Lx", "Ly", "Lz", "Lt"` : Physical dimensions $[m]$, $[s]$.
    - `"Nx", "Ny", "Nt"` : Grid sizes, ints.
    - `"deltaZ", "deltaZ_NL"` : Linear and nonlinear step-sizes $[m]$.
    - `"save_at"` : Array of z-locations to save $A(x, y, t)$.
    - `"lambda0"` : Central wavelength of pulse or CW source in free space $[m^{-1}]$.
    - `"n2"` : Nonlinear coefficient
    - `"beta0", "beta1", "beta2"` : Dispersion coefficients.
    - `"gain_coeff", "gain_fwhm"` : Gain parameters (set coeff to 0.0 to turn off).
    - `"t1", "t2"` :
    - `"n_xyomega"` : Refractive index profile, specified for each frequency.
    - `"pml_thickness"` : depth of PMLs, in $[m]$.
    - `"pml_Wmax"` : maximum damping exponent of the PML, in $[W/m]$.
    - `"m_nl_substeps"` : Number of substeps used in the nonlinear step of integration.
  - `"A0"` : The initial field at $z =$, $A(x, y, t)$. Has dimensions `Nx` $\times$ `Ny` $\times$ `Nz` .

gnlse_medium.py: Contains functions for creating the refractive index profile $n(x, y, \omega)$.

Refractive index profiles:

- `make_space(Lx, Nx, Ly, Ny)` $\rightarrow$ `X, Y`
  - `Lx` , `Ly` : spatial x/y dimensions of sim.
  - `Nx` , `Ny` : spatial x/y grid sizes.
  - `X` , `Y` : arrays of physical distances along axes.
  - dim(`X`)= `Nx` , dim(`Y`)= `Ny` .
  - Use the outputs of make_space as inputs `X, Y` of the below functions.
- `make_polynomial_n(X, Y, n_core, n_clad, r_core, alpha = 2)` $\rightarrow$ n(x, y):
  - Returns $n(x, y) = n_{\text{core}}\sqrt{1 - 2\delta[\frac{x^2+y^2}{r_{\text{core}}}]^{\alpha}}$ inside core radius, and n_{clad} without.
  - $n(x, y)$ is an $N_x \times N_y$ dimensional array.
  - $\delta = \frac{n_{\text{core}} - n_{\text{clad}}}{n_{\text{core}}}$
  - E.g., use $\alpha = 2$ for GRIN fiber.

- `make_supergauss_n(X, Y, n_core, n_clad, r_core, m=20)` $\rightarrow n(x, y)$
  - $n(x, y) = 1 +$ -
- `make_bulk_n(X, Y, n)` $\rightarrow n(x, y)$: Unwritten, will wrap `n * jnp.ones((len(X),len(Y)))`
- `bend_n(n_xy, Lz, a)` $\rightarrow n(x, y, z)$: Unwritten
- `gaussian_disorder_n`: Unwritten.

gnlse_source.py: Contains functions for building transverse and temporal field profiles.

Generating Modes:

- f
- Accessing Modes:

Generating Temporal Profiles:

gnlse_visualizations.py: Contains functions for plotting simulation outputs. These are heavily GPT'ed since they're pretty trivial matplotlib tasks. Presumably the user will DIY most of their own plotting.

gnlse_events.py: Contains functions for implementing event handling in simulations (e.g., halting simulation at onset of self-focusing collapse).

Ziyu recommended a Basler ace camera (this one seems suitable: https://www.baslerweb.com/en/shop/aca3088-57um/) and a pco.edge 4.2 camera, if we want higher resolution or lower noise: https://www.excelitas.com/product/pcoedge-42-usb-scmos-camera. **I think the Basler camera seems adequate for what we're doing, but we can discuss.** The Basler ace A3088-57um has:

- 7.41 mm x 4.95 mm sensor area.
- 3088 x 2064 (6 MP) resolution.
- 2.4 micron x 2.4 micron pixel size.
- 59 fps framerate.
  Ziyu said he's using a 600 x 800 pixel Basler, which I think has higher framerate, but (discussed below), I'm not sure (a) we need a framerate higher than what we expect the SLM's to be and (b) I think we have a premium on spatial resolution since we're looking at fine features of the transverse field.

My intuition for specs is probably a little skewed, since my experience is with applications requiring single-photon resolution. That being said, my assumptions for our use-case are as follows:

- Pixel pitch: two options for reference are either the scale of the SLM pixels (which, I think, are ~ 10s of microns), or (more demandingly) the order of 2-4 times the grid used in simulation (since the pixel-scale features in simulation are somewhat unreliable anyway). So I think higher spatial resolution is actually something to prioritize here, especially as some of our optimization (or Novelty Search) targets may involve "shrinking" feature sizes.
- Quantum efficiency of visible spectrum cameras at 1064 nm is nonzero, but low--but I can't imagine we are at a shortage of photons here.
- I do not have a good sense of how noise works for these high power cases; I don't think we're doing anything so subtle that I can imagine it being a big consideration here, but the pco.edge 4.2 is an option if it's important.
- Assuming our worst-case-scenario output profile dimensions are simply those of ~10 cm linear propagation (diffraction) of a 3 mm diameter, 1064 nm beam, the broadening is effectively negligible. Any sensor of dimensions >= 3 mm by 3mm should be fine--probably at least 5mm by 5mm (without sacrificing resolution) would make life easier.
- The nonlinear propagation distance required will depend on the wedge/block material. Dongjin and I are going to run some bulk media propagation demos for his presentation, so we can double check this 10 cm estimate quickly. However, I strongly doubt it will change this beam-broadening estimate drastically.
- Unless I am misunderstanding, I don't think sub-pulse duration-resolving framerates are reasonable or expected here. I imagine our intention is to send one-shape-of-pulse-per-frame, limited by the slower of either the framerate or SLM speed. SLM framerate (from Thorlabs, as reference) is ~ 60 Hz. So we can probably get away with a low framerate on the camera. It seems like we'd want r_{slm} >= r_{cam} anyway, for things to be easily synchronized.