Suggested tasks for Steven

J. F. Myatt
15 June 2020

15 June 2020

Steven, I've made the Matlab program more useable. I've also reorganized it a bit. The script *main_program.m* now does most of the work. Here is a description of what it does:

Rough description of the code operation

I've reorganized a bit. I put the functions that deal with plotting in their own directory (./Plotting), while the rest is in the ./Source directory. This is why I add these paths at the beginning of the script.

I'm keeping a bunch of definitions as globals that are initialized with *initCnst*. I'm happy with this and more things can be added as needed.

I'm trying to come up with a good way of dealing with plotting. For now, I have a "plot list" that is controlled by a bunch of flags. These can be initialized to defaults (using *initDefaultPlots*) or set separately. There is also a function called *clearPlotList* that removes everything. There is then a function called *makePlotList* that will plot all the things that have been set on the plot list. Since this doesn't cover everything, there are other functions such as *addBundlePlt* and *addContourPlt* that can modify the above by overplotting ray trajectories and contours respectively. Take a look at these and see how it works. We might want to change this at some point, but it seems o.k. for now.

I haven't changed the way that the hydrodynamic profiles are loaded. As you know the *importDracoGrid* function returns a *rayGd* structure that is required for most of the plotting and ray integration. One thing that I would like to do is to write another function (*importAnalyticGrid*?) that returns a *rayGd* structure, but based on analytic plasma profiles that the user selects (e.g., a density profile that is linear). This could be useful for checking our integrators as we add more functionality such as computing the intensity and absorption along a ray. I don't want you to work on this yet though.

Currently there are two main ways of creating bundles of rays (ray bundles are structures that seem useful and I think we should continue to use them). These are via a *launchList* that is passed as an argument to the *makeRayBundle* function or the *getRamanWaveVectors* function. I'm thinking of changing the name of this second function to be *getRamanBundles* as it is a more accurate description of what it does.

- The use of *makeRayBundle* is to launch rays that correspond to incident beams. This includes the laser drive beams, detected light (that is integrated backwards in time). In the future it could include Thomson scattering probe beams or detected Thomson scattered light.
- The use of *getRamanWaveVectors/getRamanBundles* is to return ray bundles corresponding to the daughter productions of stimulated Raman scattering events along a previously defined EM ray. I think I like the way this works. We could write similar functions for two-plasmon decay and stimulated Brillouin scattering events.

The nice thing about ray bundles is that they can all be integrated in time (i.e., pushed) with the same function (*pushBundle*). It knows what type of wave it is dealing with because the *type* (EM/EPW) descriptor is part of the bundle.

This function works o.k. (ish). There is still occasionally a problem with integrating outside of the domain where the hydro is defined. I'll fix this at some point. You might also want to try to figure it out. It's not too bad though.

We are at a point where the above stuff works, so I suggest we use it a little bit before we go on to add more functionality. I think the act of using it will give us better insight into how we should proceed with development.

Suggestions for first use of the code

1. Run the example given in *main_program.m* and understand what it is doing. You will see that the SRS light along a particular incident ray corresponding to perfect backscatter only retraces the original ray if the scattering is at low density. For longer wavelength backscatter (e.g. 670 nm) it exits at a very different angle (close to the direction anti-parallel to the density gradient.
2. For the case of two beams incident at $\pm 23^\circ$ figure out which rays fall into the detector. In practice this means that the Raman must exit with an angle that is within $\pm 3^\circ$ of the angle to the incident beam. Do this for one particular hydro time slice for now. You should be able to make a histogram of number of rays in a given frequency bin hitting the detector. You'll need to write some functions to help you. I can assist after you've had a good think about it.
3. Once you've completed #2 we'll need to discuss because the next step will be to automate all of the above so that we can look at SRS events that are not just backscatter but include several angles between forward and backscatter (using the changes that you make to *getRamanWavevectors*. We'd also like to do this for all the time slices so that we can make a simulated streaked spectrum.

June 18, 2020

The following changes were made:

- Two new functions (*setHaltAll* and *resetHaltAll*) have been written that help with ray integrations. They prevent (allow) pushBundle to move the rays.
- I fixed *makeRayBundle*. When using the *LaunchList* it improperly rotated the initial points. Quite embarassing. It now uses atan2() which picks up the proper quandrant from the centroid vector.
- I made some changes to *main_program* when working on the above.
  - note that in the previous main program the initial push bundle call left too few trajectory points in the interesting Raman region. I modified this - as you will have to do also (see the new *main_program*)
- There is a new driver program called *ramanMovie*. I think it could be useful for you to have a look at it. I loop over backscatter one ray at a time and make a movie. You need to do this and then score the results on the detector
- There is a .avi file which is obtained from the above on using the new script *makeAVI*