Global Particle Acceleration and Transport (GPAT) Model

Release 0.2

Xiaocan Li

Apr 12, 2024

INSTALLATION

GPAT (Global Particle Acceleration and Transport) is a program for solving particle transport equations. It solves either Parker's transport equation for nearly isotropic particle distributions or the focused transport equations for anisotropic particle distributions. The code uses the stochastic integration method to solve these equations. For details on the algorithms implemented, please check out the theory section.

CHAPTER

ONE

CONTACT US

If you're beginning to use GPAT or have any questions, feel free to drop by the discussions page. For bug reports or to request new features, you can also open a new issue.

CHAPTER

TWO

INSTALL

2.1 Installation

This package uses a Multiple stream Mersenne Twister PRNG to generated the required random numbers. This package uses FLAP to deal with comand line arguments. The FLAP pacakge is recommended to install using the FoBiS.py, which is a building system for Fortran projects. Currently, the code can only be run on CPUs.

2.1.1 Requirments

- Programs that are commonly installed on HPC clusters
  - **Git**: for cloning the code from GitHub
  - **CMake**: 3.9.0 or higher
  - **GCC** or **Intel** Fortran compilers.
  - **MPI**: OpenMPI or MPICH. It should work with the others.
  - **HDF5**: parallel version.

Tip: On Perlmutter@NERSC, you can load the modules by `module load cpu cmake cray-hdf5-parallel`

- FoBiS.py: a building system for Fortran projects
  - The installation wiki: https://github.com/szaghi/FoBiS/wiki/Install.
  - It is recommended to use PyPI to install it: `pip install FoBiS.py --user`
  - After installing, make sure that `FoBiS.py` is in your `$PATH` by including

    ```
    export PATH=$HOME/.local/bin/:$PATH
    ```

    in your `.bashrc`.
- Multiple stream Mersenne Twister PRNG: for generating pseudorandom numbers parallelly.

  This program module splits single long period random number series from Mersenne Twister (MT) into multiple (almost) independent streams.
  - Download: http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/PRNG/mt_stream_f90.tar.gz
  - Load the same packages as listed above.

```
tar zxvf mt_stream_f90.tar.gz
cd mt_stream_f90-1.11
```

Depending on the system type, you might need to modify `FC` in `Makefile` by hand. The default is *ifort*, which is for the Intel compilers. For *gfortran*, please use the script `config/gfortran_mt_stream.sh` to make the relevant changes in `Makefile`. First, copy the script to `mt_stream_f90-1.11`. Then,

```
./gfortran_mt_stream.sh
```

Note: If you are using newer versions of *gfortran*, there will be compiling errors. Therefore, we need to update the source code slightly using `config/fix_mt_stream.sh`. First, copy the script to `mt_stream_f90-1.11`. Then,

```
./fix_mt_stream.sh
```

– You can then compile the code by

```
make
```

– After compiling the code, you need to set the environment variable `MT_STREAM` to the installation directory. For example,

```
export MT_STREAM=$HOME/local/mt_stream_f90-1.11
```

This environment variable will be used to compile the GPAT code.

Tip: You can put this into your `.bashrc`.

2.1.2 Download

```
git clone https://github.com/xiaocanli/stochastic-parker
```

2.1.3 Install

In the directory `stochastic-parker`,

```
mkdir build
cd build
cmake ..
make
make install
```

To turn on OpenMP parallelization, please use `cmake -DUSE_OPENMP="On" ...` To turn on `AVX512` for the KNL nodes, please also include `-DUSE_AVX512="On"`.

Tip: On Perlmutter@NERSC, you can load the modules by `module load cpu cmake cray-hdf5-parallel python`

Note: Please use the same version of `python` as when installing `FoBiS.py`. Otherwise, `FLAP` will not be compiled correctly.

Note: The code is not carefully optimized for the KNL nodes. Please use it in caution.

You can create a soft link of the executable `stochastic-mhd.exec` in the scratch filesystem for running the code.

CHAPTER

THREE

USAGE

3.1 Usage

3.1.1 Introduction

The code uses MHD simulation results to evolve the particle distribution functions according to the transport equations. The MHD quantities used include magnetic field, velocity field, and maybe plasma density.

Note: The plasma density might be used to calculate the local Alfvén speed.

In general, the modeling procedure includes

- Run the MHD simulations and get the MHD fields.
- Pre-process the MHD results to provide the input data for the GPAT model.
- Calculate the input parameters, including the spatial diffusion coefficient, parameters for evaluating the momentum diffusion coefficient, parameters for evaluating the particle drift, etc.
- Modify the configuration for the transport modeling.
- Run the simulation.
- Analyze and visualize the results.

Below, we will use a 2D magnetic reconnection problem to illustrate the procedure in more detail.

3.1.2 Example: 2D Magnetic Reconnection

Here, we use a 2D reconnection problem solved with Parker's transport equation as an example. The relevant code and scripts are in `examples/reconnection_2d`.

Note: More examples will be included in the future.

Run the MHD simulation

Please follow the instructions in athena_reconnection to run a reconnection simulation with two current sheets and periodic boundary conditions. The input file for the Athena++ simulation is `code/athinput.reconnection` in athena_reconnection. The simulation domain size is [0, 2] $\times$ [0,2]. The grid size is 1024×1024. The simulation will last 20 Alfven-crossing time and produce 200 frames of data of the primary variables (`reconnection.prim*.athdf`) in default.

Pre-process the MHD data

Instead of directly processing the data from different MHD codes (e.g., VTK files by Athena and HDF5 files by Athena++) in the simulations, we will reorganize the MHD simulation outputs using Python scripts first. The example scripts for Athena and Athena++ outputs are in `mhd_data`. There is a shell script `reorganize_data.sh` for running the Python scripts with commandline arguments.

```
./reorganize_data.sh
```

which will include 2 ghost cells at each boundary according to the boundary conditions (periodic or open).

Note: Due to their stochastic nature, the pseudo particles can cross the local domain boundaries multiple times in a short time, significantly increasing the MPI communication cost. We choose two ghost cells to enable particles to stay in the local domain when they are near the boundaries. The two ghost cells also make calculating the gradients of the fields easier at the boundaries. **This could be improved in the future**.

The script will generate the MHD fields (in bindary format) and the simulation configuration needed for the transport modelings in `bin_data` under the MHD run directory.

- `mhd_data_*`: $v_x, v_y, v_z, \rho, B_x, B_y, B_z, B$, which have a size of 1028×1028.
- `rho_*`: plasma density (1028×1028)
- `pre_*`: plasma pressure (1028×1028)
- `mhd_config.dat`: the MHD configuration information. See the function `save_mhd_config` in `mhd_data/reorganize_fields.py` for the details of the configuration.

Calculate the input parameters

Before transport modeling, we need to calculate the parallel diffusion coefficient $\kappa_{\parallel}$ using the simulation normalizations (length scale, magnetic field, plasma density, turbulence amplitude and anisotropy). There is a Python script `sde.py` for doing that. The default parameters are ideal and similar to those used in[1]. Running the script will give a set of parameters. We will only need `Normed kappa parallel` for now, which is 7.43592e-03 in default.

Note: The script needs `plasmapy` for the particle properties (e.g., mass and charge) and the calculation of different plasma parameters.

---

[1] Large-scale Compression Acceleration during Magnetic Reconnection in a Low- Plasma, Xiaocan Li, Fan Guo, Hui Li, and Shengtai Li, *The Astrophysical Journal* Oct 2018

Particle transport modeling

- Go the the directory (e.g., `$SCRATCH/transport_test`) where you link the executable `stochastic-mhd.exec`.
- Create a directory `config` under `transport_test` and copy `conf.dat` and `diffusion_reconnection.sh` in `stochastic-parker/examples/reconnection_2d` into `config`.

Note: The input parameters are given through two files: `conf.dat` and `diffusion_reconnection.sh`. You will notice that there are many input parameters, some of which are through `conf.dat`, and the others are through command-line arguments. The parameters in `conf.dat` are read by the code; the parameters in `diffusion_reconneciton.sh` are used as the command-line arguments of the main program `stochastic-mhd.exec`. The command-line arguments are from the earlier design. As the program evolves, more and more command-line arguments are needed, making it tedious to run the simulation. **In the future, it will be better to put some of the command-line arguments into a configuration file (e.g., conf.dat).**

- Change `kpara0` in function `stochastic ()` in `diffusion_reconnection.sh` to the value calculated above.
- Change `mhd_run_dir` and `run_name` (at the bottom of `diffusion_reconnection.sh`) to your choices.

To check all the available command-line arguments,

```
srun -n 1 ./stochastic-mhd.exec -h
```

Or you can check the comments in `diffusion_reconnection.sh`.

Note: Now, the input paramters are described at *Input Parameters*.

For this test run, you don't need to change these input parameters. The default name of the transport run is `transport_test_run`. We can request an interactive node to run the test, for example, on Perlmutter@NERSC,

```
salloc --nodes 1 --qos interactive --time 04:00:00 --constraint cpu --account=m4054
module load cpu cray-hdf5-parallel
./diffusion_reconnection.sh
```

It will take about one hour to run the simulation. The output files are in `data/athena_reconnection_test/transport_test_run`.

Warning: The binary outputs of the particle distributions have been deprecated. The new outputs are saved in the same HDF5 file `fdists_****.h5`. The corresponding analysis scripts need to be updated to read the HDF5 outputs.

- `fdpdt-*.dat`: the energization rate (only the compression is included for now).
- `fp-*.dat`: the global momentum distributions.
- `fp_local-*.dat`: the local momentum distributions.
- `fxy-*.dat`: the local particle densities in different energy bands. They are similar to `fp_local-*.dat` but only for a few energy bands.

The outputs include

- `quick.dat`: the parameters to diagnose the simulation status during runtime.

- `pmax_global.dat`: the maximum momentum at the same time frames in `quick.dat`.
- `fdists_****.h5`: particle distributions, including the global distribution, local distributions, and the relevant bins. The dimensions of the distributions are controlled in the input file `conf.dat`: `npp_global`, `nmu_global`, `dump_interval*`, `pmin*`, `pmax**`, `npbins*`, `nmu*`, `rx*`, `ry*`, `rz*`. When running Parker's transport, `nmu_global` and `nmu*` defaults to 1.

`fdists_****.h5` contains the following quantities.

- `fglobal`: global particle distribution with a size of [`npp_global`, `nmu_global`]
- `pbins_edges_global`: global momentum bins edges with a size of [`npp_global`+1]
- `mubins_edges_global`: global cosine of pitch-angle bins edges with a size of [`nmu_global`+1]
- `flocal*`: local particle distributions with a size of [`nz_mhd/rx*`, `ny_mhd/ry*`, `nz_mhd/rz*`, `npbins*`, `nmu*`], where `nx_mhd`, `ny_mhd`, and `nz_mhd` are the dimensions of the MHD simulation. In this 2D example, `nz_mhd=1`, and the first dimension of `flocal*` is 1.
- `pbins_edges*`: local momentum bins edges with a size of [`npbins*`+1]
- `mubins_edges*`: local cosine of pitch-angle bins edges with a size of [`nmu*`+1]

Visualize the results

The relevant files for plotting are in `examples/reconnection_2d/vis`. Please copy the files to a directory of your choice for data analysis, for example, `$SCRATCH/transport_test/python`. Please also copy `python/sde_util.py` into the same directory. We will use the Jupyter notebook `transport_test.ipynb` to plot the results. The notebook needs information about the MHD simulation (`mhd_runs_for_sde.json`) and the SDE run (`spectrum_config.json`).

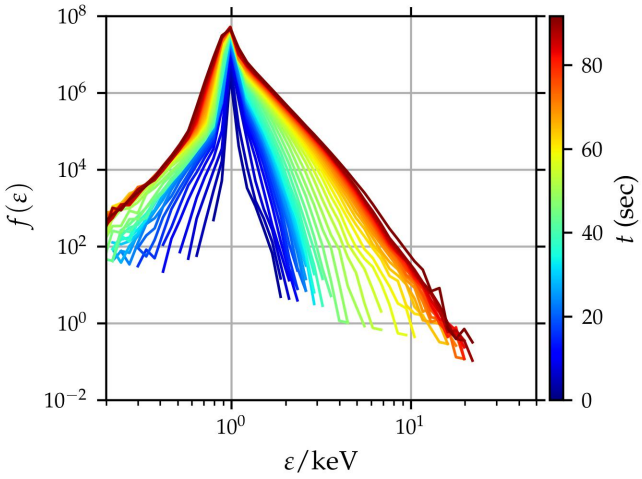
Note: The two JSON files will keep tracking the information of MHD runs and the SED runs for each MHD simulation, respectively. We recommend keeping the records in this kind of JSON file.

For `mhd_runs_for_sde.json`, please change `run_dir` to your reocnnection simulation directory. In `spectrum_config.json`,

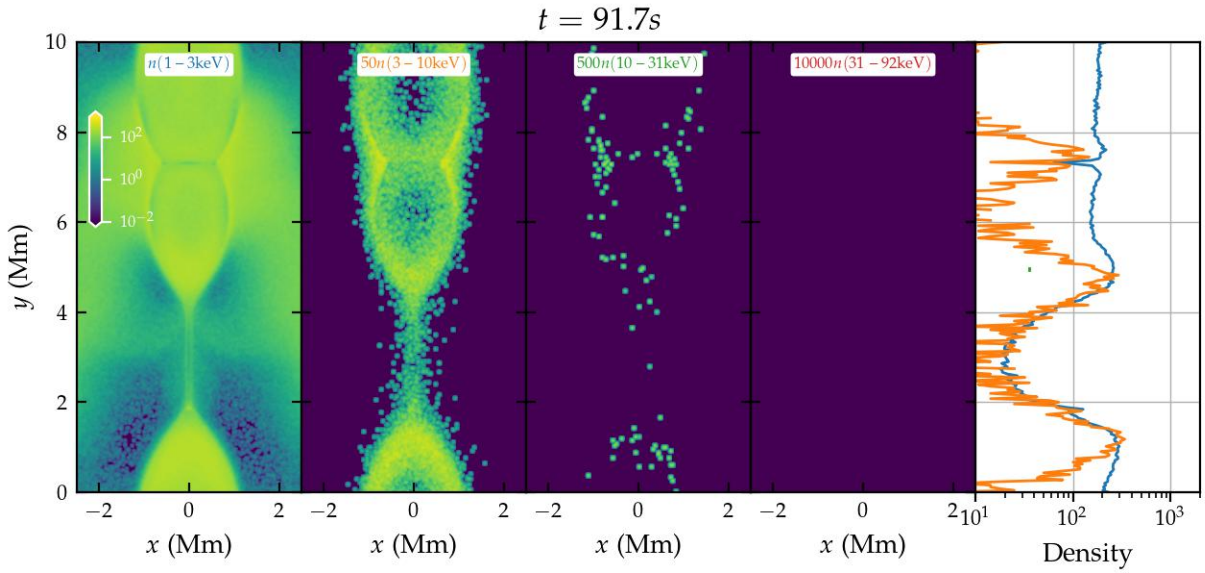
- `run_name`: a unique name for the SDE run starting from the MHD run name
- `e0`: the energy normalization in keV (default: 10 keV)
- `xlim_e`, `ylim_e`: the limites for the energy spectrum plots

The rest of the parameters in `spectrum_config.json` are not commonly used and will be deprecated in the future.

After running the Jupyter Notebook, you will get the time evolution of the global energy spectrum shown below. The different colors indicate the spectra at different time frames.



We can also get the spatial distributions of local particle distributions at the final time step. The four left panels show the electron distributions in four energy bands with corresponding scaling factors. The rightmost panel shows the vertical through the center of the reconnection region at $x = 0$.



Both the spectra and the spatial distributions show that the acceleration is weak. The reason is that the compression in the MHD simulation is not strong. The acceleration will be stronger in MHD simulations with higher Lundquist numbers and resolutions.

Note: The notebook includes the script to read the global and local particle distributions. You can perform further analysis of the particle distributions.

3.1.3 What are not included in this description?

The code currently supports many functionalities that are not included here. Those will be gradually included in future updates of the documentation, for example,

- Different particle injection methods (e.g., spatially dependent injection)
- Particle-splitting techniques
- Particle tracking
- Momentum diffusion
- Spatially dependent turbulence properties
- 3D simulations
- Spherical coordinates

3.2 Input Parameters

The input parameters are mostly in script `diffusion*.sh` (e.g., `diffusion_reconnection.sh` for the 2D reconnection example). A few others are in `conf*.dat` (e.g., `conf.dat` for the 2D reconnection example). The input parameters are explained below.

3.2.1 diffusion.sh

- `MACHINE`: either `Frontera` or `Perlmutter`. `Perlmutter` is the default.
- `quota_hour`: the maximum wall time in hours. The code dump restart files if the simulation is not finished when the wall time is reached (30 mins before that in the code). The restart files are saved in the `restart` sub-directory under the diagnostic directory. These are data files `particles_*.h5` for particle data, `mtss.mpi_rank.thread_id` for the state of the random number generators, `particle_module_stare_*.h5` for the state of the particle module, and `latest_restart` for the last time frame before duming restart files.
- `restart_flag`: whether the simulation is a restart from previous simulations. `.false.` is the default, which means it is not a restart. `.true.` means it is a restart. When it is a restart, the code will read the particle data and `random_number_generater` from the previous simulation and continue the simulation from the last time step.
- `conf`: the name of the configuration file. `conf.dat` is the default.
- `mhd_config_filename`: the name of the MHD configuration file. `mhd_config.dat` is the default.
- `focused_transport`: whether to solve the focused transport equation. `.false.` is the default, which means solving the Parker transport equation.
- `mpi_size`: the total number of MPI processes.
- `ntasks_per_node`: the number of MPI processes per node.
- `size_mpi_sub`: the size of an MPI sub-communicator. `1` is the default, which means there is only one MPI communicator (MPI_COMM_WORLD). If not `1`, the total number of MPI processes must be divisible by `size_mpi_sub`. The MPI communications are done within each sub-communicator. In this way, we don't need to divide the domain into too many sub-domains, which is not efficient for MPI communications. To get the global properties (e.g., global particle spectrum), we will use the cross-communicators.
- `nptl`: the total number of particles for one MPI process.
- `nptl_max`: the maximum number of particles for one MPI process. If the number of particles exceeds `nptl_max`, we will stop injecting more particles or splitting particles. Typically, we can choose up to 10 million.

- `ts`: the start time step.
- `te`: the end time step.
- `tmax_mhd`: the maximum time frame for the MHD fields. Typically, `tmax_mhd` is larger than `te`, so the transport modeling will have enough MHD fields to use. When `tmax_mhd` is smaller than `te`, there are two cases: 1) when `single_time_frame=1` (discussed below), the code will keep running until `te`; 2) when `single_time_frame=0`, the code will stop reading MHD fields after `tmax_mhd` and use the last MHD fields for the rest of the simulation.
- `single_time_frame`: whether to use only one-time frame. `0` is the default, which means using multiple time frames (from `ts` to `te`). `1` means using only one time frame (`ts`). Then, the simulation will run `te-ts+1` time steps using the same MHD fields.
- `time_interp`: whether to interpolate in-between time frames. `1` is the default, which means interpolation. `0` means no interpolation.
- `dist_flag`: the initial particle distribution. `0` is the default, which means Maxwellian. `1` means delta function. `2` means power-law.
- `power_index`: the power-law index for the initial momentum distribution if `dist_flag=2`.
- `split_flag`: whether to split particles when they reach certain momentum. `0` is the default, which means no splitting. `1` means splitting.
- `split_ratio`: the momentum increase ratio for particle splitting. Assuming the particle's initial momentum is `p0`, the particle will be splitted into two particles once the particle momentum reaches `p0*split_ratio`. The rul of thumb is to set `split_ratio` to be around $2^{1/\gamma}$, where γ is the power-law index for the particle momentum distribution.
- `pmin_split`: the minimum momentum (in `p0`) to start splitting particles. Instead of starting from `p0`, we can start from a higher momentum to avoid splitting too many particles at low momentum. It is particularly important when injecting Maxwellian particles, where the number of particles is large at low momentum.
- `local_dist`: whether to diagnose local particle distribution.
- `dump_escaped_dist`: whether to dump the escaped particle distributions, either `.true` or `.false`. If `.false.`, the particles will be removed once they escape from the boundaries. Otherwise, the escaped particles will be accumulated in one MHD data interval. The distributions of the escaped particles are stored in files `escaped_dist_*.h5`. They have the same grid sizes as the local particle distributions but with one dimension less. For example, if we have a 3D simulation, the escaped particle distributions will be 2D for the low-x, high-x, low-y, high-y, low-z, and high-z boundaries.

Warning: The escaped particle distributions are only dumped when `local_dist=.true.`. When `local_dist=.false.`, the escaped particle distributions will not be dumped event if `dump_escaped_dist=.true.`.

- `dump_escaped`: whether to dump the escaped particles. `dump_escaped_dist` has to be `.true.` for this to work. If `dump_escaped_dist=.false.`, the raw data of the escaped particles will be dumped. Otherwise, only the distributions of the escaped particles will be dumped. Note that the raw data of the escaped particles can be large. It is not recommended to dump the raw data of the escaped particles unless you want to analyze the raw data in detail.
- `track_particle_flag`: whether to track particles. `.false.` is the default, which means not tracking particles. `.true.` means tracking particles. When tracking particles, we will select `nptl_selected` particles with the highest energy, rerun the simulation, and output the trajectories of these particles.
- `particle_tags_file`: the file name for the particle tags. It is used to select the particles to track. The file `tags_selected_*.h5` has one dataset `tags`. The dataset is a 1D array of integers. These tags can be obtained from analyzing the particle data. For example, we can select the particles with the highest energy or particles in

certain regions and output their tags to the file `tags_selected_*.h5`. The file is located in the same directory as the diagnostic directory.

- `nsteps_interval`: the steps interval to track particles. If it is `1`, we will track particles at every time step. If it is `10`, we will track particles at every 10 steps. Particles can be evolves for millions of time steps. It is not recommended to track particles at every time step unless you want to analyze the particle trajectories in great detail.
- `inject_new_ptl`: whether to inject new particles at every step. The values are `.true.` or `.false.`. `.true.` means injecting new particles at every MHD step. `.false.` means only injecting particles at the beginning.
- `inject_same_nptl`: whether to inject the same number of particles every step. `.true.` is the default, which means injecting the same number of particles. `.false.` means injecting a different number of particles every step. Then, the number of injected particles depends on how particles are injected (see below). `inject_same_nptl` only works when `inject_new_ptl=.true.`.
- `tmax_to_inject`: the maximum time frame to inject particles. It is used to stop injecting particles after some time. It is useful, for example, when we want to run simulations for a long time but only inject particles at the beginning. Note that `tmax_to_inject` should be larger than `ts`. Note that it only works when `inject_new_ptl=.true.`.
- `inject_part_box`: whether to inject particles in part of the simulation box. `.false.` is the default, which means injecting particles in the whole box. `.true.` means injecting particles in part of the box.
- `ptl_xmin`: the minimum x coordinate for particle injection. It is the absolute coordinate.
- `ptl_xmax`: the maximum x coordinate for particle injection.
- `ptl_ymin`: the minimum y coordinate for particle injection.
- `ptl_ymax`: the maximum y coordinate for particle injection.
- `ptl_zmin`: the minimum z coordinate for particle injection.
- `ptl_zmax`: the maximum z coordinate for particle injection.
- `inject_large_jz`: whether to inject particles where jz is large. `.false.` is the default, which means not injecting particles where jz is large. `.true.` means injecting particles where jz is large.
- `jz_min`: the minimum jz for injection when `inject_large_jz=.true.`. It is to make sure that the particles are only injected in regions with large jz. It is useful for reconnection simulations, where we might want to inject particles in the reconnection region. Depending on the simulation, we might need to adjust this value because each simulation has a different jz range or different normalizations. Note that `inject_large_jz=.true.`, `inject_large_absj=.true.`, `inject_large_db2=.true.`, `inject_large_divv=.true.`, and `inject_large_rho=.true.` are exclusive. Only one of them can be `.true.`.
- `ncells_large_jz_norm`: the normalization for the number of cells with large jz. It is used to determine the number of particles to inject. The number of particles to inject is `nptl*ncells(jz>jz_min)/ncells_large_jz_norm`, where `ncells(jz>jz_min)` is the number of cells with jz larger than `jz_min`.
- `inject_large_absj`: whether to inject particles where **|j|** is large. `.false.` is the default, which means not injecting particles where **|j|** is large. `.true.` means injecting particles where **|j|** is large.
- `absj_min`: the minimum **|j|** for injection when `inject_large_absj=.true.`. It is to make sure that the particles are only injected in regions with large **|j|**.
- `ncells_large_absj_norm`: the normalization for the number of cells with large **|j|**. It is used to determine the number of particles to inject. The number of particles to inject is `nptl*ncells(|j|>absj_min)/ncells_large_absj_norm`, where `ncells(|j|>absj_min)` is the number of cells with **|j|** larger than absj_min.
- `inject_large_db2`: whether to inject particles where turbulence amplitude (db2) is large. `.false.` is the default, which means not injecting particles where db2 is large. `.true.` means injecting particles where db2 is

large. When `inject_large_db2=.true.`, the code needs to read an additional file `deltab*.dat` to get the turbulence amplitude.

- `db2_min`: the minimum db2 for injection when `inject_large_db2=.true.`. It is to make sure that the particles are only injected in regions with large db2. It is useful for reconnection simulations, where we might want to inject particles in the reconnection region, where reconnection-driven turbulence can be intense.
- `ncells_large_db2_norm`: the normalization for the number of cells with large db2. It is used to determine the number of particles to inject. The number of particles to inject is `nptl*ncells(db2>db2_min)/ncells_large_db2_norm`, where `ncells(db2>db2_min)` is the number of cells with db2 larger than `db2_min`.

Note: The functionality of `inject_large_db2` is not fully tested. It is not recommended to use it for now. Additionally, we need to understand spatially dependent turbulence amplitude better.

- `inject_large_divv`: whether to inject particles where flow compression divv is negatively large. `.false.` is the default, which means not injecting particles where divv is negatively large. `.true.` means injecting particles where divv is negatively large.
- `divv_min`: the minimum divv for injection when `inject_large_divv=.true.`. This is to make sure that the particles are only injected in regions with large volumes. It is useful for reconnection or shock simulations, where we might want to inject particles near regions with strong flow compression.
- `ncells_large_divv_norm`: the normalization for the number of cells with large `divv`. It is used to determine the number of particles to inject. The number of particles to inject is `nptl*ncells(|divv|>|divv_min|)/ncells_large_divv_norm`, where `ncells(|div|>|divv_min|)` is the number of cells with `|divv|` larger than `|divv_min|`.
- `inject_large_rho`: whether to inject particles where density is large. `.false.` is the default, which means not injecting particles where density is large. `.true.` means injecting particles where density is large.
- `rho_min`: the minimum density for injection when `inject_large_rho=.true.`. It is to make sure that the particles are only injected in regions with large density.
- `ncells_large_rho_norm`: the normalization for the number of cells with large density. It is used to determine the number of particles to inject. The number of particles to inject is `nptl*ncells(rho>rho_min)/ncells_large_rho_norm`, where `ncells(rho>rho_min)` is the number of cells with density larger than `rho_min`.
- `dpp_wave`: whether to include momentum diffusion due to wave scattering. `0` is the default, which means not including momentum diffusion due to wave scattering. `1` means including momentum diffusion due to wave scattering.
- `dpp_shear`: whether to include momentum diffusion due to flow shear. `0` is the default, which means not including momentum diffusion due to flow shear. `1` means including momentum diffusion due to flow shear.
- `weak_scattering`: whether particle scattering is in the weak-scattering regime. `1` is the default, which means the weak-scattering regime. `0` means the strong-scattering regime.
- `deltab_flag`: whether to have spatially dependent turbulence amplitude. `0` is the default, which means that the turbulence amplitude is spatially uniform. `1` means having spatially dependent turbulence amplitude. When `deltab_flag=1`, the code needs to read an additional file `deltab*.dat` to get the turbulence amplitude.
- `correlation_flag`: whether to have spatially dependent turbulence correlation length. `0` is the default, which means that the turbulence correlation length is spatially uniform. `1` means having spatially dependent turbulence correlation length. When `correlation_flag=1`, the code needs to read an additional file `lc*.dat` to get the turbulence correlation length.

Note: The functionalities of `deltab_flag` and `corrlation_flag` are not fully tested. It is not recommended to use them for now. Additionally, we need to understand spatially dependent turbulence amplitude and correlation length better.

- `ndim_field`: the dimension of the field. The values can be `1`, `2`, or `3`. `1` means 1D simulation, but it is not fully tested.
- `drift_param1`: the parameter 1 for particle drift. It is used to determine the drift velocity. See the Theory section for details.
- `drift_param2`: the parameter 2 for particle drift. It is used to determine the drift velocity. See the Theory section for details.
- `charge`: the charge of the particle in unit charge. `-1` is the default, which means electron.
- `spherical_coord`: whether the grid is spherical. `0` is the default, which means the grid is Cartesian. `1` means the grid is spherical.
- `uniform_grid`: whether the grid is uniform. `1` is the default, which means the grid is uniform. `0` means the grid is non-uniform. Then, we need to the coordinates of the grid points in files `xpos.dat`, `ypos.dat`, and `zpos.dat`, located in the same directory as the MHD configuration file.
- `check_drift_2d`: whether to check particle drift in 2D simulations. `0` is the default, which means not checking particle drift in 2D simulations. `1` means checking particle drift in 2D simulations. It is useful for 2D simulations, where we can check how much particles drift along the out-of-plane direction.
- `particle_data_dump`: whether to dump particle data. `0` is the default, which means not dumping particle data. `1` means dumping particle data. When dumping particle data, the code will output the particle data at every output time step. The particle data are stored in files `particles_*.h5`. Since particle data can be large, it is not recommended to dump particle data unless you want to analyze the particle data.
- `include_3rd_dim`: whether to include transport along the 3rd-dim in 2D simulations. `0` is the default, which means not including transport along the 3rd-dim in 2D simulations. `1` means including transport along the 3rd-dim in 2D simulations. It is useful for 2D simulations, where we can check how much particles transport along the out-of-plane direction.
- `acc_by_surface`: whether the acceleration region is separated by a surface. `0` is the default, which means the acceleration region is not separated by a surface. `1` means the acceleration region is separated by a surface. Then, we need to specify the surface file name and the normal direction of the surface to get the 2D surfaces that separate the acceleration region. The surface file name is specified by `surface_filename1` and `surface_filename2`. The normal direction of the surface is specified by `surface_norm1` and `surface_norm2`. The surface normal direction can be `+x`, `-x`, `+y`, `-y`, `+z`, or `-z`. It is useful when we want to selectively turn on/off particle acceleration in certain regions. For example, we can turn on particle acceleration only in the reconnection region or termination shock region in flare simulations. To get the 2D surfaces separating different acceleration regions, we need to write our own scripts to look into the MHD simulation data.

Note: The functionality of `acc_by_surface` is not fully tested. It is not recommended to use it for now. Example scripts to get the 2D surfaces separating different acceleration regions will be provided later.

- `surface2_existed`: whether the second surface exists. `0` is the default, which means the second surface does not exist. Then, we only have two regions. `1` means the second surface exists. Then, we will have acceleration regions separated by these two surfaces.
- `varying_dt_mhd`: whether the time interval for MHD fields is varying. `0` is the default, which means the time interval for MHD fields is uniform. `1` means the time interval for MHD fields is varying. It is useful when the MHD simulation has varying time intervals. For example, the MHD simulation might have a large time interval

at the beginning and a small time interval later. Then, we can use `varying_dt_mhd=1` to use the varying time interval for MHD fields. When `varying_dt_mhd=1`, the code needs to read an additional file `time_stamps.dat` to get the time stamps for each MHD frame. The file is located in the same directory as the MHD data files.

Then, the script will modify the configuration file `conf.dat`. The parameters in `conf.dat` are explained below. Additionally, a few other parameters are modified in this script for more flexibility.

- `tau0_scattering`: the scattering time for initial particles. It is only used for momentum diffusion due to wave scattering. It is not used for Parker transport. The parameters are calculated based on the initial particle momentum and turbulence properties in `sde.py`.
- `duu0`: the normalization for pitch-angle diffusion coefficient. It is only used in the focused transport equation. The parameters are calculated based on the initial particle momentum and turbulence properties in `sde.py`.
- `particle_v0`: the particle speed/velocity normalization. It is only used in the focused transport equation. The parameters are calculated based on the initial particle momentum and turbulence properties in `sde.py`.
- `dir_mhd_data`: the directory for MHD simulation data.
- `diagnostic_directory`: the directory for diagnostics data.

3.2.2 conf.dat

- `b0`: initial magnetic field strength (deprecated).
- `p0`: initial particle momentum. Its value is arbitrary. 0.1 is typically used so that the particle momentum is not too small or too large. Note that `p0` corresponds to particles with the input diffusion coefficients.
- `pmin`: the minimum particle momentum. It is used when injecting particles and when calculating the global particle spectrum. It is typically set to `1E-2`.
- `pmax`: the maximum particle momentum. It is used when injecting particles and when calculating the global particle spectrum. It is typically set to `1E1`.
- `momentum_dependency`: whether the diffusion coefficients depend on particle momentum. `1` is the default, which means the diffusion coefficients depend on particle momentum. `0` means the diffusion coefficients do not depend on particle momentum.
- `pindex`: the power-law index for the momentum dependency of the diffusion coefficients. It is only used when `momentum_dependency=1`. It is typically set to `3-5/3=4/3=1.3333333`, where `5/3` is the turbulence spectral slope for the Kolmogorov spectrum. It can be modified in `diffusion.sh` when using different turbulence models.
- `mag_dependency`: whether the diffusion coefficients depend on magnetic field strength. `1` is the default, which means the diffusion coefficients depend on magnetic field strength. `0` means the diffusion coefficients do not depend on magnetic field strength.
- `kpara0`: the normalization for the parallel diffusion coefficient. It is calculated based on the initial particle momentum, magnetic field, and turbulence properties in `sde.py`.
- `kret`: the ratio of the perpendicular diffusion coefficient to the parallel diffusion coefficient. It is typically set to less than `0.1`.
- `dt_min`: the minimum time step allowed to avoid infinite time step.
- `dt_min_rel`: the minimum relative time step w.r.t. one field time interval. `dt_min` is set to `dt_min_rel` times the time interval for MHD fields if the latter is larger than `dt_min`.

Note: The time step is adaptive. It is calculated based on the particle momentum, magnetic field, pitch angle, and diffusion coefficients. The rule of thumb for `dt_min_rel` is `1E-6`. For MHD simulations with lower resolution, it can

be up to 1E-4. For MHD simulations with very high resolutions, a large dt_min_rel might lead to wrong results in these high-resolution simulations, while a small dt_min_rel might lead to a long simulation time. We suggest doing a convergence test to get the optimal value.

- dt_max_rel: the maximum relative time step w.r.t. one field time interval to avoid a time step too large, which could cause the particles to jump over multiple grid cells.
- npp_global: the number of momentum bins for the global particle spectrum.
- nmu_global: the number of pitch-angle bins for global particle distributions.
- dump_interval1: the interval to dump local particle distributions. It is only used when local_dist=1 in diffusion.sh.
- pmin1: the minimum particle momentum for local particle distributions.
- pmax1: the maximum particle momentum for local particle distributions.
- npbins1: the number of momentum bins for local particle distributions.
- nmu1: the number of pitch-angle bins for local particle distributions.
- rx1: reduced factor along the x direction for local particle distributions. For every rx1 grid cell along the x direction, we will have one bin for local particle distributions.
- ry1: reduced factor along the y direction for local particle distributions. For every ry1 grid cell along the y direction, we will have one bin for local particle distributions.
- rz1: reduced factor along the z-direction for local particle distributions. For every rz1 grid cell along the z direction, we will have one bin for local particle distributions.

Note: The other three local distributions are similar. We can adjust the number of bins and reduce factors to get different distributions. For example, we can get a distribution with higher resolution in the momentum space and lower resolution in the pitch-angle space by increasing npbins and decreasing nmu. Or we can get distributions with higher momentum resolution but coarse spatial resolution by increasing rx, ry, and rz.

Note: We only dump local distributions every few MHD output intervals. When dump_interval is larger than the number of MHD outputs, it will not dump the distribution. In this way, we don't have to dump all four kinds of local distributions.

- acc_region_flag: whether to turn on particle acceleration in certain regions. 0 is the default, which means turning on particle acceleration in the entire region. 1 means turning on particle acceleration in certain regions. When acc_region_flag=1, we need to specify the acceleration region. The acceleration region is specified by acc_xmin, acc_xmax, acc_ymin, acc_ymax, acc_zmin, and acc_zmax. These are the relative values from 0 to 1. The acceleration region is a box with the minimum coordinate (acc_xmin, acc_ymin, acc_zmin) and the maximum coordinate (acc_xmax, acc_ymax, acc_zmax). It is useful when we want to selectively turn on/off particle acceleration in certain regions. For example, we can turn on particle acceleration only in the reconnection region or termination shock region in flare simulations. If we set acc_xmax or acc_ymax or acc_zmax to negative values, the acceleration in the entire simulation domain will be turned off.
- pbcx: the boundary condition for particles along the x direction. 0 is the default, which means periodic boundary condition. 1 means open boundary condition.
- pbcy: the boundary condition for particles along the y direction. 0 is the default, which means periodic boundary condition. 1 means open boundary condition.

- pbcz: the boundary condition for particles along the z direction. 0 is the default, which means periodic boundary condition. 1 means open boundary condition.

Note: Additional boundary conditions should be included in the future, such as reflecting boundary condition.

- mpi_sizex: the number of MPI processes along the x direction. It is default to 1 when size_mpi_sub=1. Otherwise, mpi_sizex*mpi_sizey*mpi_sizez should be equal to size_mpi_sub.
- mpi_sizey: the number of MPI processes along the y direction. It is default to 1 when size_mpi_sub=1. Otherwise, mpi_sizex*mpi_sizey*mpi_sizez should be equal to size_mpi_sub.
- mpi_sizez: the number of MPI processes along the z direction. It is default to 1 when size_mpi_sub=1. Otherwise, mpi_sizex*mpi_sizey*mpi_sizez should be equal to size_mpi_sub.

Note: When size_mpi_sub>1 in diffusion.sh. mpi_sizex*mpi_sizey*mpi_sizez should be equal to size_mpi_sub. Otherwise, the code will stop.

CHAPTER

FOUR

THEORY

4.1 Introduction

The GPAT code was initially developed by Fan Guo[1]. It has been updated over the years. Recent papers using this code can be found at *Publications*. The code solves particle transport equations, including Parker's transport equation and focused transport equation, using the stochastic integration method. It is supposed to be used for studying particle acceleration and transport at scales much larger than the kinetic scales.

The code solves the equation using the stochastic integration method since the Fokker-Planck form of the transport equation is equivalent to a set of the stochastic differential equations (SDEs) of Ito type[2]. The SDEs are solved using a large number of pseudo particles.

It uses MHD simulation data (magnetic field, velocity, density, etc.) as background to evolve the transport equations. It has been used in studying particle acceleration and transport in magnetic reconnection, solar flares, and solar eruption regions.

The simulations will produce spatially and temporally dependent energy spectra and maps of energetic particles. See *Usage* for how to get these from MHD simulation outputs.

4.2 Parker's Transport Equation

4.2.1 The Equation Solved

Parker's transport equation

$$\frac{\partial f}{\partial t} + (\boldsymbol{V} + \boldsymbol{V}_d) \cdot \nabla f - \frac{1}{3} \nabla \cdot \boldsymbol{V} \frac{\partial f}{\partial \ln p} = \nabla \cdot (\boldsymbol{\kappa} \nabla f) + Q, \tag{4.1}$$

where $f(x_i, p, t)$ is the particle distribution function as a function of the particle position x_i, momentum p (isotropic momentum assumed), and time t; $\boldsymbol{\kappa}$ is the spatial diffusion coefficient tensor, $\boldsymbol{V}$ is the bulk plasma velocity, $\boldsymbol{V}_d$ is the particle drift, and Q is the source. In general, $\boldsymbol{V}$ can be obtained directly from the MHD simulations, $\boldsymbol{V}_d$ and $\boldsymbol{\kappa}$ both depend on the vector magnetic field, and Q could depend on many plasma properties (e.g., number density, current density, and compression). $\boldsymbol{\kappa}$ also depends on the turbulence properties (e.g., amplitude, spectral slope, and anisotropy). The diffusion coefficient tensor is given by

$$\kappa_{ij} = \kappa_\perp \delta_{ij} - \frac{(\kappa_\perp - \kappa_\parallel) B_i B_j}{B^2},$$

where $\kappa_\parallel$ and $\kappa_\perp$ are the parallel and perpendicular diffusion coefficients. Here $\kappa_\parallel$ can be calculated from the quasi-linear theory [Jokipii71]. Assuming that magnetic turbulence is well developed and has an isotropic power spectrum

[1] https://ui.adsabs.harvard.edu/abs/2010ApJ...725..128G/abstract
[2] https://en.wikipedia.org/wiki/Fokker%E2%80%93Planck_equation

$P \sim k^{-5/3}$, the resulting $\kappa_\parallel \sim p^{4/3}$ when the particle gyroradius is much smaller than the correlation length of turbulence. In particular, we use the following expression for $\kappa_\parallel$ [Giacalone99],

$$\begin{aligned}\kappa_\parallel(v) &= \frac{3v^3}{20L_c\Omega_0^2\sigma^2}\csc\left(\frac{3\pi}{5}\right)\left[1+\frac{72}{7}\left(\frac{\Omega_0 L_c}{v}\right)^{5/3}\right] \\ &\approx 1.622\frac{v^{4/3}L_c^{2/3}}{\Omega_0^{1/3}\sigma^2}\end{aligned} \tag{4.2}$$

where v is the particle speed, L_c is the turbulence correlation length, Ω_0 is the particle gyrofrequency, and $\sigma^2 = \langle \delta B^2 \rangle / B_0^2$ is the normalized wave variance of turbulence. Reference [Giacalone99] gave a derivation of *equ_kpara_qlt* . Below is summary of it with some missing details. The velocity-dependent parallel diffusion coefficient is

$$\kappa_\parallel(v) = \frac{v^2}{4}\int_0^1 \frac{(1-\mu^2)^2 d\mu}{D_{\mu\mu}}$$

where μ is cosine of the pitch angle and $D_{\mu\mu}$ the pitch-angle diffusion coefficient. $D_{\mu\mu}$ is related to magnetic field fluctuations.

$$D_{\mu\mu} = \frac{\pi}{4}\Omega_0(1-\mu^2)\frac{k_{\text{res}}P(k_{\text{res}})}{B_0^2}$$

where Ω_0 is particle gyrofrequency, $k_{\text{res}} = |\Omega_0/v\mu|$ is the resonant wavenumber. The above equation is strictly applicable only for the case of 1D turbulence in which the wavevectors are aligned with the mean field. For anisotropic turbulence (e.g., 2D + slab), only the slab turbulence (about 20% of all turbulent fluctuations [Bieber96]) affect particle parallel transport [Florinski03]. The turbulence power is usually expressed as

$$P(k) = \frac{\langle \delta B^2 \rangle}{1+(kL_c)^\gamma}\left[\int_{k_{\text{min}}}^{k_{\text{max}}} \frac{dk}{1+(kL_c)^\gamma}\right]^{-1}$$

where k_{min} and k_{max} are the smallest and largest wavenumbers in the system, L_c is the turbulence correlation length, and γ is the turbulence spectrum index (e.g., 5/3). For $k_{\text{min}} \ll 1/L_c \ll k_{\text{max}}$, the integral in the above equation can be taken from 0 to ∞. From the table of integral, $\int_0^\infty x^{\mu-1}dx/(1+x^\nu) = \pi\csc(\mu\pi/\nu)/\nu$. Then,

$$P(k) = \frac{\langle \delta B^2 \rangle L_c}{1+(kL_c)^\gamma}\left[(\pi/\gamma)\csc(\pi/\gamma)\right]^{-1}$$

The parallel diffusion coefficient is

$$\begin{aligned}\kappa_\parallel(v) &= \frac{v^2\csc(\pi/\gamma)}{\Omega_0^2\sigma^2\gamma L_c}\int_0^1 (1-\mu^2)|v\mu|\left(1+\left|\frac{\Omega_0}{v\mu}\right|L_c\right)^\gamma d\mu \\ &= \frac{v^3\csc(\pi/\gamma)}{4\Omega_0^2\sigma^2\gamma L_c}\left[1+\left(\frac{\Omega_0 L_c}{v}\right)^\gamma \frac{8}{(2-\gamma)(4-\gamma)}\right]\end{aligned}$$

where we assume $v > 0$ and $\gamma < 2$. When $\gamma = 5/3$ (Kolmogorov), we can get *equ_kpara_qlt*. (What will happen when $\gamma > 2$?).

Test-particle simulations have suggested that $\kappa_\perp/\kappa_\parallel$ is about 0.02-0.04 and is nearly independent of particle energy [Giacalone99]. There is also observational evidence suggesting that $\kappa_\perp/\kappa_\parallel$ can be quite large [Dwyer97] [Zhang03].

The Parker transport equation can be solved by integrating the stochastic differential equation corresponding to the Fokker–Planck form of the transport equation [Zhang99] [Florinski09] [Pei10] [Kong17]. Neglecting the source term Q in *equ_parker* and assuming $F = fp^2$,

$$\frac{\partial F}{\partial t} = -\nabla\cdot[(\nabla\cdot\boldsymbol{\kappa}+\boldsymbol{V})F] + \frac{\partial}{\partial p}\left[\frac{p}{3}\nabla\cdot\boldsymbol{V}F\right] + \nabla\cdot(\nabla\cdot(\boldsymbol{\kappa}F)),$$

which is equivalent to a system of stochastic differential equations (SDEs) of the Ito type.

4.2.2 1D & 2D Models

In a 1D model, the Fokker-Planck form of the transport is equivalent to the SDEs

$$dX = \left(\frac{\partial \kappa}{\partial x} + V_x\right) ds + \sqrt{2\kappa} dW_\sigma(s),$$
$$dp = -\frac{p}{3}\frac{\partial V_x}{\partial x} ds,$$

Note: In a 1D or 2D model, the particle drift $\boldsymbol{V}_d$ is out of the simulation plane and is not considered here.

where dW_σ is the normalized distributed random number with mean zero and variance $\sqrt{\Delta t}$, and Δt is the time step for stochastic integration. This corresponds to a Wiener process. Numerical approximation is often used for the Wiener process to replace the normal distribution. We use a uniform distribution in $[-\sqrt{3}, \sqrt{3}]$ in the code.

Note: Future tests should be included to test this method thoroughly.

In a 2D model, the corresponding SDEs are

$$dX = (\nabla \cdot \boldsymbol{\kappa} + \boldsymbol{V}) ds + \sum_\sigma \boldsymbol{\alpha}_\sigma dW_\sigma(s),$$
$$dp = -\frac{p}{3}(\nabla \cdot \boldsymbol{V}) ds,$$

where $\sum_\sigma \alpha_\sigma^\mu \alpha_\sigma^\nu = 2\kappa^{\mu\nu}$.

$$\boldsymbol{\alpha}_1 = \begin{pmatrix} \sqrt{2\kappa_\perp} \\ 0 \end{pmatrix}, \quad \boldsymbol{\alpha}_2 = \begin{pmatrix} 0 \\ \sqrt{2\kappa_\perp} \end{pmatrix}, \quad \boldsymbol{\alpha}_3 = \sqrt{2(\kappa_\parallel - \kappa_\perp)} \begin{pmatrix} B_x/B \\ B_y/B \end{pmatrix}.$$

The parameters used at particle locations are calculated from v_x, v_y, B_x, B_y, $\nabla \cdot \boldsymbol{v}$, $\partial B_x/\partial x$, $\partial B_x/\partial y$, $\partial B_y/\partial x$, and $\partial B_y/\partial y$, which are all obtained from the MHD simulations. We interpolate these parameters to the particle positions and then calculate the other required parameters:

$$\frac{\partial \kappa_{xx}}{\partial x} = \frac{\partial \kappa_\perp}{\partial x} - \frac{\partial(\kappa_\perp - \kappa_\parallel)}{\partial x}\frac{B_x^2}{B^2} - 2(\kappa_\perp - \kappa_\parallel)\frac{\frac{\partial B_x}{\partial x}B_x B - \frac{\partial B}{\partial x}B_x^2}{B^3},$$
$$\frac{\partial \kappa_{yy}}{\partial y} = \frac{\partial \kappa_\perp}{\partial y} - \frac{\partial(\kappa_\perp - \kappa_\parallel)}{\partial y}\frac{B_y^2}{B^2} - 2(\kappa_\perp - \kappa_\parallel)\frac{\frac{\partial B_y}{\partial y}B_y B - \frac{\partial B}{\partial y}B_y^2}{B^3},$$
$$\frac{\partial \kappa_{xy}}{\partial x} = -\frac{\partial(\kappa_\perp - \kappa_\parallel)}{\partial x}\frac{B_x B_y}{B^2} - (\kappa_\perp - \kappa_\parallel)\frac{\left(\frac{\partial B_x}{\partial x}B_y + B_x\frac{\partial B_y}{\partial x}\right)B - 2B_x B_y\frac{\partial B}{\partial x}}{B^3},$$
$$\frac{\partial \kappa_{xy}}{\partial y} = -\frac{\partial(\kappa_\perp - \kappa_\parallel)}{\partial y}\frac{B_x B_y}{B^2} - (\kappa_\perp - \kappa_\parallel)\frac{\left(\frac{\partial B_x}{\partial y}B_y + B_x\frac{\partial B_y}{\partial y}\right)B - 2B_x B_y\frac{\partial B}{\partial y}}{B^3},$$
$$\frac{\partial B}{\partial x} = \frac{1}{B}\left(B_x\frac{\partial B_x}{\partial x} + B_y\frac{\partial B_y}{\partial x}\right),$$
$$\frac{\partial B}{\partial y} = \frac{1}{B}\left(B_x\frac{\partial B_x}{\partial y} + B_y\frac{\partial B_y}{\partial y}\right).$$

where $\kappa_\parallel$ and $\kappa_\perp$ can be functions of B_x, B_y and B, so $\partial\kappa_\parallel/\partial x$, $\partial\kappa_\parallel/\partial y$, $\partial\kappa_\perp/\partial x$, and $\partial\kappa_\perp/\partial y$ still depend on the derivatives $\partial B_x/\partial x$, $\partial B_x/\partial y$, $\partial B_y/\partial x$, and $\partial B_y/\partial y$. The detailed expressions depend on the diffusion model to choose. Considering the expression of $\kappa_\parallel$, we get

$$\frac{\partial \kappa}{\partial x} \sim \kappa\left(\frac{2}{3L_c}\frac{\partial L_c}{\partial x} - \frac{1}{3B}\frac{\partial B}{\partial x} - \frac{1}{\sigma^2}\frac{\partial(\sigma^2)}{\partial x}\right)$$

Note: We typically assume L_c and σ^2 are spatially independent. However, they could vary spatially in a large system.

Time step criteria

For a 1D problem, the particle moves a distance satisfying $l_x^2 = \max\left(\langle\Delta x\rangle^2, \langle\Delta x^2\rangle\right)$ [Strauss17], where

$$\langle\Delta x\rangle = \left(V_x + \frac{d\kappa(x)}{dx}\right)\Delta t, \quad \langle\Delta x^2\rangle = 2\kappa(x)\Delta t,$$

and l_x should be much smaller than the spatial variation scale of the fields. In this code, we require $\langle\Delta x\rangle^2 < \langle\Delta x^2\rangle$ and choose Δt so that $l_x < \delta_x$, where δ_x is the grid size. For the 2D problems, we choose the following criteria to determine the time step:

$$\begin{aligned}
\Delta t_x &= \min\left[\frac{(0.5\delta x)^2}{2\kappa_\parallel}, \frac{2\kappa_\perp}{(V_x + \partial_x\kappa_{xx} + \partial_y\kappa_{xy})^2}\right], \\
\Delta t_y &= \min\left[\frac{(0.5\delta y)^2}{2\kappa_\parallel}, \frac{2\kappa_\perp}{(V_y + \partial_y\kappa_{yy} + \partial_x\kappa_{xy})^2}\right], \\
\Delta t &= \min(\Delta t_x, \Delta t_y).
\end{aligned}$$

Higher-order method

To get a higher-order solution, we can use a derivative-free Milstein method [Burrage04] to solve the SDEs. It is different from the usual method due to one more term, which makes it a higher-order method.

Note: This method was used by [Li18] but has since not been supported in default.

$$\begin{aligned}
dX_t &= f(X_t, t)dt + g(X_t, t)dW_t, \\
X_{n+1} &= X_n + f_n h + g_n \Delta W_n + \frac{1}{2\sqrt{h}}[g(\bar{X}_n) - g_n][(\Delta W_n)^2 - h], \\
\bar{X}_n &= X_n + f_n h + g_n\sqrt{h}, \\
\Delta W_n &= [W_{t+h} - W_t] \sim \sqrt{h}N(0, 1),
\end{aligned}$$

where X corresponds to spatial positions x, y and particle momentum p in our simulation. Here $f(X_t, t)$ is the deterministic term, $g(X_t, t)$ is the probabilistic term, h is the time step, and $N(0, 1)$ indicates a normal distribution, which is substituted with a uniform distribution $[-\sqrt{3}, \sqrt{3}]$ in our simulations to speed up the computation.

4.2.3 3D Model

The relationship $\sum_\sigma \alpha_\sigma^\mu \alpha_\sigma^\nu = 2\kappa^{\mu\nu}$ is actually a matrix decomposition. We need to decompose $2\kappa = PP^T$, where $P = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3)$. In a 2D problem, the third component of $\boldsymbol{\alpha}_i$ is essentially 0. In a 3D problem, we need to find all three components of $\boldsymbol{\alpha}_i$. We need some linear algebra for that. Every real symmetric matrix can be written in the form (https://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix#Real_symmetric_matrices)

$$A = Q\Lambda Q^T$$

where Q is an orthogonal matrix whose columns are the eigenvectors of A, and Λ is a diagonal matrix whose entries are the eigenvalues of A. If the eigenvalues are non-negative, then the real matrix $P = Q\Lambda^{1/2}$, and

$$A = Q\Lambda^{1/2}\Lambda^{1/2}Q^T = \frac{PP^T}{2}$$

According to WolframAlpha, the eigenvalue of κ is $k_\parallel$, $k_\perp$, and $k_\perp$, and the corresponding eigenvectors are

$$\begin{aligned}
v_1 &= \left(\frac{b_x}{b_z}, \frac{b_y}{b_z}, 1\right), \\
v_2 &= \left(-\frac{b_z}{b_x}, 0, 1\right), \\
v_3 &= \left(-\frac{b_y}{b_x}, 1, 0\right).
\end{aligned}$$

where $b_x = B_x/B$, $b_y = B_y/B$, and $b_z = B_z/B$. v_1, v_2, and v_3 are not unit vectors, and v_2 and v_3 are not orthogonal to v_1, so we need to re-organize v_2 and v_3 and normalize the vectors.

$$\begin{aligned}
v_1 &= (b_x, b_y, b_z), \\
v_2 &= \left(-\frac{b_x b_z}{\sqrt{b_x^2 + b_y^2}}, -\frac{b_y b_z}{\sqrt{b_x^2 + b_y^2}}, \sqrt{b_x^2 + b_y^2}\right), \\
v_3 &= \left(-\frac{b_y}{\sqrt{b_x^2 + b_y^2}}, \frac{b_x}{\sqrt{b_x^2 + b_y^2}}, 0\right)
\end{aligned}$$

where v_2 is calculated from the perpendicular component of the original v_2 w.r.t. v_3. Then,

$$Q = \begin{pmatrix}
b_x & -b_x b_z/\sqrt{b_x^2 + b_y^2} & -b_y/\sqrt{b_x^2 + b_y^2} \\
b_y & -b_y b_z/\sqrt{b_x^2 + b_y^2} & b_x/\sqrt{b_x^2 + b_y^2} \\
b_z & \sqrt{b_x^2 + b_y^2} & 0
\end{pmatrix}$$

$$\Lambda = \begin{pmatrix}
\kappa_\parallel & 0 & 0 \\
0 & \kappa_\perp & 0 \\
0 & 0 & \kappa_\perp
\end{pmatrix}$$

$$P = \sqrt{2}Q\Lambda^{1/2} = \begin{pmatrix}
b_x\sqrt{2\kappa_\parallel} & -b_x b_z\sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} & -b_y\sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} \\
b_y\sqrt{2\kappa_\parallel} & -b_y b_z\sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} & b_x\sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} \\
b_z\sqrt{2\kappa_\parallel} & \sqrt{b_x^2 + b_y^2}\sqrt{2\kappa_\perp} & 0
\end{pmatrix}$$

We can verify that $PP^T = 2\kappa$. For 3D simulation, we need to calculate more terms of the gradient of the diffusion tensor. The parameters used at particle locations are calculated from V_x, V_y, V_z, b_x, b_y, b_z, $\nabla \cdot \boldsymbol{V}$, $\partial_x b_x$, $\partial_y b_x$, $\partial_z b_x$, $\partial_x b_y$, $\partial_y b_y$, $\partial_z b_y$, $\partial_x b_z$, $\partial_y b_z$, $\partial_z b_z$.

$$\begin{aligned}
\partial_x\kappa_{xx} &= \partial_x\kappa_\perp + \partial_x(\kappa_\parallel - \kappa_\perp)b_x^2 + 2(\kappa_\parallel - \kappa_\perp)b_x\partial_x b_x, \\
\partial_y\kappa_{yy} &= \partial_y\kappa_\perp + \partial_y(\kappa_\parallel - \kappa_\perp)b_y^2 + 2(\kappa_\parallel - \kappa_\perp)b_y\partial_y b_y, \\
\partial_z\kappa_{zz} &= \partial_z\kappa_\perp + \partial_z(\kappa_\parallel - \kappa_\perp)b_z^2 + 2(\kappa_\parallel - \kappa_\perp)b_z\partial_z b_z, \\
\partial_x\kappa_{xy} &= \partial_x(\kappa_\parallel - \kappa_\perp)b_x b_y + (\kappa_\parallel - \kappa_\perp)(\partial_x b_x b_y + b_x\partial_x b_y), \\
\partial_y\kappa_{xy} &= \partial_y(\kappa_\parallel - \kappa_\perp)b_x b_y + (\kappa_\parallel - \kappa_\perp)(\partial_y b_x b_y + b_x\partial_y b_y), \\
\partial_x\kappa_{xz} &= \partial_x(\kappa_\parallel - \kappa_\perp)b_x b_z + (\kappa_\parallel - \kappa_\perp)(\partial_x b_x b_z + b_x\partial_x b_z), \\
\partial_z\kappa_{xz} &= \partial_z(\kappa_\parallel - \kappa_\perp)b_x b_z + (\kappa_\parallel - \kappa_\perp)(\partial_z b_x b_z + b_x\partial_z b_z), \\
\partial_y\kappa_{yz} &= \partial_y(\kappa_\parallel - \kappa_\perp)b_y b_z + (\kappa_\parallel - \kappa_\perp)(\partial_y b_y b_z + b_y\partial_y b_z), \\
\partial_z\kappa_{yz} &= \partial_z(\kappa_\parallel - \kappa_\perp)b_y b_z + (\kappa_\parallel - \kappa_\perp)(\partial_z b_y b_z + b_y\partial_z b_z)
\end{aligned}$$

Or we may prefer to use current code structure that calculates $\partial_x B_x$, $\partial_y B_x$, $\partial_z B_x$, $\partial_x B_y$, $\partial_y B_y$, $\partial_z B_y$, $\partial_x B_z$, $\partial_y B_z$, $\partial_z B_z$. Then, the derivatives are calculated as

$$
\begin{aligned}
\partial_x B &= b_x \partial_x B_x + b_y \partial_x B_y + b_z \partial_x B_z, \\
\partial_y B &= b_x \partial_y B_x + b_y \partial_y B_y + b_z \partial_y B_z, \\
\partial_z B &= b_x \partial_z B_x + b_y \partial_z B_y + b_z \partial_z B_z, \\
\partial_x \kappa_{xx} &= \partial_x \kappa_\perp + \partial_x(\kappa_\parallel - \kappa_\perp) b_x^2 + 2(\kappa_\parallel - \kappa_\perp)(B_x B \partial_x B_x - B_x^2 \partial_x B)/B^3, \\
\partial_y \kappa_{yy} &= \partial_y \kappa_\perp + \partial_y(\kappa_\parallel - \kappa_\perp) b_y^2 + 2(\kappa_\parallel - \kappa_\perp)(B_y B \partial_y B_y - B_y^2 \partial_y B)/B^3, \\
\partial_z \kappa_{zz} &= \partial_z \kappa_\perp + \partial_z(\kappa_\parallel - \kappa_\perp) b_z^2 + 2(\kappa_\parallel - \kappa_\perp)(B_z B \partial_z B_z - B_z^2 \partial_z B)/B^3, \\
\partial_x \kappa_{xy} &= \partial_x(\kappa_\parallel - \kappa_\perp) b_x b_y + (\kappa_\parallel - \kappa_\perp)[(B_y \partial_x B_x + B_x \partial_x B_y) B - 2 B_x B_y \partial_x B]/B^3, \\
\partial_y \kappa_{xy} &= \partial_y(\kappa_\parallel - \kappa_\perp) b_x b_y + (\kappa_\parallel - \kappa_\perp)[(B_y \partial_y B_x + B_x \partial_y B_y) B - 2 B_x B_y \partial_y B]/B^3, \\
\partial_x \kappa_{xz} &= \partial_x(\kappa_\parallel - \kappa_\perp) b_x b_z + (\kappa_\parallel - \kappa_\perp)[(B_z \partial_x B_x + B_x \partial_x B_z) B - 2 B_x B_z \partial_x B]/B^3, \\
\partial_z \kappa_{xz} &= \partial_z(\kappa_\parallel - \kappa_\perp) b_x b_z + (\kappa_\parallel - \kappa_\perp)[(B_z \partial_z B_x + B_x \partial_z B_z) B - 2 B_x B_z \partial_z B]/B^3, \\
\partial_y \kappa_{yz} &= \partial_y(\kappa_\parallel - \kappa_\perp) b_y b_z + (\kappa_\parallel - \kappa_\perp)[(B_z \partial_y B_y + B_y \partial_y B_z) B - 2 B_y B_z \partial_y B]/B^3, \\
\partial_z \kappa_{yz} &= \partial_z(\kappa_\parallel - \kappa_\perp) b_y b_z + (\kappa_\parallel - \kappa_\perp)[(B_z \partial_z B_y + B_y \partial_z B_z) B - 2 B_y B_z \partial_z B]/B^3.
\end{aligned}
$$

Particle drift velocity

In the 3D model, we need the drift velocity, which is given by

$$
\begin{aligned}
\boldsymbol{V}_d &= \frac{pcw}{3q} \nabla \times \left(\frac{\boldsymbol{B}}{B^2} \right) = \frac{1}{3q} \frac{p^2 c}{\sqrt{p^2 + m^2 c^2}} \left(\frac{1}{B^2} \nabla \times \boldsymbol{B} - \frac{2}{B^3} \nabla B \times \boldsymbol{B} \right) \\
\nabla \times \boldsymbol{B} &= (\partial_y B_z - \partial_z B_y) \hat{i} + (\partial_z B_x - \partial_x B_z) \hat{j} + (\partial_x B_y - \partial_y B_x) \hat{k} \\
\nabla B \times \boldsymbol{B} &= (B_z \partial_y B - B_y \partial_z B) \hat{i} + (B_x \partial_z B - B_z \partial_x B) \hat{j} + (B_y \partial_x B - B_x \partial_y B) \hat{k}
\end{aligned}
$$

where $p = \gamma m v$ is particle momentum, c is the speed of light, $w = v/c$ is the normalized particle speed, and q is particle charge. Using normalized quantities, we have

$$
\begin{aligned}
\tilde{\boldsymbol{V}}_d &= \frac{1}{v_A} \frac{1}{3\tilde{q}e} \frac{\tilde{p}^2 p_0^2 c}{\sqrt{\tilde{p}^2 p_0^2 + m^2 c^2}} \frac{1}{B_0 L_0} \left(\frac{1}{\tilde{B}^2} \tilde{\nabla} \times \tilde{\boldsymbol{B}} - \frac{2}{\tilde{B}^3} \tilde{\nabla} \tilde{B} \times \tilde{\boldsymbol{B}} \right) \\
&= \frac{1}{\sqrt{d_1^2 \tilde{p}^{-2} + d_2^2 \tilde{p}^{-4}}} \frac{1}{3\tilde{q}} \left(\frac{1}{\tilde{B}^2} \tilde{\nabla} \times \tilde{\boldsymbol{B}} - \frac{2}{\tilde{B}^3} \tilde{\nabla} \tilde{B} \times \tilde{\boldsymbol{B}} \right)
\end{aligned}
$$

where $\tilde{\boldsymbol{V}}_d = \boldsymbol{V}_d / v_A$, $\tilde{q} = q/e$, $\tilde{\nabla} = L_0 \nabla$, $\tilde{\boldsymbol{B}} = \boldsymbol{B}/B_0$, $\tilde{p} = p/p_0$, $d_1 = eB_0 v_A L_0/(p_0 c)$, and $d_2 = emB_0 v_A L_0 / p_0^2$. Note that in the code, $\tilde{p}$ will be re-normalized. For example, $\tilde{p}_0 = 1$ might correspond to $\tilde{p}_{n0} = 0.1$ in simulations. The re-normalized numerical momentum $\tilde{p}_n = \tilde{p} \tilde{p}_{n0}$. Thus, $\tilde{p} = \tilde{p}_n / \tilde{p}_{n0}$ in simulations, and we need provide d_1 and d_2 based on the normalization.

Note: The velocity normalization v_A should be changed to v_0 if $v_0 \neq v_A$

4.2.4 Momentum Diffusion

Note: Momentum diffusion is only partially supported now.

We can include an momentum diffusion term to the right side of the Parker transport equation.

$$\frac{\partial f}{\partial t} + (\boldsymbol{V} + \boldsymbol{V}_d)\cdot\nabla f - \frac{1}{3}\nabla\cdot\boldsymbol{V}\frac{\partial f}{\partial \ln p} = \nabla\cdot(\boldsymbol{\kappa}\nabla f) + \frac{1}{p^2}\frac{\partial}{\partial p}\left(p^2 D_{pp}\frac{\partial f}{\partial p}\right) + Q, \tag{4.3}$$

Neglecting the source term Q in *equ_parker_2nd* and assuming $F = fp^2$,

$$\begin{aligned}\frac{\partial F}{\partial t} =& -\nabla\cdot[(\nabla\cdot\boldsymbol{\kappa} + \boldsymbol{V} + \boldsymbol{V}_d)F] + \nabla\cdot(\nabla\cdot(\boldsymbol{\kappa}F)) + \\ & \frac{\partial}{\partial p}\left[\left(\frac{p}{3}\nabla\cdot\boldsymbol{V} - \frac{\partial D_{pp}}{\partial p} - \frac{2D_{pp}}{p}\right)F\right] + \frac{\partial(D_{pp}F)}{\partial p^2}.\end{aligned}$$

which is equivalent to a system of SDEs of the Ito type,

$$\begin{aligned}dX &= (\nabla\cdot\boldsymbol{\kappa} + \boldsymbol{V} + \boldsymbol{V}_d)ds + \sum_\sigma \boldsymbol{\alpha}_\sigma dW_\sigma(s) \\ dp &= \left(-\frac{p}{3}\nabla\cdot\boldsymbol{V} + \frac{\partial D_{pp}}{\partial p} + \frac{2D_{pp}}{p}\right)ds + \sqrt{2D_{pp}}dW(s)\end{aligned}$$

where $\sum_\sigma \alpha^\mu_\sigma\alpha^\nu_\sigma = 2\kappa^{\mu\nu}$, dW is the normalized distributed random number with mean zero and variance $\sqrt{\Delta t}$, and Δt is the time step for stochastic integration.

Wave-particle interaction

For a 2D problem, reference [Skilling75] shows that for forward and backward propagating Alfvén waves,

$$\begin{aligned}\boldsymbol{u} &= \boldsymbol{v}_0 + \left\langle \frac{3}{2}(1-\mu^2)\frac{\nu^+ - \nu^-}{\nu^+ + \nu^-}\right\rangle, \text{ the velocity of mean wave frame} \\ \kappa_\parallel &= v^2\left\langle \frac{1-\mu^2}{2(\nu^+ + \nu^-)}\right\rangle, \text{ parallel spatial diffusion coefficient} \\ D_{pp} &= 4\gamma^2 m^2 v_A^2\left\langle \frac{1-\mu^2}{2}\frac{\nu^+\nu^-}{\nu^+ + \nu^-}\right\rangle, \text{ momentum diffusion coefficient}\end{aligned}$$

where $\langle \dots \rangle$ donates μ-average, ν^+ and ν^- are collision frequency against forward waves and backward waves, respectively. If ν^+ is equal to ν^-,

$$D_{pp} = \frac{1}{9}\frac{p^2 v_A^2}{\kappa_\parallel}$$

where $p = \gamma m v$ is particle momentum. Depending on the plasma parameter and wave properties, we may have to use more complicated models [Schlickeiser89] [Schlickeiser98] [LeRoux07]. The corresponding SDE is

$$\begin{aligned}dp &= \left(-\frac{p}{3}\nabla\cdot\boldsymbol{V} + \frac{4pv_A^2}{9\kappa_\parallel}\right)ds + \sqrt{\frac{2p^2v_A^2}{9\kappa_\parallel}}dW(s), \text{if } \kappa_\parallel \text{ is independent of } p \\ dp &= \left(-\frac{p}{3}\nabla\cdot\boldsymbol{V} + \frac{8pv_A^2}{27\kappa_\parallel}\right)ds + \sqrt{\frac{2p^2v_A^2}{9\kappa_\parallel}}dW(s), \text{if } \kappa_\parallel \sim p^{4/3}\end{aligned}$$

which are normalized to

$$d\tilde{p}_n = \left(-\frac{\tilde{p}_n}{3}\tilde{\nabla}\cdot\tilde{\boldsymbol{V}} + \frac{4\tilde{p}_n\tilde{v}_A^2}{9\tilde{\kappa}_\parallel}\right)d\tilde{s} + \tilde{p}_n\tilde{v}_A\sqrt{\frac{2}{9\tilde{\kappa}_\parallel}}dW(\tilde{s}), \text{if } \kappa_\parallel \text{ is independent of } p$$

$$d\tilde{p}_n = \left(-\frac{\tilde{p}_n}{3}\tilde{\nabla}\cdot\tilde{\boldsymbol{V}} + \frac{8\tilde{p}_n\tilde{v}_A^2}{27\tilde{\kappa}_\parallel}\right)d\tilde{s} + \tilde{p}_n\tilde{v}_A\sqrt{\frac{2}{9\tilde{\kappa}_\parallel}}dW(\tilde{s}), \text{if } \kappa_\parallel \sim p^{4/3}$$

where $\tilde{p}_n = \tilde{p}\tilde{p}_{n0} = p\tilde{p}_{n0}/p_0$, where is $\tilde{p}_{n0}$ is the numerical value for particles with p_0 in the code (e.g., 0.1 as often used), $\tilde{\nabla} = L_0\nabla$, $\tilde{\boldsymbol{V}} = \boldsymbol{V}/v_{A0}$, $\tilde{v}_A = \tilde{v}_{A0}$, $\tilde{\kappa}_\parallel = \kappa_\parallel/\kappa_0$, $\kappa_0 = L_0 v_{A0}$, $\tilde{s} = s/t_0$, and $t_0 = L_0/v_{A0}$. These are all given in the code.

Flow shear

For isotropic particle distributions, the flow shear introduces another momentum diffusion term. If there is no average magnetic field [Earl88].

$$D_{pp} = \Gamma\tau p^2,$$
$$\Gamma = \frac{1}{30}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)^2 - \frac{2}{45}\frac{\partial U_i}{\partial x_i}\frac{\partial U_j}{\partial x_j} = \frac{2}{15}\sum_{ij}\sigma_{ij}^2$$

where Γ is the coefficient of viscous momentum transfer, $\sigma_{ij} = (\partial_i U_j + \partial_j U_i - 2\nabla\cdot\boldsymbol{U}\delta_{ij}/3)/2$ is the shear tensor, τ is the relaxation time for particle scattering. According to [Webb18], τ is related particle diffusion coefficient $\kappa_\parallel = v^2\tau/3$. The corresponding SDE is

$$dp = \left(-\frac{p}{3}\nabla\cdot\boldsymbol{v} + \frac{\Gamma}{p^2}\frac{\partial(p^4\tau)}{\partial p}\right)ds + \sqrt{2\Gamma\tau p^2}dW(s)$$

For $\tau \sim \tau_0(p_0/p)^\alpha$,

$$dp = \left(-\frac{p}{3}\nabla\cdot\boldsymbol{v} + \frac{\Gamma\tau_0 p_0^\alpha}{p^2}(4-\alpha)p^{3-\alpha}\right)ds + \sqrt{2\Gamma\tau_0 p_0^\alpha p^{2-\alpha}}dW(s)$$

which is normalized to

$$d\tilde{p}_n = \left(-\frac{\tilde{p}_n}{3}\tilde{\nabla}\cdot\tilde{\boldsymbol{v}} + (4-\alpha)\tilde{\Gamma}\tilde{\tau}_0\tilde{p}_n^{1-\alpha}\tilde{p}_{n0}^\alpha\right)d\tilde{s} + \sqrt{2\tilde{\Gamma}\tilde{\tau}_0\tilde{p}_n^{2-\alpha}\tilde{p}_{n0}^\alpha}dW(\tilde{s})$$

where $\tilde{p}_n = \tilde{p}\tilde{p}_{n0} = p\tilde{p}_{n0}/p_0$, where is $\tilde{p}_{n0}$ is the numerical value for particles with p_0 in the code (e.g., 0.1 as often used), $\tilde{\nabla} = L_0\nabla$, $\tilde{\boldsymbol{v}} = \boldsymbol{v}/v_{A0}$, $\tilde{\Gamma} = \Gamma t_0^2$, $\tilde{\tau}_0 = \tau_0/t_0$, $\tilde{s} = s/t_0$, and $t_0 = L_0/v_{A0}$. For $\tau \sim \tau_0(p_0/p)^2$ [Earl88],

$$d\tilde{p}_n = \left(-\frac{\tilde{p}_n}{3}\tilde{\nabla}\cdot\tilde{\boldsymbol{v}} + \frac{2\tilde{\Gamma}\tilde{\tau}_0\tilde{p}_{n0}^2}{\tilde{p}_n}\right)d\tilde{s} + \sqrt{2\tilde{\Gamma}\tilde{\tau}_0\tilde{p}_{n0}^2}dW(\tilde{s})$$

For $\tau \sim \tau_0(p_0/p)^{2/3}$ [Giacalone99],

$$d\tilde{p}_n = \left(-\frac{\tilde{p}_n}{3}\tilde{\nabla}\cdot\tilde{\boldsymbol{v}} + \frac{10}{3}\tilde{\Gamma}\tilde{\tau}_0\tilde{p}_n^{1/3}\tilde{p}_{n0}^{2/3}\right)d\tilde{s} + \sqrt{2\tilde{\Gamma}\tilde{\tau}_0\tilde{p}_n^{4/3}\tilde{p}_{n0}^{2/3}}dW(\tilde{s})$$
$$\tau_0 = 3\kappa_{\parallel 0}/v_0^2$$

If there is an average magnetic field, the equation is more complicated (see [Williams91] [Williams93]).

4.2.5 Spherical Coordinates

In spherical coordinates, the drift velocity

$$\boldsymbol{V}_d = \frac{pcw}{3q}\nabla\times\left(\frac{\boldsymbol{B}}{B^2}\right) = \frac{1}{3q}\frac{p^2c^3}{\sqrt{p^2c^2+m^2c^4}}\left(\frac{1}{B^2}\nabla\times\boldsymbol{B} - \frac{2}{B^3}\nabla B\times\boldsymbol{B}\right)$$

$$(\nabla\times\boldsymbol{B})_r = \frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta B_\phi) - \frac{1}{r\sin\theta}\frac{\partial B_\theta}{\partial\phi} = \frac{1}{r}\frac{\partial B_\phi}{\partial\theta} + \frac{\cos\theta}{r\sin\theta}B_\phi - \frac{1}{r\sin\theta}\frac{\partial B_\theta}{\partial\phi}$$

$$(\nabla\times\boldsymbol{B})_\theta = \frac{1}{r\sin\theta}\frac{\partial B_r}{\partial\phi} - \frac{1}{r}\frac{\partial}{\partial r}(rB_\phi) = \frac{1}{r\sin\theta}\frac{\partial B_r}{\partial\phi} - \frac{\partial B_\phi}{\partial r} - \frac{B_\phi}{r}$$

$$(\nabla\times\boldsymbol{B})_\phi = \frac{1}{r}\frac{\partial}{\partial r}(rB_\theta) - \frac{1}{r}\frac{\partial B_r}{\partial\theta} = \frac{\partial B_\theta}{\partial r} + \frac{B_\theta}{r} - \frac{1}{r}\frac{\partial B_r}{\partial\theta}$$

$$(\nabla B)_r = \frac{\partial B}{\partial r};\quad (\nabla B)_\theta = \frac{1}{r}\frac{\partial B}{\partial\theta};\quad (\nabla B)_\phi = \frac{1}{r\sin\theta}\frac{\partial B}{\partial\phi}$$

$$(\nabla B\times\boldsymbol{B})_r = (\nabla B)_\theta B_\phi - (\nabla B)_\phi B_\theta$$

$$(\nabla B\times\boldsymbol{B})_\theta = (\nabla B)_\phi B_r - (\nabla B)_r B_\phi$$

$$(\nabla B\times\boldsymbol{B})_\phi = (\nabla B)_r B_\theta - (\nabla B)_\theta B_r$$

The spatial diffusion coefficient is in the same form.

$$\kappa = \begin{bmatrix}\kappa_{rr} & \kappa_{r\theta} & \kappa_{r\phi}\\ \kappa_{r\theta} & \kappa_{\theta\theta} & \kappa_{\theta\phi}\\ \kappa_{r\phi} & \kappa_{\theta\phi} & \kappa_{\phi\phi}\end{bmatrix}$$

where $\kappa_{ij} = \kappa_\perp\delta_{ij} - (\kappa_\perp - \kappa_\parallel)b_ib_j$, and i,j are r,θ,ϕ.

Since κ is of the same form as that in the Cartesian coordinates, the gradients of κ are

$$\partial_r\kappa_{rr} = \partial_r\kappa_\perp + \partial_r(\kappa_\parallel-\kappa_\perp)b_r^2 + 2(\kappa_\parallel-\kappa_\perp)(B_rB\partial_rB_r - B_r^2\partial_rB)/B^3,$$
$$\partial_\theta\kappa_{\theta\theta} = \partial_\theta\kappa_\perp + \partial_\theta(\kappa_\parallel-\kappa_\perp)b_\theta^2 + 2(\kappa_\parallel-\kappa_\perp)(B_\theta B\partial_\theta B_\theta - B_\theta^2\partial_\theta B)/B^3,$$
$$\partial_\phi\kappa_{\phi\phi} = \partial_\phi\kappa_\perp + \partial_\phi(\kappa_\parallel-\kappa_\perp)b_\phi^2 + 2(\kappa_\parallel-\kappa_\perp)(B_\phi B\partial_\phi B_\phi - B_\phi^2\partial_\phi B)/B^3,$$
$$\partial_r\kappa_{r\theta} = \partial_r(\kappa_\parallel-\kappa_\perp)b_rb_\theta + (\kappa_\parallel-\kappa_\perp)[(B_\theta\partial_rB_r + B_r\partial_rB_\theta)B - 2B_rB_\theta\partial_rB]/B^3,$$
$$\partial_\theta\kappa_{r\theta} = \partial_\theta(\kappa_\parallel-\kappa_\perp)b_rb_\theta + (\kappa_\parallel-\kappa_\perp)[(B_\theta\partial_\theta B_r + B_r\partial_\theta B_\theta)B - 2B_rB_\theta\partial_\theta B]/B^3,$$
$$\partial_r\kappa_{r\phi} = \partial_r(\kappa_\parallel-\kappa_\perp)b_rb_\phi + (\kappa_\parallel-\kappa_\perp)[(B_\phi\partial_rB_r + B_r\partial_rB_\phi)B - 2B_rB_\phi\partial_rB]/B^3,$$
$$\partial_\phi\kappa_{r\phi} = \partial_\phi(\kappa_\parallel-\kappa_\perp)b_rb_\phi + (\kappa_\parallel-\kappa_\perp)[(B_\phi\partial_\phi B_r + B_r\partial_\phi B_\phi)B - 2B_rB_\phi\partial_\phi B]/B^3,$$
$$\partial_\theta\kappa_{\theta\phi} = \partial_\theta(\kappa_\parallel-\kappa_\perp)b_\theta b_\phi + (\kappa_\parallel-\kappa_\perp)[(B_\phi\partial_\theta B_\theta + B_\theta\partial_\theta B_\phi)B - 2B_\theta B_\phi\partial_\theta B]/B^3,$$
$$\partial_\phi\kappa_{\theta\phi} = \partial_\phi(\kappa_\parallel-\kappa_\perp)b_\theta b_\phi + (\kappa_\parallel-\kappa_\perp)[(B_\phi\partial_\phi B_\theta + B_\theta\partial_\phi B_\phi)B - 2B_\theta B_\phi\partial_\phi B]/B^3.$$

We then need to transfer the Parker transport equation to the spherical coordinates. Since we don't have cross-diffusion terms (spatial and momentum), we can ignore the momentum diffusion for now.

Note: For a more complete equation, the cross-diffusion terms should be included.

$$\frac{\partial F}{\partial t} = -(\boldsymbol{V}+\boldsymbol{V}_d)\cdot\nabla F - (\nabla\cdot\boldsymbol{V})F + \frac{\partial}{\partial p}\left[\frac{p}{3}(\nabla\cdot\boldsymbol{V})F\right] + \nabla\cdot(\boldsymbol{\kappa}\cdot\nabla F)$$

where $F = fp^2$. Since $\nabla\cdot\boldsymbol{V}_d = 0$, we can add one more term $-(\nabla\cdot\boldsymbol{V}_d)F$ to the right. Then,

$$\frac{\partial F}{\partial t} = -\nabla\cdot((\boldsymbol{V}+\boldsymbol{V}_d)F) + \frac{\partial}{\partial p}\left[\frac{p}{3}(\nabla\cdot\boldsymbol{V})F\right] + \nabla\cdot(\boldsymbol{\kappa}\cdot\nabla F)$$

Taking $\boldsymbol{V} + \boldsymbol{V}_d \rightarrow \boldsymbol{V}$,

$$\begin{aligned}\nabla \cdot (\boldsymbol{V}F) &= \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 V_r F) + \frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta V_\theta F) + \frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}(V_\phi F) \\ &= \frac{\partial(V_r F)}{\partial r} + \frac{2}{r}V_r F + \frac{\partial}{\partial\theta}\left(\frac{V_\theta F}{r}\right) + \frac{\cos\theta}{r\sin\theta}V_\theta F + \frac{\partial}{\partial\phi}\left(\frac{V_\phi F}{r\sin\theta}\right)\end{aligned}$$

so there is 2 additional terms (2nd and 4th) if we want to write the equation Fokker–Planck form. It turns out that we need to change F to $F_1 = F\sin\theta r^2$ [Jokipii77] [Pei10]. Multiplying the above equation by $r^2\sin\theta$, we get

$$r^2\sin\theta\nabla\cdot(\boldsymbol{V}F) = \frac{\partial(V_r F_1)}{\partial r} + \frac{\partial}{\partial\theta}\left(\frac{V_\theta F_1}{r}\right) + \frac{\partial}{\partial\phi}\left(\frac{V_\phi F_1}{r\sin\theta}\right)$$

For the diffusion term,

$$\begin{aligned}\boldsymbol{\kappa}\cdot\nabla F = &\left(\kappa_{rr}\frac{\partial F}{\partial r} + \kappa_{r\theta}\frac{1}{r}\frac{\partial F}{\partial\theta} + \kappa_{r\phi}\frac{1}{r\sin\theta}\frac{\partial F}{\partial\phi}\right)\hat{e}_r + \\ &\left(\kappa_{r\theta}\frac{\partial F}{\partial r} + \kappa_{\theta\theta}\frac{1}{r}\frac{\partial F}{\partial\theta} + \kappa_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial F}{\partial\phi}\right)\hat{e}_\theta + \\ &\left(\kappa_{r\phi}\frac{\partial F}{\partial r} + \kappa_{\theta\phi}\frac{1}{r}\frac{\partial F}{\partial\theta} + \kappa_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial F}{\partial\phi}\right)\hat{e}_\phi\end{aligned}$$

Taking $\boldsymbol{A} = \boldsymbol{\kappa}\cdot\nabla F$,

$$r^2\sin\theta\nabla\cdot\boldsymbol{A} = \frac{\partial(r^2\sin\theta A_r)}{\partial r} + \frac{\partial(r\sin\theta A_\theta)}{\partial\theta} + \frac{\partial(rA_\phi)}{\partial\phi}$$

The 1st term on the right is expanded to

$$\begin{aligned}&\frac{\partial^2}{\partial r^2}(\kappa_{rr}F_1) + \frac{\partial^2}{\partial r\partial\theta}\left(\frac{\kappa_{r\theta}}{r}F_1\right) + \frac{\partial^2}{\partial r\partial\phi}\left(\frac{\kappa_{r\phi}}{r\sin\theta}F_1\right) \\ &-\frac{\partial}{\partial r}\left[\left(\frac{1}{r^2}\frac{\partial(r^2\kappa_{rr})}{\partial r} + \frac{1}{r}\frac{\partial\kappa_{r\theta}}{\partial\theta} + \frac{\cos\theta}{r\sin\theta}\kappa_{r\theta} + \frac{1}{r\sin\theta}\frac{\partial\kappa_{r\phi}}{\partial\phi}\right)F_1\right]\end{aligned}$$

The 2nd term one the right is expanded to

$$\begin{aligned}&\frac{\partial^2}{\partial r\partial\theta}\left(\frac{\kappa_{r\theta}}{r}F_1\right) + \frac{\partial^2}{\partial\theta^2}\left(\frac{\kappa_{\theta\theta}}{r^2}F_1\right) + \frac{\partial^2}{\partial\theta\partial\phi}\left(\frac{\kappa_{\theta\phi}}{r^2\sin\theta}F_1\right) \\ &-\frac{\partial}{\partial\theta}\left[\left(\frac{1}{r^2}\frac{\partial(r\kappa_{r\theta})}{\partial r} + \frac{1}{r^2\sin\theta}\frac{\partial(\sin\theta\kappa_{\theta\theta})}{\partial\theta} + \frac{1}{r^2\sin\theta}\frac{\partial\kappa_{\theta\phi}}{\partial\phi}\right)F_1\right]\end{aligned}$$

The 3rd term one the right is expanded to

$$\begin{aligned}&\frac{\partial^2}{\partial r\partial\phi}\left(\frac{\kappa_{r\phi}}{r\sin\theta}F_1\right) + \frac{\partial^2}{\partial\theta\partial\phi}\left(\frac{\kappa_{\theta\phi}}{r^2\sin\theta}F_1\right) + \frac{\partial^2}{\partial\phi^2}\left(\frac{\kappa_{\phi\phi}}{r^2\sin^2\theta}F_1\right) \\ &-\frac{\partial}{\partial\phi}\left[\left(\frac{1}{r^2\sin\theta}\frac{\partial(r\kappa_{r\phi})}{\partial r} + \frac{1}{r^2\sin\theta}\frac{\partial(\kappa_{\theta\phi})}{\partial\theta} + \frac{1}{r^2\sin^2\theta}\frac{\partial\kappa_{\phi\phi}}{\partial\phi}\right)F_1\right]\end{aligned}$$

The final transferred version of Parker transport equation is

$$\begin{aligned}
\frac{\partial F_1}{\partial t} =& -\frac{\partial}{\partial r}\left[\left(v_r + v_{dr} + \frac{1}{r^2}\frac{\partial(r^2\kappa_{rr})}{\partial r} + \frac{1}{r}\frac{\partial \kappa_{r\theta}}{\partial \theta} + \frac{\cos\theta}{r\sin\theta}\kappa_{r\theta} + \frac{1}{r\sin\theta}\frac{\partial \kappa_{r\phi}}{\partial \phi}\right)F_1\right] \\
&- \frac{\partial}{\partial \theta}\left[\left(\frac{v_\theta + v_{d\theta}}{r} + \frac{1}{r^2}\frac{\partial(r\kappa_{r\theta})}{\partial r} + \frac{1}{r^2\sin\theta}\frac{\partial(\sin\theta\kappa_{\theta\theta})}{\partial \theta} + \frac{1}{r^2\sin\theta}\frac{\partial \kappa_{\theta\phi}}{\partial \phi}\right)F_1\right] \\
&- \frac{\partial}{\partial \phi}\left[\left(\frac{v_\phi + v_{d\phi}}{r\sin\theta} + \frac{1}{r^2\sin\theta}\frac{\partial(r\kappa_{r\phi})}{\partial r} + \frac{1}{r^2\sin\theta}\frac{\partial(\kappa_{\theta\phi})}{\partial \theta} + \frac{1}{r^2\sin^2\theta}\frac{\partial \kappa_{\phi\phi}}{\partial \phi}\right)F_1\right] \\
&+ \frac{\partial}{\partial p}\left(\frac{p}{3}\left(\frac{1}{r^2}\frac{\partial(r^2 v_r)}{\partial r} + \frac{1}{r\sin\theta}\frac{\partial(\sin\theta v_\theta)}{\partial \theta} + \frac{1}{r\sin\theta}\frac{\partial v_\phi}{\partial \phi}\right)F_1\right) \\
&+ \frac{\partial^2}{\partial r^2}(\kappa_{rr}F_1) + \frac{\partial^2}{\partial r\partial\theta}\left(\frac{\kappa_{r\theta}}{r}F_1\right) + \frac{\partial^2}{\partial r\partial\phi}\left(\frac{\kappa_{r\phi}}{r\sin\theta}F_1\right) \\
&+ \frac{\partial^2}{\partial r\partial\theta}\left(\frac{\kappa_{r\theta}}{r}F_1\right) + \frac{\partial^2}{\partial\theta^2}\left(\frac{\kappa_{\theta\theta}}{r^2}F_1\right) + \frac{\partial^2}{\partial\theta\partial\phi}\left(\frac{\kappa_{\theta\phi}}{r^2\sin\theta}F_1\right) \\
&+ \frac{\partial^2}{\partial r\partial\phi}\left(\frac{\kappa_{r\phi}}{r\sin\theta}F_1\right) + \frac{\partial^2}{\partial\theta\partial\phi}\left(\frac{\kappa_{\theta\phi}}{r^2\sin\theta}F_1\right) + \frac{\partial^2}{\partial\phi^2}\left(\frac{\kappa_{\phi\phi}}{r^2\sin^2\theta}F_1\right)
\end{aligned}$$

This corresponds to a set of SDEs.

$$\begin{aligned}
dr &= \left(v_r + v_{dr} + \frac{\partial \kappa_{rr}}{\partial r} + \frac{2}{r}\kappa_{rr} + \frac{1}{r}\frac{\partial \kappa_{r\theta}}{\partial \theta} + \frac{\cos\theta}{r\sin\theta}\kappa_{r\theta} + \frac{1}{r\sin\theta}\frac{\partial \kappa_{r\phi}}{\partial \phi}\right)dt + [P.dW_t]_r \\
d\theta &= \left(\frac{v_\theta + v_{d\theta}}{r} + \frac{1}{r}\frac{\partial \kappa_{r\theta}}{\partial r} + \frac{\kappa_{r\theta}}{r^2} + \frac{1}{r^2}\frac{\partial \kappa_{\theta\theta}}{\partial \theta} + \frac{\cos\theta}{r^2\sin\theta}\kappa_{\theta\theta} + \frac{1}{r^2\sin\theta}\frac{\partial \kappa_{\theta\phi}}{\partial \phi}\right)dt + [P.dW_t]_\theta \\
d\phi &= \left(\frac{v_\phi + v_{d\phi}}{r\sin\theta} + \frac{1}{r\sin\theta}\frac{\partial \kappa_{r\phi}}{\partial r} + \frac{\kappa_{r\phi}}{r^2\sin\theta} + \frac{1}{r^2\sin\theta}\frac{\partial \kappa_{\theta\phi}}{\partial \theta} + \frac{1}{r^2\sin^2\theta}\frac{\partial \kappa_{\phi\phi}}{\partial \phi}\right)dt + [P.dW_t]_\phi \\
dp &= -\frac{p}{3}\left(\frac{\partial v_r}{\partial r} + \frac{2v_r}{r} + \frac{1}{r}\frac{\partial v_\theta}{\partial \theta} + \frac{\cos\theta}{r\sin\theta}v_\theta + \frac{1}{r\sin\theta}\frac{\partial v_\phi}{\partial \phi}\right)
\end{aligned}$$

where

$$PP^T = \begin{bmatrix} 2\kappa_{rr} & \frac{2\kappa_{r\theta}}{r} & \frac{2\kappa_{r\phi}}{r\sin\theta} \\ \frac{2\kappa_{r\theta}}{r} & \frac{2\kappa_{\theta\theta}}{r^2} & \frac{2\kappa_{\theta\phi}}{r^2\sin\theta} \\ \frac{2\kappa_{r\phi}}{r\sin\theta} & \frac{2\kappa_{\theta\phi}}{r^2\sin\theta} & \frac{2\kappa_{\phi\phi}}{r^2\sin^2\theta} \end{bmatrix}$$

According to [Pei10], one possibility for P is

$$\begin{bmatrix} \sqrt{\frac{\kappa_{rr}\kappa_{\theta\phi}^2 + \kappa_{\theta\theta}\kappa_{r\phi}^2 + \kappa_{\phi\phi}\kappa_{r\theta}^2 - 2\kappa_{r\phi}\kappa_{r\theta}\kappa_{\theta\phi} - \kappa_{rr}\kappa_{\theta\theta}\kappa_{\phi\phi}}{0.5(\kappa_{\theta\phi}^2 - \kappa_{\theta\theta}\kappa_{\phi\phi})}} & \frac{\kappa_{r\phi}\kappa_{\theta\phi} - \kappa_{r\theta}\kappa_{\phi\phi}}{\kappa_{\theta\phi}^2 - \kappa_{\theta\theta}\kappa_{\phi\phi}}\sqrt{2\kappa_{\theta\theta} - \frac{2\kappa_{\theta\phi}^2}{\kappa_{\phi\phi}}} & \frac{\sqrt{2}\kappa_{r\phi}}{\sqrt{\kappa_{\phi\phi}}} \\ 0 & \frac{\sqrt{2\left(\kappa_{\theta\theta} - \kappa_{\theta\phi}^2/\kappa_{\phi\phi}\right)}}{r} & \frac{\kappa_{\theta\phi}}{r}\sqrt{\frac{2}{\kappa_{\phi\phi}}} \\ 0 & 0 & \frac{\sqrt{2\kappa_{\phi\phi}}}{r\sin\theta} \end{bmatrix}$$

For 1D probelms, $F_1 = fp^2r^2$, and the corresponding SDEs are

$$\begin{aligned}
dr &= \left(v_r + \frac{\partial \kappa_{rr}}{\partial r} + \frac{2}{r}\kappa_{rr}\right)dt + \sqrt{2\kappa_{rr}}dW_t \\
dp &= -\frac{p}{3}\left(\frac{\partial v_r}{\partial r} + \frac{2v_r}{r}\right)
\end{aligned}$$

For 2D problems, $F_1 = fp^2r^2\sin\theta$, and the corresponding SDEs are

$$dr = \left(v_r + \frac{\partial\kappa_{rr}}{\partial r} + \frac{2}{r}\kappa_{rr} + \frac{1}{r}\frac{\partial\kappa_{r\theta}}{\partial\theta} + \frac{\cos\theta}{r\sin\theta}\kappa_{r\theta}\right)dt + [P.dW_t]_r$$
$$d\theta = \left(\frac{v_\theta}{r} + \frac{1}{r}\frac{\partial\kappa_{r\theta}}{\partial r} + \frac{\kappa_{r\theta}}{r^2} + \frac{1}{r^2}\frac{\partial\kappa_{\theta\theta}}{\partial\theta} + \frac{\cos\theta}{r^2\sin\theta}\kappa_{\theta\theta}\right)dt + [P.dW_t]_\theta$$
$$dp = -\frac{p}{3}\left(\frac{\partial v_r}{\partial r} + \frac{2v_r}{r} + \frac{1}{r}\frac{\partial v_\theta}{\partial\theta} + \frac{\cos\theta}{r\sin\theta}v_\theta\right)$$

where

$$PP^T = \begin{bmatrix} 2\kappa_{rr} & \frac{2\kappa_{r\theta}}{r} \\ \frac{2\kappa_{r\theta}}{r} & \frac{2\kappa_{\theta\theta}}{r^2} \end{bmatrix}$$

One possibility for P is

$$\begin{bmatrix} -\frac{Q_{--}\sqrt{-Q_{-+}}}{\sqrt{Q_{--}^2+4b^2}} & \frac{Q_{+-}\sqrt{Q_{++}}}{\sqrt{Q_{+-}^2+4b^2}} \\ \frac{2b\sqrt{-Q_{-+}}}{\sqrt{Q_{--}^2+4b^2}} & \frac{2b\sqrt{Q_{++}}}{\sqrt{Q_{+-}^2+4b^2}} \end{bmatrix}$$

where

$$Q_{++} = \sqrt{(a-c)^2+4b^2} + (a+c)$$
$$Q_{-+} = \sqrt{(a-c)^2+4b^2} - (a+c)$$
$$Q_{+-} = \sqrt{(a-c)^2+4b^2} + (a-c)$$
$$Q_{--} = \sqrt{(a-c)^2+4b^2} - (a-c)$$

where $a = \kappa_{rr}$, $b = \kappa_{r\theta}/r$, $c = \kappa_{\theta\theta}/r^2$.

4.3 Focused Transport Equation

4.3.1 The Equation Solved

Warning: This part is still in early development. Please use it with caution.

Note: It is better to read the section on Parker's transport equation first since some of the details are skipped here.

The focused transport equation is [Zank14]

$$\frac{\partial f}{\partial t} + (U_i + c\mu b_i)\frac{\partial f}{\partial x_i} + \frac{dc}{dt}\frac{\partial f}{\partial c} + \frac{d\mu}{dt}\frac{\partial f}{\partial \mu} = \left\langle \frac{\delta f}{\delta t}\bigg|_s \right\rangle$$
$$\frac{dc}{dt} = \left[\frac{1-3\mu^2}{2}b_i b_j\frac{\partial U_i}{\partial x_j} - \frac{1-\mu^2}{2}\nabla\cdot\boldsymbol{U} - \frac{\mu b_i}{c}\left(\frac{\partial U_i}{\partial t} + U_j\frac{\partial U_i}{\partial x_j}\right)\right]c$$
$$\frac{d\mu}{dt} = \frac{1-\mu^2}{2}\left[c\nabla\cdot\boldsymbol{b} + \mu\nabla\cdot\boldsymbol{U} - 3\mu b_i b_j\frac{\partial U_i}{\partial x_j} - \frac{2b_i}{c}\left(\frac{\partial U_i}{\partial t} + U_j\frac{\partial U_i}{\partial x_j}\right)\right]$$

where $\boldsymbol{c}$ is the particle velocity in the flow frame ($\boldsymbol{v} = \boldsymbol{c}+\boldsymbol{U}$), $\mu \equiv \cos\theta = \boldsymbol{c}\cdot\boldsymbol{b}/c$ is the particle pitch angle, $\boldsymbol{b} \equiv \boldsymbol{B}/B$ is the unit vector along the magnetic field. The following focused transport equation is often used [Zhang09] [Zuo13] [Zhang17] [Kong22],

$$\frac{\partial f}{\partial t} = \nabla\cdot(\boldsymbol{\kappa}_\perp\nabla f) - (v\mu\boldsymbol{b} + \boldsymbol{V} + \boldsymbol{V}_d)\cdot\nabla f + \frac{\partial}{\partial\mu}\left(D_{\mu\mu}\frac{\partial f}{\partial\mu}\right) - \frac{d\mu}{dt}\frac{\partial f}{\partial\mu} - \frac{dp}{dt}\frac{\partial f}{\partial p} \tag{4.4}$$

where the terms on the right-hand side are cross-field spatial diffusion with a tensor $\boldsymbol{\kappa}_\perp = \kappa_\perp\left(\bar{\bar{\mathbf{I}}} - \boldsymbol{bb}\right)$ [Zhang09], streaming along the ambient or average magnetic field direction $\boldsymbol{b}$ with particle speed v and pitch-angle cosine μ, convection with the background plasma $\boldsymbol{V}$, partial gradient/curvature drift $\boldsymbol{V}_d$, pitch-angle diffusion with a coefficient $D_{\mu\mu}$, focusing $d\mu/dt$, and adiabatic heating/cooling dp/dt. In the adiabatic approximation, the drift velocity, focusing rate, and cooling rate may be calculated from the ambient magnetic field $\boldsymbol{B} = B\boldsymbol{b}$ and plasma velocity $\boldsymbol{V}$ through

$$\begin{aligned}
\boldsymbol{V}_d &= \frac{cpv}{qB}\left\{\frac{1-\mu^2}{2}\frac{\boldsymbol{B}\times\nabla B}{B^2} + \mu^2\frac{\boldsymbol{B}\times[(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}]}{B^3} + \frac{1-\mu^2}{2}\frac{\boldsymbol{B}(\boldsymbol{B}\cdot\nabla\times\boldsymbol{B})}{B^3}\right\}\\
\frac{d\mu}{dt} &= \frac{1-\mu^2}{2}\left[-v\boldsymbol{b}\cdot\nabla\ln B + \mu\nabla\cdot\boldsymbol{V} - 3\mu b_i b_j\frac{\partial V_i}{\partial x_j} - \frac{2b_i}{v}\frac{dV_i}{dt}\right]\\
\frac{dp}{dt} &= -p\left[\frac{1-\mu^2}{2}\left(\nabla\cdot\boldsymbol{V} - b_i b_j\frac{\partial V_i}{\partial x_j}\right) + \mu^2 b_i b_j\frac{\partial V_i}{\partial x_j} + \frac{\mu b_i}{v}\frac{dV_i}{dt}\right]
\end{aligned}$$

where $\boldsymbol{V}_d$ includes gradient, curvature, and parallel drifts. We may ignore $\partial V_i/\partial t$ in dV_i/dt if the flow is not dramatically evolving.

In the quasi-linear theory, resonant interaction between the particle and the turbulent magnetic field can be related by the pitch-angle diffusion coefficient $D_{\mu\mu}$ [Jokipii77] [Kong22]

$$D_{\mu\mu} = \frac{\pi}{4}\Omega_0(1-\mu^2)\frac{k_{\rm res}P(k_{\rm res})}{B_0^2}$$

where Ω_0 is particle gyrofrequency, $k_{\rm res} = |\Omega_0/v\mu|$ is the resonant wavenumber. The above equation is strictly applicable only for the case of 1D turbulence in which the wavevectors are aligned with the mean field. For anisotropic turbulence (e.g., 2D + slab), only the slab turbulence (about 20% of all turbulent fluctuations [Bieber96]) affect particle parallel transport [Florinski03]. The turbulence power is usually expressed as

$$P(k) = \frac{\langle\delta B^2\rangle}{1+(kL_c)^\gamma}\left[\int_{k_{\rm min}}^{k_{\rm max}}\frac{dk}{1+(kL_c)^\gamma}\right]^{-1}$$

where $k_{\rm min}$ and $k_{\rm max}$ are the smallest and largest wavenumbers in the system, L_c is the turbulence correlation length, and γ is the turbulence spectrum index (e.g., 5/3). For $k_{\rm min} \ll 1/L_c \ll k_{\rm max}$, the integral in the above equation can be taken from 0 to ∞. From the table of integral, $\int_0^\infty x^{\mu-1}dx/(1+x^\nu) = \pi\csc(\mu\pi/\nu)/\nu$. Then,

$$P(k) = \frac{\langle\delta B^2\rangle L_c}{1+(kL_c)^\gamma}A_0$$

where $A_0 = [(\pi/\gamma)\csc(\pi/\gamma)]^{-1}$. In the nonrelativistic limit, we take the pitch-angle diffusion coefficient in the form of [Kong22]

$$D_{\mu\mu} = D_{\mu\mu0}\left(\frac{p}{p_0}\right)^{\gamma-1}(1-\mu^2)(|\mu|^{\gamma-1}+h_0)$$

where $D_{\mu\mu0} = \frac{\pi}{4}A_0\sigma^2\Omega_0^{2-\gamma}L_c^{1-\gamma}v_0^{\gamma-1}$, and $p_0(v_0)$ is the initial particle momentum (velocity) at the injection energy. The parameter h_0 is added to describe the scattering through $\mu = 0$, and we set $h_0 = 0.2$ [Zhang17] [Kong22].

$$\frac{\partial D_{\mu\mu}}{\partial\mu} = D_{\mu\mu0}\left(\frac{p}{p_0}\right)^{\gamma-1}\left[-2\mu(|\mu|^{\gamma-1}+h_0) + (1-\mu^2)\text{sign}(\mu)|\mu|^{\gamma-2}\right]$$

4.3.2 Cartesian Coordinates

The Fokker-Planck form of the focused transport equation is

$$\begin{aligned}\frac{\partial F}{\partial t} =& \nabla \cdot [\nabla \cdot (\boldsymbol{\kappa}_\perp \nabla F)] - \nabla \cdot [(v\mu \boldsymbol{b} + \boldsymbol{V} + \boldsymbol{V}_d + \nabla \cdot \boldsymbol{\kappa}_\perp) F] \\ &+ \frac{\partial}{\partial \mu^2}\left(D_{\mu\mu} F\right) - \frac{\partial}{\partial \mu}\left[\left(\frac{d\mu}{dt} + \frac{\partial D_{\mu\mu}}{\partial \mu}\right) F\right] - \frac{\partial}{\partial p}\left(\frac{dp}{dt} F\right)\end{aligned}$$

where $F = fp^2$. The corresponding SDEs are

$$\begin{aligned} d\boldsymbol{X} &= (v\mu \boldsymbol{b} + \boldsymbol{V} + \boldsymbol{V}_d + \nabla \cdot \boldsymbol{\kappa}_\perp) dt + \sum_\sigma \boldsymbol{\alpha}_\sigma dW_\sigma(s) \\ dp &= \left(\frac{dp}{dt}\right) dt \\ d\mu &= \left(\frac{d\mu}{dt} + \frac{\partial D_{\mu\mu}}{\partial \mu}\right) dt + \sqrt{2D_{\mu\mu}} dW_\mu(t)\end{aligned}$$

where $\sum_\sigma \alpha_\sigma^\mu \alpha_\sigma^\nu = 2\kappa_\perp^{\mu\nu}$. We can use the results from the 3D model with whole spatial diffusion tensor $\boldsymbol{\kappa}$ but set $\kappa_\parallel = 0$. The matrix

$$P = \begin{pmatrix} 0 & -b_x b_z \sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} & -b_y \sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} \\ 0 & -b_y b_z \sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} & b_x \sqrt{2\kappa_\perp}/\sqrt{b_x^2 + b_y^2} \\ 0 & \sqrt{b_x^2 + b_y^2}\sqrt{2\kappa_\perp} & 0 \end{pmatrix}$$

Since the first column is all zeros, we only need two Weiner processes to describe the spatial diffusion. For 2D problems,

$$P = \frac{\sqrt{2\kappa_\perp}}{\sqrt{b_x^2 + b_y^2}} \begin{pmatrix} -b_x b_z & -b_y \\ -b_y b_z & b_x \end{pmatrix}$$

To calculate the drift velocity

$$\boldsymbol{V}_d = \frac{cpv}{qB}\left\{\frac{1-\mu^2}{2}\frac{\boldsymbol{B}\times\nabla B}{B^2} + \mu^2 \frac{\boldsymbol{B}\times[(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}]}{B^3} + \frac{1-\mu^2}{2}\frac{\boldsymbol{B}(\boldsymbol{B}\cdot\nabla\times\boldsymbol{B})}{B^3}\right\},$$

we need

$$\begin{aligned}(\boldsymbol{B}\times\nabla B)_x &= B_y\partial_z B - B_z\partial_y B \\ (\boldsymbol{B}\times\nabla B)_y &= B_z\partial_x B - B_x\partial_z B \\ (\boldsymbol{B}\times\nabla B)_z &= B_x\partial_y B - B_y\partial_x B \\ \{\boldsymbol{B}\times[(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}]\}_x &= B_y(\boldsymbol{B}\cdot\nabla)B_z - B_z(\boldsymbol{B}\cdot\nabla)B_y \\ \{\boldsymbol{B}\times[(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}]\}_y &= B_z(\boldsymbol{B}\cdot\nabla)B_x - B_x(\boldsymbol{B}\cdot\nabla)B_z \\ \{\boldsymbol{B}\times[(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}]\}_z &= B_x(\boldsymbol{B}\cdot\nabla)B_y - B_y(\boldsymbol{B}\cdot\nabla)B_x \\ \boldsymbol{B}\cdot\nabla &= B_x\partial_x + B_y\partial_y + B_z\partial_z \\ \boldsymbol{B}\cdot\nabla\times\boldsymbol{B} &= B_x(\partial_y B_z - \partial_z B_y) + \\ &\quad B_y(\partial_z B_x - \partial_x B_z) + B_z(\partial_x B_y - \partial_y B_x)\end{aligned}$$

For 2D problems, these can be simplified using

$$\begin{aligned}(\boldsymbol{B}\times\nabla B)_x &= -B_z\partial_y B \\ (\boldsymbol{B}\times\nabla B)_y &= B_z\partial_x B \\ (\boldsymbol{B}\times\nabla B)_z &= B_x\partial_y B - B_y\partial_x B \\ \boldsymbol{B}\cdot\nabla &= B_x\partial_x + B_y\partial_y \\ \boldsymbol{B}\cdot\nabla\times\boldsymbol{B} &= B_x\partial_y B_z - B_y\partial_x B_z + B_z(\partial_x B_y - \partial_y B_x)\end{aligned}$$

4.3.3 Spherical Coordinates

The spatial diffusion coefficient is in the same form.

$$\boldsymbol{\kappa}_\perp = \begin{bmatrix} \kappa_{\perp rr} & \kappa_{\perp r\theta} & \kappa_{\perp r\phi} \\ \kappa_{\perp r\theta} & \kappa_{\perp\theta\theta} & \kappa_{\perp\theta\phi} \\ \kappa_{\perp r\phi} & \kappa_{\perp\theta\phi} & \kappa_{\perp\phi\phi} \end{bmatrix}$$

where $\kappa_{\perp ij} = \kappa_\perp \delta_{ij} - \kappa_\perp b_i b_j$, and i, j are r, θ, ϕ. The gradients of κ_{ij} are

$$\partial_i \kappa_{\perp ij} = \delta_{ij}\partial_i\kappa_\perp - b_i b_j \partial_i \kappa_\perp - \frac{\kappa_\perp}{B}(b_j \partial_i B_i + b_i \partial_i B_j - 2 b_i b_j \partial_i B)$$

We then need to transfer the focused transport equation to the spherical coordinates. Since we don't have cross-diffusion terms (spatial and momentum), we can ignore the momentum diffusion for now.

$$\begin{aligned} \frac{\partial F}{\partial t} =& \nabla \cdot (\boldsymbol{\kappa}_\perp \cdot \nabla F) - \nabla \cdot [(v\mu \boldsymbol{b} + \boldsymbol{V} + \boldsymbol{V}_d) F] \\ &+ \frac{\partial}{\partial \mu^2}(D_{\mu\mu} F) - \frac{\partial}{\partial \mu}\left[\left(\frac{d\mu}{dt} + \frac{\partial D_{\mu\mu}}{\partial \mu}\right) F\right] - \frac{\partial}{\partial p}\left(\frac{dp}{dt} F\right) \end{aligned}$$

The rest is essentially the same as before except for $\boldsymbol{\kappa}_\perp$, $v\mu\boldsymbol{b}$, and the pitch-angle diffusion terms. We will first need to change F to $F_1 = F \sin\theta r^2$ [Jokipii77] [Pei10]. The resulting SDEs are

$$\begin{aligned} dr &= \frac{dr}{dt}dt + [P.dW_t]_r \\ d\theta &= \frac{d\theta}{dt}dt + [P.dW_t]_\theta \\ d\phi &= \frac{d\phi}{dt}dt + [P.dW_t]_\phi \\ dp &= \frac{dp}{dt}dt \\ d\mu &= \left(\frac{d\mu}{dt} + \frac{\partial D_{\mu\mu}}{\partial \mu}\right)dt + \sqrt{2D_{\mu\mu}}dW_\mu(t) \end{aligned}$$

where

$$\begin{aligned} \frac{dr}{dt} &= v\mu b_r + V_r + V_{dr} + \frac{\partial \kappa_{\perp rr}}{\partial r} + \frac{2}{r}\kappa_{\perp rr} + \frac{1}{r}\frac{\partial \kappa_{\perp r\theta}}{\partial \theta} + \frac{\cos\theta}{r\sin\theta}\kappa_{\perp r\theta} + \frac{1}{r\sin\theta}\frac{\partial \kappa_{\perp r\phi}}{\partial \phi} \\ \frac{d\theta}{dt} &= \frac{v\mu b_\theta + V_\theta + V_{d\theta}}{r} + \frac{1}{r}\frac{\partial \kappa_{\perp r\theta}}{\partial r} + \frac{\kappa_{\perp r\theta}}{r^2} + \frac{1}{r^2}\frac{\partial \kappa_{\perp\theta\theta}}{\partial \theta} + \frac{\cos\theta}{r^2\sin\theta}\kappa_{\perp\theta\theta} + \frac{1}{r^2\sin\theta}\frac{\partial \kappa_{\perp\theta\phi}}{\partial \phi} \\ \frac{d\phi}{dt} &= \frac{v\mu b_\phi + V_\phi + V_{d\phi}}{r\sin\theta} + \frac{1}{r\sin\theta}\frac{\partial \kappa_{\perp r\phi}}{\partial r} + \frac{\kappa_{\perp r\phi}}{r^2\sin\theta} + \frac{1}{r^2\sin\theta}\frac{\partial \kappa_{\perp\theta\phi}}{\partial \theta} + \frac{1}{r^2\sin^2\theta}\frac{\partial \kappa_{\perp\phi\phi}}{\partial \phi} \end{aligned}$$

To calculate $d\mu/dt$ and dp/dt, we will need

$$\begin{aligned} -\boldsymbol{b}\cdot\nabla \ln B &= -\frac{1}{B}\left(b_r\partial_r B + \frac{b_\theta}{r}\partial_\theta B + \frac{b_\phi}{r\sin\theta}\partial_\phi B\right) \\ \nabla\cdot\boldsymbol{V} &= \partial_r V_r + \frac{2V_r}{r} + \frac{1}{r}\partial_\theta V_\theta + \frac{\cos\theta}{r\sin\theta}V_\theta + \frac{1}{r\sin\theta}\partial_\phi V_\phi \end{aligned}$$

$b_i b_j \frac{\partial V_i}{\partial x_j}$ is more complicated.

$$\begin{aligned} b_i b_j \frac{\partial V_i}{\partial x_j} =& b_r^2 \partial_r V_r + \frac{b_r b_\theta}{r}\partial_\theta V_r + \frac{b_r b_\phi}{r\sin\theta}\partial_\phi V_r - \frac{b_r b_\theta V_\theta + b_r b_\phi V_\phi}{r} + \\ & b_r b_\theta \partial_r V_\theta + \frac{b_\theta^2}{r}\partial_\theta V_\theta + \frac{b_\theta b_\phi}{r\sin\theta}\partial_\phi V_\theta + \frac{b_\theta^2 V_r}{r} - \frac{\cot\theta b_\theta b_\phi V_\phi}{r} + \\ & b_r b_\phi \partial_r V_\phi + \frac{b_\theta b_\phi}{r}\partial_\theta V_\phi + \frac{b_\phi^2}{r\sin\theta}\partial_\phi V_\phi + \frac{b_\phi^2 V_r}{r} + \frac{\cot\theta b_\phi^2 V_\theta}{r} \end{aligned}$$

Similar for $b_i V_j \frac{\partial V_i}{\partial x_j}$,

$$\begin{aligned} b_i V_j \frac{\partial V_i}{\partial x_j} =& b_r V_r \partial_r V_r + \frac{b_r V_\theta}{r}\partial_\theta V_r + \frac{b_r V_\phi}{r\sin\theta}\partial_\phi V_r - \frac{b_r(V_\theta^2 + V_\phi^2)}{r} + \\ & b_\theta V_r \partial_r V_\theta + \frac{b_\theta V_\theta}{r}\partial_\theta V_\theta + \frac{b_\theta V_\phi}{r\sin\theta}\partial_\phi V_\theta + \frac{b_\theta V_\theta V_r}{r} - \frac{\cot\theta b_\theta V_\phi^2}{r} + \\ & b_\phi V_r \partial_r V_\phi + \frac{b_\phi V_\theta}{r}\partial_\theta V_\phi + \frac{b_\phi V_\phi}{r\sin\theta}\partial_\phi V_\phi + \frac{b_\phi V_\phi V_r}{r} + \frac{\cot\theta b_\phi V_\phi V_\theta}{r} \end{aligned}$$

For 1D probelms, $F_1 = fp^2r^2$, and the corresponding SDE for r is

$$dr = \left(v\mu b_r + V_r + \frac{\partial \kappa_{\perp rr}}{\partial r} + \frac{2}{r}\kappa_{\perp rr} \right) dt + \sqrt{2\kappa_{\perp rr}} dW_t$$

For $d\mu$ and dp, we will need

$$\begin{aligned} -\boldsymbol{b}\cdot\nabla \ln B =& -\frac{1}{B} b_r \partial_r B \\ \nabla\cdot\boldsymbol{V} =& \partial_r V_r + \frac{2V_r}{r} \\ b_i b_j \frac{\partial V_i}{\partial x_j} =& b_r^2 \partial_r V_r - \frac{b_r b_\theta V_\theta + b_r b_\phi V_\phi}{r} + \\ & b_r b_\theta \partial_r V_\theta + \frac{b_\theta^2 V_r}{r} - \frac{\cot\theta b_\theta b_\phi V_\phi}{r} + \\ & b_r b_\phi \partial_r V_\phi + \frac{b_\phi^2 V_r}{r} + \frac{\cot\theta b_\phi^2 V_\theta}{r} \\ b_i V_j \frac{\partial V_i}{\partial x_j} =& b_r V_r \partial_r V_r - \frac{b_r(V_\theta^2 + V_\phi^2)}{r} + \\ & b_\theta V_r \partial_r V_\theta + \frac{b_\theta V_\theta V_r}{r} - \frac{\cot\theta b_\theta V_\phi^2}{r} + \\ & b_\phi V_r \partial_r V_\phi + \frac{b_\phi V_\phi V_r}{r} + \frac{\cot\theta b_\phi V_\phi V_\theta}{r} \end{aligned}$$

For 2D problems ($r-\theta$ plane), $F_1 = fp^2r^2\sin\theta$, and the corresponding SDEs are

$$\begin{aligned} dr &= \left(v\mu b_r + V_r + \frac{\partial \kappa_{\perp rr}}{\partial r} + \frac{2}{r}\kappa_{\perp rr} + \frac{1}{r}\frac{\partial \kappa_{\perp r\theta}}{\partial \theta} + \frac{\cos\theta}{r\sin\theta}\kappa_{\perp r\theta} \right) dt + [P.dW_t]_r \\ d\theta &= \left(\frac{v\mu b_\theta + V_\theta}{r} + \frac{1}{r}\frac{\partial \kappa_{\perp r\theta}}{\partial r} + \frac{\kappa_{\perp r\theta}}{r^2} + \frac{1}{r^2}\frac{\partial \kappa_{\perp\theta\theta}}{\partial\theta} + \frac{\cos\theta}{r^2\sin\theta}\kappa_{\perp\theta\theta} \right) dt + [P.dW_t]_\theta \end{aligned}$$

where P has the same form as the one in Parker transport. For $d\mu$ and dp, we will need

$$\begin{aligned}
-\boldsymbol{b}\cdot\nabla\ln B =& -\frac{1}{B}\left(b_r\partial_r B+\frac{b_\theta}{r}\partial_\theta B\right)\\
\nabla\cdot\boldsymbol{V} =& \partial_r V_r+\frac{2V_r}{r}+\frac{1}{r}\partial_\theta V_\theta+\frac{\cos\theta}{r\sin\theta}V_\theta\\
b_ib_j\frac{\partial V_i}{\partial x_j} =& b_r^2\partial_r V_r+\frac{b_rb_\theta}{r}\partial_\theta V_r-\frac{b_rb_\theta V_\theta+b_rb_\phi V_\phi}{r}+\\
& b_rb_\theta\partial_r V_\theta+\frac{b_\theta^2}{r}\partial_\theta V_\theta+\frac{b_\theta^2V_r}{r}-\frac{\cot\theta b_\theta b_\phi V_\phi}{r}+\\
& b_rb_\phi\partial_r V_\phi+\frac{b_\theta b_\phi}{r}\partial_\theta V_\phi+\frac{b_\phi^2V_r}{r}+\frac{\cot\theta b_\phi^2V_\theta}{r}\\
b_iV_j\frac{\partial V_i}{\partial x_j} =& b_rV_r\partial_r V_r+\frac{b_rV_\theta}{r}\partial_\theta V_r-\frac{b_r(V_\theta^2+V_\phi^2)}{r}+\\
& b_\theta V_r\partial_r V_\theta+\frac{b_\theta V_\theta}{r}\partial_\theta V_\theta+\frac{b_\theta V_\theta V_r}{r}-\frac{\cot\theta b_\theta V_\phi^2}{r}+\\
& b_\phi V_r\partial_r V_\phi+\frac{b_\phi V_\theta}{r}\partial_\theta V_\phi+\frac{b_\phi V_\phi V_r}{r}+\frac{\cot\theta b_\phi V_\phi V_\theta}{r}
\end{aligned}$$

The drift velocity

$$\boldsymbol{V}_d=\frac{cpv}{qB}\left\{\frac{1-\mu^2}{2}\frac{\boldsymbol{B}\times\nabla B}{B^2}+\mu^2\frac{\boldsymbol{B}\times[(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}]}{B^3}+\frac{1-\mu^2}{2}\frac{\boldsymbol{B}(\boldsymbol{B}\cdot\nabla\times\boldsymbol{B})}{B^3}\right\}$$

Calculations needed for the first term (gradient drift):

$$\begin{aligned}
&(\nabla B)_r=\partial_r B;\quad (\nabla B)_\theta=\frac{\partial_\theta B}{r};\quad (\nabla B)_\phi=\frac{\partial_\phi B}{r\sin\theta}\\
&(\boldsymbol{B}\times\nabla B)_r=B_\theta(\nabla B)_\phi-B_\phi(\nabla B)_\theta\\
&(\boldsymbol{B}\times\nabla B)_\theta=B_\phi(\nabla B)_r-B_r(\nabla B)_\phi\\
&(\boldsymbol{B}\times\nabla B)_\phi=B_r(\nabla B)_\theta-B_\theta(\nabla B)_r
\end{aligned}$$

Calculations needed for the second term (curvature drift):

$$\begin{aligned}
&\boldsymbol{C}=(\boldsymbol{B}\cdot\nabla)\boldsymbol{B}\\
&C_r=B_r\partial_r B_r+\frac{B_\theta}{r}\partial_\theta B_r+\frac{B_\phi}{r\sin\theta}\partial_\phi B_r-\frac{B_\theta^2+B_\phi^2}{r}\\
&C_\theta=B_r\partial_r B_\theta+\frac{B_\theta}{r}\partial_\theta B_\theta+\frac{B_\phi}{r\sin\theta}\partial_\phi B_\theta+\frac{B_rB_\theta}{r}-\frac{\cot\theta B_\phi^2}{r}\\
&C_\phi=B_r\partial_r B_\phi+\frac{B_\theta}{r}\partial_\theta B_\phi+\frac{B_\phi}{r\sin\theta}\partial_\phi B_\phi+\frac{B_rB_\phi}{r}+\frac{\cot\theta B_\theta B_\phi}{r}\\
&(\boldsymbol{B}\times\boldsymbol{C})_r=B_\theta C_\phi-B_\phi C_\theta\\
&(\boldsymbol{B}\times\boldsymbol{C})_\theta=B_\phi C_r-B_r C_\phi\\
&(\boldsymbol{B}\times\boldsymbol{C})_\phi=B_r C_\theta-B_\theta C_r
\end{aligned}$$

Calculations needed for the third term (parallel drift):

$$\begin{aligned}
&(\nabla\times\boldsymbol{B})_r=\frac{\partial_\theta(\sin\theta B_\phi)}{r\sin\theta}-\frac{\partial_\phi B_\theta}{r\sin\theta}=\frac{\partial_\theta B_\phi}{r}+\frac{\cos\theta}{r\sin\theta}B_\phi-\frac{\partial_\phi B_\theta}{r\sin\theta}\\
&(\nabla\times\boldsymbol{B})_\theta=\frac{\partial_\phi B_r}{r\sin\theta}-\frac{\partial_r(rB_\phi)}{r}=\frac{\partial_\phi B_r}{r\sin\theta}-\partial_r B_\phi-\frac{B_\phi}{r}\\
&(\nabla\times\boldsymbol{B})_\phi=\frac{\partial_r(rB_\theta)}{r}-\frac{\partial_\theta B_r}{r}=\partial_r B_\theta+\frac{B_\theta}{r}-\frac{\partial_\theta B_r}{r}
\end{aligned}$$

CHAPTER

FIVE

DEVELOPMENT

5.1 Implementation Details

This section provides an overview of the code implementation details. The code solves particle transport equations using stochastic differential equations (SDEs). To solve the SDEs, the code uses pseudo particles to sample the particle distributions. The code is designed to be modular and scalable. It is parallelized using MPI and OpenMP and can handle large-scale 3D simulations. The code is organized into several modules, each of which is responsible for a specific aspect of the simulation. See the next section for more details. In general, the code needs to read MHD data, interpolate it at particle positions, evolve the particles, and produce diagnostics.

- Initialization Phase:
  - Set up the simulation parameters (`simulation_setup_module`, `mhd_config`, and `mpi_module`).
  - Read the MHD data from files (`mhd_data_parallel`)
  - Initialize the particle data (`particle_module`).
  - Initialize diagnostics (`diagnostics`).
- Main Simulation Loop:
  - Inject pseudo particles (`particle_module`).
  - Interpolate MHD data to particle positions (`mhd_data_parallel`).
  - Evolve particles using SDEs and MHD fields (`particle_module`, `random_number_generator`).
- Diagnostics and Data Handling:
  - Produce diagnostics of the simulation (`diagnostics`).
  - Handle data read/write operations (`hdf5_io`, `mpi_io`).

5.1.1 Module Structure

The code is organized into several modules to improve modularity and maintainability. The modules are saved in `src/modules` directory. These modules are used by `src/stochastic_mhd.f90` to perform the simulation. The main modules include:

Note: This section needs to be updated to include more details about each of these modules.

- `particle_module`: This module contains particle data and the methods to inject, remove, and push particles. All the core algorithms of SDE are implemented in this module.

- `mhd_data_parallel`: This module contains the data structures and methods to initialize and read MHD simulation data. It also contains the methods to calculate the gradients of these fields and to interpolate the MHD data to the particle positions. Additionally, this module also includes the methods for spatially dependent turbulence correlation length and turbulence amplitude.
- `simulation_setup_module`: This module contains the methods to set up the simulation parameters, such as the simulation MPI topology, the MHD field configuration, particle boundary conditions, and neighbors of local MPI domains.
- `hdf5_io`: This module contains the methods to create, open, close, read and write data in HDF5 format. It can deal with data up to 5D and with/without parallel I/O.
- `mhd_config`: This module contains the methods to read the MHD configuration file and to set up the MHD simulation parameters, including the grid and time stamps if needed.
- `constants`: This module contains constants used in the simulation.
- `diagnostics`: This module contains the methods to diagnose the simulations, such as the particle distribution functions, particle data, escaped particle data, and methods for quick checks as the simulation proceeds.
- `mpi_io`: This module contains the methods to read and write data in parallel using MPI-IO.
- `mpi_module`: This module contains the methods to set up the MPI topology and to communicate between different MPI domains.
- `random_number_generator`: This module contains the methods to generate random numbers using the Mersenne Twister algorithm.
- `read_confg`: This module contains the methods to read the simulation configuration file.
- `acc_region_surface`: This module contains the methods for the surface to separate the acceleration region and the non-acceleration region. It is useful when we want to isolate certain particle acceleration regions, for example, near the shock front.

5.2 Particle Module

This section provides an overview of the particle module, including the particle data structure, the methods for injecting, removing, and pushing particles.

5.2.1 Particle Data Structure

```
type particle_type
    integer(i1)  :: split_times     !< Particle splitting times
    integer(i1)  :: count_flag      !< Only count particle when it is 1
    integer(i4)  :: origin          !< The origin MPI rank of the particle
    integer(i4)  :: nsteps_tracked  !< # of tracked steps in MHD data interval
    integer(i4)  :: nsteps_pushed   !< # of steps have been pushed
    integer(i4)  :: tag_injected    !< Particle tag 1 when injected
    integer(i4)  :: tag_splitted    !< Particle tag 2 after splitting
    real(dp) :: x, y, z, p          !< Position and momentum
    real(dp) :: v, mu               !< Velocity and cosine of pitch-angle
    real(dp) :: weight, t, dt       !< Particle weight, time and time step
end type particle_type
```

- `x`, `y`, `z`: position of the particle in the simulation domain. The positions are in the global coordinate system. For spherical coordinates, `x` is the radial distance r, `y` is the polar angle θ, and `z` is the azimuthal angle ϕ.

- `p`: momentum magnitude of the particle.
- `v`: velocity magnitude of the particle. We typically assume non-relativistic particles, so that $v \sim p$.
- `mu`: cosine of the pitch angle of the particle.
- `weight`: weight of the particle. The weight is initialized to be 1.0. When a particle reaches certain momentum, it will be splitted into two identical particles with half of the weight. Due to stochastic nature of the process, these particles will evolve differently after the splitting.
- `t`: time of the particle. The time is initialized to be the time when particle is injected.
- `dt`: time step of the particle. The time step is adaptive and is updated at each time step.
- `split_times`: number of times the particle has been splitted.
- `count_flag`: flag to count the particle. If the flag is 1, the particle will be counted when accumulating particle distributions. When the flag is negative (-1 to -6), it means that the particle has escaped from one of the six boundaries of the simulation domain (for 2D, -1 to -4 for the four boundaries). When the flag is 0, the particle has negative momentum and will be removed from the simulation.
- `origin`: the origin MPI rank of the particle. Combined with particle `tag_injected` and `tag_splitted`, it can be used to identify the particle for particle tracking.
- `nsteps_tracked`: number of steps the particle has been tracked in the MHD data interval. It will be reset to 0 at the start of the MHD data interval.
- `nsteps_pushed`: number of steps the particle has been pushed since they are injected. It is useful when tracking particles. It will accummulate to `nsteps_interval` and then reset to 0 once the tracked particle is recorded.
- `tag_injected`: tag of the particle when it is injected. The tag is unique on each MPI rank.
- `tag_splitted`: tag of the particle after it is splitted. The tag is unique on each MPI rank.

Note: The tag of a particle only unique on each MPI rank. When combined with the origin MPI rank, the particle can be uniquely identified in the simulation. Since `tag_splitted` is a integer4, the maximum number of particles that one particle can be splitted into is 2^31. If the number of particles exceeds this limit, it means that we probably need to use a larger `split_ratio`.

5.2.2 Particle Tracking

As described above, we can track particles using the combination of `tag_injected`, `tag_splitted`, and `origin`. The particle tracking is useful for diagnosing how the particles are accelerated and transported. To get the particle tracking information, we need to runt the simulation twice. The first run is a regular run. We need to run the simulation to a certain time frame and dump the particles (also included in the restart files). Then, we need to use a post-processing tool (e.g., a Python script) to select the particles we want to track and save the particle information to a file. The second run is a tracking run. We need to set `track_particle_flag = .true.` in the input file and set the `particle_tags_file` to the file that contains the particle information. The tracking run is similar to the regular run except that we need to load the `particle_tags_file` and use the information to identify the particles we want to track.

- When a particle is injected, the particle is assigned with an `origin` MPI rank and a `tag_injected`, which starts from 0 and increases by 1 for each particle injected on each MPI rank. When the particle is splitted, the `tag_splitted` is assigned to the particle. `tag_splitted` starts from 1. When one particle is splitted into two particles, each particle has a half of the original weight. One of the two particles has the same `tag_splitted` as the original particle, and the other has the `tag_splitted=tag_splitted_original+2**(split_times-1)`. The `split_times` is the number of times the particle has been splitted. The `tag_splitted` is used to identify the particle after it is splitted. The combination of `tag_injected`, `tag_splitted`, and `origin` can be used to uniquely identify the particle in the simulation.

- The first time we run the simulation, we need to set `track_particle_flag = .false.` in the input file. The particle information will be saved in the restart files. We can use the restart files to get the particle information. We can use a Python script to select the particles we want to track. For example, we can select the partilces with the highest energies form the particle file using the following script.

```
tframe = 200
run_name = sde_run_config["run_name"]
fname = "../data/" + run_name + "/restart/particles_" + str(tframe).zfill(4) + ".h5"
with h5py.File(fname, 'r') as fh:
    p = fh["p"][:]
    tag_injected = fh["tag_injected"][:]
    tag_splitted = fh["tag_splitted"][:]
    origin = fh["origin"][:]
    split_times = fh["split_times"][:]
sort_index = np.argsort(p)
psort = p[sort_index]
tag_injected_sort = tag_injected[sort_index]
tag_splitted_sort = tag_splitted[sort_index]
origin_sort = origin[sort_index]
split_times_sort = split_times[sort_index]
nptl_selected = 1000
# select the highest-energy particles
origin_s = origin_sort[-nptl_selected:]
tag_injected_s = tag_injected_sort[-nptl_selected:]
tag_splitted_s = tag_splitted_sort[-nptl_selected:]
split_times_s = split_times_sort[-nptl_selected:]
# trace back to the injected particle
nsplit_max = np.max(split_times_s)
tags_all = np.zeros([nptl_selected, nsplit_max+2], dtype=np.int32)
tags_all[:, 0] = origin_s  # the first col is the origin
tags_all[:, 1] = tag_injected_s  # the second col is tag_injected
for iptl in range(nptl_selected):
    nsplit = split_times_s[iptl]
    if nsplit > 0:
        tags_all[iptl, nsplit+1] = tag_splitted_s[iptl]
        for isplit in range(nsplit, 1, -1):
            if tags_all[iptl, isplit+1] > 2**(isplit-1):
                tags_all[iptl, isplit] = tags_all[iptl, isplit+1] - 2**(isplit-1)
            else:
                tags_all[iptl, isplit] = tags_all[iptl, isplit+1]
# sort by the origin, tag_injected, and then all tag_splitted
ind = np.lexsort(tags_all[:, ::-1].T)
tags_all_sorted = tags_all[ind]
# save the selected particles to a file
odir = "../data/" + run_name + "/particle_tracking/"
mkdir_p(odir)
fname = odir + "tags_selected_01.h5"
with h5py.File(fname, 'w') as fh:
    fh.create_dataset("tags", tags_all_sorted.shape, data=tags_all_sorted)
```

- The second time we run the simulation, we need to set `track_particle_flag = .true.` in the input file. We need to set the `particle_tags_file` to the file that contains the particle information. The tracking run is similar to the regular run except that we need to load the `particle_tags_file` and use the information to identify the particles we want to track.

- The simulation will output tracked particle information every MHD intervel. The files are saved in `particle_tracking` under the diagnostic directory. We can get all the particle information using a script like the following.

```
fdir = "../data/" + run_name + "/particle_tracking/"
ts, te = 51, 200
nframes = te - ts + 1
fname = fdir + "particles_tracked_" + str(ts).zfill(4) + ".h5"
with h5py.File(fname, "r") as fh:
    x = fh["x"][:, :]
nptl_tracking, nsteps_tracked = x.shape
t = np.zeros([nptl_tracking, nsteps_tracked*nframes])
x = np.zeros([nptl_tracking, nsteps_tracked*nframes])
y = np.zeros([nptl_tracking, nsteps_tracked*nframes])
p = np.zeros([nptl_tracking, nsteps_tracked*nframes])
tag_splitted = np.zeros([nptl_tracking, nsteps_tracked*nframes])
for tframe in range(ts, te+1):
    fname = fdir + "particles_tracked_" + str(tframe).zfill(4) + ".h5"
    it1 = (tframe - ts) * nsteps_tracked
    it2 = it1 + nsteps_tracked
    with h5py.File(fname, "r") as fh:
        t[:, it1:it2] = fh["t"][:, :]
        x[:, it1:it2] = fh["x"][:, :]
        y[:, it1:it2] = fh["y"][:, :]
        p[:, it1:it2] = fh["p"][:, :]
        tag_splitted[:, it1:it2] = fh["tag_splitted"][:, :]
```

To plot one particle trajectory, we can use, for example,

```
iptl = 100
ind = t[iptl, :] > 0
plt.plot(x[iptl, ind], y[iptl, ind])
```

CHAPTER

SIX

EPILOGUE

6.1 Publications

- Modeling Electron Acceleration and Transport in the Early Impulsive Phase of the 2017 September 10th Solar Flare, Xiaocan Li, Fan Guo, Bin Chen, and Chengcai Shen, Lindsay Glesener, *The Astrophysical Journal* Jun 2022
- Large-scale Compression Acceleration during Magnetic Reconnection in a Low- Plasma, Xiaocan Li, Fan Guo, Hui Li, and Shengtai Li, *The Astrophysical Journal* Oct 2018

6.2 Funding and Acknowledgements

The development of the GPAT model is supported by NASA through Grant 80NSSC21K1313, NSF Grant No. AST-2107745, Smithsonian Astrophysical Observatory (SAO) through subcontract SV1-21012 (primary: NASA 80NSSC21K2044), and Los Alamos National Laboratory through subcontract No. 622828 (primary: NASA 80NSSC20K1318).

We thank the open-source projects, including MT_STREAM PRNG, FoBiS.py, FLAP, and plasmapy to make the code possible. We acknowledge all the contributors and users of the GPAT community to provide feedback and help improve the code.

BIBLIOGRAPHY

[Bieber96] Bieber, J.W., Wanner, W. and Matthaeus, W.H., 1996. Dominant two-dimensional solar wind turbulence with implications for cosmic ray transport. Journal of Geophysical Research: Space Physics, 101(A2), pp.2511-2522.

[Dwyer97] Dwyer, J.R., Mason, G.M., Mazur, J.E., Jokipii, J.R., Von Rosenvinge, T.T. and Lepping, R.P., 1997. Perpendicular transport of low-energy corotating interaction region-associated nuclei. The Astrophysical Journal, 490(1), p.L115.

[Florinski03] Florinski, V., Zank, G.P. and Pogorelov, N.V., 2003. Galactic cosmic ray transport in the global heliosphere. Journal of Geophysical Research: Space Physics, 108(A6).

[Florinski09] Florinski, V. and Pogorelov, N.V., 2009. Four-dimensional transport of galactic cosmic rays in the outer heliosphere and heliosheath. The Astrophysical Journal, 701(1), p.642.

[Giacalone99] Giacalone, J. and Jokipii, J.R., 1999. The transport of cosmic rays across a turbulent magnetic field. The Astrophysical Journal, 520(1), p.204.

[Jokipii71] Jokipii, J.R., 1971. Propagation of cosmic rays in the solar wind. Reviews of Geophysics, 9(1), pp.27-87.

[Kong17] Kong, X., Guo, F., Giacalone, J., Li, H. and Chen, Y., 2017. The acceleration of high-energy protons at coronal shocks: the effect of large-scale streamer-like magnetic field structures. The Astrophysical Journal, 851(1), p.38.

[Pei10] Pei, C., Bieber, J. W., Burger, R. A., & Clem, J. 2010, Journal of Geophysical Research (Space Physics), 115, A12107

[Zhang99] Zhang, M., 1999. A Markov stochastic process theory of cosmic-ray modulation. The Astrophysical Journal, 513(1), p.409.

[Zhang03] Zhang, M., Jokipii, J.R. and McKibben, R.B., 2003. Perpendicular transport of solar energetic particles in heliospheric magnetic fields. The Astrophysical Journal, 595(1), p.493.

[Burrage04] Burrage, K., Burrage, P.M. and Tian, T., 2004. Numerical methods for strong solutions of stochastic differential equations: an overview. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 460(2041), pp.373-402.

[Li18] Large-scale Compression Acceleration during Magnetic Reconnection in a Low- Plasma, Xiaocan Li, Fan Guo, Hui Li, and Shengtai Li, The Astrophysical Journal 866, no. 1 (2018): 4.

[Strauss17] Strauss, R. and Effenberger, F., 2017. A hitch-hiker's guide to stochastic differential equations. Space Science Reviews, 212(1), pp.151-192.

[Earl88] Earl, J.A., Jokipii, J.R. and Morfill, G., 1988. Cosmic-ray viscosity. The Astrophysical Journal, 331, pp.L91-L94.

[LeRoux07] Le Roux, J.A. and Webb, G.M., 2007. Nonlinear cosmic-ray diffusive transport in combined two-dimensional and slab magnetohydrodynamic turbulence: a BGK-Boltzmann approach. The Astrophysical Journal, 667(2), p.930.

[Schlickeiser89] Schlickeiser, R., 1989. Cosmic-ray transport and acceleration. I-Derivation of the kinetic equation and application to cosmic rays in static cold media. II-Cosmic rays in moving cold media with application to diffusive shock wave acceleration. The Astrophysical Journal, 336, pp.243-293.

[Schlickeiser98] Schlickeiser, R. and Miller, J.A., 1998. Quasi-linear theory of cosmic ray transport and acceleration: the role of oblique magnetohydrodynamic waves and transit-time damping. The Astrophysical Journal, 492(1), p.352.

[Skilling75] Skilling, J., 1975. Cosmic Ray Streaming—II effect of particles on alfvén waves. Monthly Notices of the Royal Astronomical Society, 173(2), pp.245-254.

[Webb18] Webb, G. M., Barghouty, A. F., Hu, Q., & le Roux, J. A. 2018, The Astrophysical Journal, 855, 31

[Williams91] Williams, L.L. and Jokipii, J.R., 1991. Viscosity and inertia in cosmic-ray transport-Effects of an average magnetic field. The Astrophysical Journal, 371, pp.639-647.

[Williams93] Williams, L.L., Schwadron, N., Jokipii, J.R. and Gombosi, T.I., 1993. A unified transport equation for both cosmic rays and thermal particles. The Astrophysical Journal, 405, pp.L79-L81.

[Jokipii77] Jokipii, J.R. and Levy, E.H., 1977. Effects of particle drifts on the solar modulation of galactic cosmic rays. The Astrophysical Journal, 213, pp.L85-L88.

[Kong22] Kong, X., Chen, B., Guo, F., Shen, C., Li, X., Ye, J., Zhao, L., Jiang, Z., Yu, S., Chen, Y. and Giacalone, J., 2022. Numerical modeling of energetic electron acceleration, transport, and emission in solar flares: connecting loop-top and footpoint hard X-ray sources. The Astrophysical Journal Letters, 941(2), p.L22.

[Zank14] Zank, G.P., 2014. Transport processes in space physics and astrophysics (Vol. 877, p. 185). Berlin: Springer.

[Zhang09] Zhang, M., Qin, G. and Rassoul, H., 2009. Propagation of solar energetic particles in three-dimensional interplanetary magnetic fields. The Astrophysical Journal, 692(1), p.109.

[Zhang17] Zhang, M. and Zhao, L., 2017. Precipitation and release of solar energetic particles from the solar coronal magnetic field. The Astrophysical Journal, 846(2), p.107.

[Zuo13] Zuo, P., Zhang, M. and Rassoul, H.K., 2013. The role of cross-shock potential on pickup ion shock acceleration in the framework of focused transport theory. The Astrophysical Journal, 776(2), p.93.