LOKI TUTORIAL

BILL ARRIGHI

Introduction

Loki is a parallel code for the solution of the coupled Vlasov-Poisson or Vlasov-Maxwell equations. It consists of 2 separate programs. The first is the actual parallel simulation code and is named vlasovPoisson4D. The second is a serial post-processing tool named vp4DPostProcess. Despite the name, the simulation code is able to solve both the Vlasov-Poisson and the Vlasov-Maxwell systems. The system to be solved is specified in the user's input deck the details of which will be discussed below.

The parallel simulation code produces a number of output files containing the fields and time histories relevent to the simulation. The serial post-processor may be run to serialize the field and time history data from these output files. Several large files containing the 4D distribution functions of the kinetic species in the problem are produced. In addition, smaller files containing the 2D fields and the 0D time histories generated by the simulation are generated. The details of the contents of these files are discussed below in the section about Output Files.

There is also a serial utility named testDist which will read an input deck and number of Vlasov processors and compute and print the smallest partition in each dimension. If the partition is less than the required stencil width, a warning for that dimension will also be printed. This allows a user to avoid the pain of submitting a large job and waiting for it to eventually run only to discover that the problem has been overdecomposed and can not run.

Of interest to developers is the testing infrastructure. It consists of a set of tests under the tests directory. The tests are executed and differenced against a set of baselines via the "make check" command. The results of running this command may be deleted with the "make checkclean" command. The differencing is performed by a program named checkTests which is controlled by a set of files specific to each test. There is more information on the test infrastructure later in this document.

Building Loki

Loki requires several external packages including Metis, ParMetis, SuperLU, SuperLU_dist, PETSc, FFTW, and HDF5. We will describe how to build and organize these packages below. Although Loki needs several separate packages the instructions given

Date: Octber 17, 2022.
LLNL-SM-737619.

below to build these packages will result in a directory structure such that a single, top-level path is all that Loki's configure script needs in order to find all these packages.

External Packages. These instructions have been written explicitly for LLNL's LC TOSS3 computers. It is likely that they will work for an LC CHAOS (TOSS2) system as well. The instructions worked largely unchanged to build the packages on Jeff Bank's computer at RPI.

Note that under TOSS3 there are serious issues mixing compilers. Specifically, it was not possible to get a good build if GNU C++ and C compilers were used in conjunction with the PGI Fortran compiler. Thus, these instructions are written to build all the external packages with the GNU compiler suite. It is likely that these restrictions may be relaxed when building on a CHOAS (TOSS2) machine.

All the external packages have a traditional notion of separate build and install locations. Thus, you may untar and build wherever you wish. The installation process places the necessary components into the requested installation location.

Here are the instructions to build each external package. They should be built in the order in which they appear in these instructions. All packages will be installed in subdirectories of $INST_DIR. On LC's TOSS3 systems INST_DIR was /usr/gapps/valhalla/LOKI/PACKAGES_TOSS3.

Before doing anything:
ml gcc/4.9.3

Build METIS:
You need metis_4.0.3 which is part of parmetis_4.0.3.

(1) cd to the directory in which you will build METIS
(2) tar xvzf parmetis-4.0.3.tar.gz
(3) cd parmetis-4.0.3/metis
(4) setenv METIS_INST $INST_DIR/METIS_4.0.3
(5) make config cc="path to C compiler" prefix="$METIS_INST"
(6) make
(7) make install

Build PARMETIS:
These instructions assume that you just built METIS and follow directly from the final METIS build step.

(1) cd ..
    This should put you in the top level directory of the PARMETIS distribution.
(2) setenv PARMETIS_INST $INST_DIR/PARMETIS_4.0.3
(3) make config cc="path to C compiler MPI wrapper" cxx="path to C++ compiler MPI wrapper" prefix="$PARMETIS_INST"
(4) make
(5) make install
(6) cd ..
(7) rm -rf parmetis-4.0.3

Build SUPERLU:
You need superlu_4.3. Note that the README file for this package is quite misleading. The default and install make targets do not work.

(1) cd to the directory in which you will build SUPERLU
(2) tar xvzf superlu_4.3.tar.gz
(3) cd SuperLU_4.3
(4) cp MAKE_INC/make.linux make.inc
(5) Edit make.inc:
    (a) Edit SuperLUroot to point to where you've untarred everything.
    (b) You may need to edit BLASLIB to point to where libblas lives. We needed to make it /usr/lib64
    (c) Edit CC to point to the C compiler.
    (d) Edit FORTRAN to point to the fortran compiler.
    (e) You may need to edit CDEFS, we didn't need to.
    (f) Ignore MATLAB.
(6) make superlulib
(7) As stated above, the install target does not work so you need to do the following:
    (a) setenv SUPERLU_INST $INST_DIR/SUPERLU_4.3
    (b) Ensure that $SUPERLU_INST contains a lib and an include directory.
    (c) cp SRC/*.h $SUPERLU_INST/include
    (d) cp lib/libsuperlu_4.3.a $SUPERLU_INST/lib
    (e) cd $SUPERLU_INST/lib
    (f) ln -s libsuperlu_4.3.a libsuperlu.a
(8) You may now rm -rf SuperLU_4.3 from wherever you built SUPERLU.

Build SUPERLU_DIST:
You need superlu_dist_3.3-p1. The build process is similar to superlu above and the same caveats about the README and default/install make targets apply.

(1) cd to the directory in which you will build SUPERLU_DIST
(2) tar xvzf superlu_dist_3.3-p1.tar.gz
(3) cd SuperLU_DIST-3.3-p1
(4) cp MAKE_INC/make.i386_linux make.inc
(5) Edit make.inc:
    (a) Edit DSuperLUroot to point to where you've untarred everything.
    (b) Get rid of the /lib in DSUPERLULIB.
    (c) You may need to edit BLASLIB to point to where libblas lives. We needed to make it /usr/lib64
    (d) Modify METISLIB and PARMETISLIB to indicate where the metis and parmetis libs are. If you installed metis in $METIS_INST and parmetis in $PARMETIS_INST you'll need:
    METISLIB = -L$METIS_INST/lib -lmetis
    PARMETISLIB = -L$PARMETIS_INST/lib -lparmetis
    (e) Edit CC to point to the C compiler MPI wrapper.

   (f) Edit FORTRAN to point to the fortran compiler MPI wrapper.
   (g) You may need to edit CDEFS, we didn't need to.
(6) We needed to edit SRC/xerbla.c to add
    #include <stdio.h >
(7) make superlulib
(8) Again, the install target does not work so you need to do the following:
   (a) setenv SUPERLU_DIST_INST $INST_DIR/SUPERLU_DIST_3.3-p1
   (b) Ensure that $SUPERLU_DIST_INST contains a lib and and include directory.
   (c) cp SRC/*.h $SUPERLU_DIST_INST/include
   (d) cp libsuperlu_dist_3.3.a $SUPERLU_DIST_INST/lib
   (e) cd $SUPERLU_DIST_INST/lib
   (f) ln -s libsuperlu_dist_3.3.a libsuperlu_dist.a
   (g) As SuperLU_Dist depends on Metis and ParMetis it is useful to define the versions used by doing the following in $SUPERLU_DIST_INST:
       ln -s ../METIS_4.0.3 METIS
       ln -s ../PARMETIS_4.0.3 PARMETIS
(9) You may now rm -rf SupreLU_DIST-3.3-p1 from wherever you built SUPERLU_DIST.

Build PETSc:
You need PETSc 3.4.5. When configuring PETSc we didn't need and in fact could not use --with-cc, --with-cxx, or --with-fc due to PETSc's configure script's assumptions and where the TOSS3 machines put the compilers/mpi. This may not hold for another architecture.

(1) cd to the directory in which you will build PETSc
(2) tar xvzf petsc-3.4.5.tar.gz
(3) cd petsc-3.4.5
(4) setenv PETSC_INST $INST_DIR/PETSc_3.4.5
(5) ./configure --configModules=PETSc.Configure
    --optionsModule=PETSc.compilerOptions --PETSC_ARCH=linux-gnu-opt
    --with-mpi-dir="/usr/tce/packages/mvapich2/mvapich2-2.3.6-gcc-4.9.3"
    --with-debugging=0 --with-matlab=0
    --with-superlu-include="$SUPERLU_INST/include"
    --with-superlu-lib="$SUPERLU_INST/lib/libsuperlu_4.3.a"
    --with-superlu_dist-include="$SUPERLU_DIST_INST/include"
    --with-superlu_dist-lib="$SUPERLU_DIST_INST/lib/libsuperlu_dist_3.3.a"
    --prefix="$PETSC_INST" --with-shared-libraries=0
    --with-parmetis-include="$PARMETIS_INST/include"
    --with-parmetis-lib="$PARMETIS_INST/lib/libparmetis.a"
    --with-metis-include="$METIS_INST/include"
    --with-metis-lib="$METIS_INST/lib/libmetis.a"
(6) Run the make command that configure tells you to run.
(7) Run the make install command that make tells you to run.

(8) cd ..
rm -rf petsc-3.4.5
(9) Again, as PETSc depends on SuperLU and SuperLU_Dist it is useful to define the versions used by doing the following in $PETSC_INST
ln -s ../SUPERLU_4.3 SUPERLU
ln -s ../SUPERLU_DIST_3.3-p1 SUPERLU_DIST
(10) Set up the following soft link in $INST_DIR
ln -s PETSc_3.4.5 PETSc

Build FFTW:
FFTW is available on all LC platforms hence we did not need to build it ourselves. We used version 3.3.8 built in parallel. Assuming that the FFTW installation is located in $FFTW_DIR set up the following soft link in $INST_DIR
ln -s $FFTW_DIR FFTW

Build HDF5:
HDF5 is so ubiquitous that it is often already built on many machines so explicit instructions for building this package is not included here. A parallel HDF5 build is needed by Loki. We used version 1.8.17. Assuming that the HDF5 installation is located in $HDF5_DIR set up the following soft link in $INST_DIR
ln -s $HDF5_DIR HDF

Now all the packages needed by Loki have been built and you may build Loki itself.

Loki. Now that the external packages have been built, Loki itself may be built. Loki may be built with either the GNU or Intel compiler suites. As was noted above with the external packages, mixing in a different Fortran compiler when building on a TOSS3 machine will cause problems.

Loki's build system is based on the GNU configure and make utilities. The general process is to run configure passing it information about the compilers to use, optional compiler flags, the location of the external packages, and other build options such as whether to build an optimized or debugable version of the code. Note that although there are many separate external packages needed by Loki, the instructions for building these pacakges created a directory structure such that it is only necessary to pass one, top-level path to Loki's configure script. The configure script can find all the individual packages from this one path if the the instructions for building the external packages have been followed.

You can get a complete list of the options that configure may be given by running ./configure --help from the command line. Generally, you must give configure a C++, C, and Fortran compiler. There are several scripts whose names all begin with "doinstall" in the source code distribution that describe the configure/make process for different compiler/machine combinations at LLNL.

Once the code has been configured, simply type make from the command line to compile and link the code. Dependencies have been generated and stored in Makefile.depend. Therefore if modifications are made to the code it should only be necessary to run make again. If more extensive modifications are made such as adding new header or source files

the dependencies should be regenerated. This is done by running the script "scripts/depend" from the top-level directory of the source code distribution.

Running Loki

The details of how to run vlasovPoisson4D depend on the execute environment on which the code is being run. At LLNL, the general process is to construct an msub script which contains an srun command to launch vlasovPoisson4D. The vlasovPoisson4D program itself takes only 1 argument, the name of the input deck. It produces a number of files which may be post-processed. These files all have a common root name which is specified in the input deck.

Once vlasovPoisson4D has been run, the serial post-processing tool vp4DPostProcess may be run. As this is a serial tool it may be run directly from the command line. There are only 2 arguments that this tool takes. The first is required and is the root name of the output files. The second is an option to not process the distribution functions. As the post-processor is serial, it may not be possible to process the distribution functions for large problems due to the volume of the data. The root name of the output is given with -prefix=root and the option to not process the distributions is given with -skip_dists. There is also the ability to allow processing specific distribution checkpoints with the -process_dists command. This command takes a comma separated list of integers corresponding to the indexes of the checkpoints to process. Thus to post-process a run that generated output with the root name planeEPW, one could execute one of the following 3 commands:
vlasovPoisson4D -prefix=planeEPW
or
vlasovPoisson4D -prefix=planeEPW -skip_dists or
vlasovPoisson4D -prefix=planeEPW -process_dists=2,5

Input Decks

Input Deck File Format. Loki is invoked with 1 argument which is the name of the text file containing the input parameters for the problem. Input parameters are specified in a hierarchical rather than a flat format. Input parameters are organized into hierarchical namespaces. A "." indicates the start of an embedded namespace. For example, the parameters relevant to the Poisson solve are specified in the namespace beginning with "poisson.". This will become clearer with specific examples. There is no required ordering of input parameters in the file although grouping them in their hierarchical order makes reading the file much easier. One note is that there are cases where the "." character is part of a namespace name. Reserving this character for delimiting namespaces would have been preferable. However, the input format is simple enough that one can fairly easily tell the use of a "." by context.

Each Loki input parameter must be one of the following 4 data types.

(1) booleans: specified as either true or false (these are case sensitive)

(2) reals: specified as decimal numbers or in scientific notation
examples are 100, 100.0, 1.0e2
(3) integers: specified as integer numbers
(4) strings: specified as character strings delimited by the " character
examples are "electron" and "Perturbed Maxwellian"

Arrays of these data types are also allowed. An array is specified as multiple entries on the same line separated by a space such as 0.1 0.2 1.0.

Comments are allowed in the file. Any line beginning with the "#" character is considered to be a comment. Each input parameter is specified with a line of the form:
parameter = value

It is frequently useful to define constants and build up numeric expressions from these constants for use in an input file. This can be done by embedding Perl syntax in the file. As these Perl expressions must be identified as such so that they may be passed to the Perl interpreter they all must end with a ";". Perl variables may then be referenced in the definition of Loki input parameters. Here is how something like this might look:

```
# Perl expressions to define useful quantities
$pi = 3.141592653589793238462 6;
$third = 1/3;
$twelfth = 1/12;
$xa = -3*$pi;
$xb = 3*$pi;
$xaK = -2.4375*$pi;
$xbK = 2.4375*$pi;
$ya = -78*$pi;
$yb = 78*$pi;
$vmax = 7;
$vmin = -7;
$Lx = 6*$pi;
$Ly = 288*$pi;
...
# Loki input parameters
$domain_limits = $xa $xb $ya $yb
...
```

It is not necessary to enforce any ordering on the input parameters. Embedded Perl code, however, will need to appear in the desired order of execution.

Loki has been normalized to electron thermal units. Therfore, all input parameters should be in these units.

We will now discuss the individual input parameters that control Loki. For each we will give their meaning, data type, any limits, any default values, and notes concerning restrictions or relationships with other parameters. The parameters will be discussed in their hierarchical ordering.

General Simulation Parameters. These parameters are in the top level namespace and describe general problem control and scope.

spatial_solution_order

Description: Specifies the order of spatial accuracy of the solution.
Data Type: Integer
Default: 4
Notes: Must be either 4 or 6.

temporal_solution_order

Description: Specifies the order of temporal accuracy of the solution.
Data Type: Integer
Default: 4
Notes: Must be either 4 or 6.

do_relativity

Description: Flag to indicate if realtivistic versions of the algorithms should be used.
Data Type: Boolean
Default: false

light_speed

Description: Light speed.
Data Type: Real
Default: None. This input is required if a Vlasov-Maxwell system is specified or if do_realativity is true.

verbosity

Description: If >= 1 Δt is printed for each time step.
Data Type: Integer
Default: 1

cfl

Description: Sets the time step safety factor.
Data Type: Real
Default: 0.9

final_time

Description: End simulation if this simulation time is reached.
Data Type: Real
Default: 1.0

max_step

Description: End simulation if this time step is reached.
Data Type: Integer
Default: 0
Notes: As the default is 0 you had better enter something > 0.

sequence_write_times

Description: Save 0D time histories such as energies, fluxes, and probe information at multiples of this simulation time. This data is written to files whose naming is of the form restart.write_directory.time_hists_n.hdf.
Data Type: Real
Default: 1.0

coll_sequence_write_times

Description: Save 0D collision diagnostic time histories at multiples of this simulation time. This data is written to files whose naming is of the form restart.write_directory_coll.time_hists_n.hdf.
Data Type: Real
Default: None. Either this or coll_sequence_write_steps must be specified, but not both.

coll_sequence_write_steps

Description: Save 0D collision diagnostic time histories at this multiple of simulation steps. This data is written to files whose naming is of the form restart.write_directory_coll.time_hists_n.hdf.
Data Type: Integer
Default: None. Either this or coll_sequence_write_times must be specified, but not both.

save_times

Description: Save 2D configuration space arrays at multiples of this simulation time. This data is written to files whose naming is of the form restart.write_directory.show.
Data Type: Real
Default: 1.0
Notes: This time interval is strictly enforced. Loki will adjust the simulation time step so that a time step always ends at a multiple of this time. This also ensures that time histories are written out at a consistent time interval.

coll_save_times

Description: Save 2D configuration space arrays of collision diagnostics at multiples of this simulation time. This data is written to files whose naming is of the form restart.write_directory_coll.show.
Data Type: Real
Default: None. Either this or coll_save_steps must be specified, but not both.

coll_save_steps

Description: Save 2D configuration space arrays of collision diagnostics at this multiple of simulation steps. This data is written to files whose naming is of the form restart.write_directory_coll.show.
Data Type: Integer
Default: None. Either this or coll_save_times must be specified, but not both.

plot_ke_fluxes

Description: If true, save the kinetic energy fluxes of each species through the 4 velocity space boundaries. These are 2D configuration space arrays. They may be useful for debugging purposes so the default for this parameter is false and it must be explicitly enabled in the input deck.
Data Type: Boolean
Default: false

save_data

Description: If true time histories will be sampled and plot files written. If false you will not get any time histories or plots no matter what was specified for sequence_write_times and save_times.
Data Type: Boolean
Default: true

save_coll_data

Description: This parameter is the collisional analog to save_data above for the Rosenbluth collision operator. This operator may be enabled to generate diagnostic output. If this input is true then this diagnostic output will be written. If it is false, you will ot get any collisional diagnostics no matter what was specified for coll_seq_write_times, coll_srq_write_steps, coll_save_times, and coll_save_steps.
Data Type: Boolean
Default: false

max_files_for_write

Description: This parameter controls the number of files written at each save time for the 2D configuration space arrays.
Data Type: Integer
Default: 16

plot_times_per_file

Description: This parameter controls the number of plot times written to each show file. The default of 1 will generate a large number of show files (max_files_for_write * number of plot times). This does not play well with a Lustre file system and will cause postprocessing to be unreasonably slow. Increasing this value results in a correspondingly smaller number of show files. Note that making this value too large carries some risk. First, the show file reader will fail if ALL the plot times are written to a single show file preventing postprocessing. Second, if the code terminates abnormally the show file currently being written to may not be closed properly and that file's data may be lost.
Data Type: Integer
Default: 1

start_from_restart

Description: If true problem will restart from last saved restart. Otherwise problem will start from initial conditions.
Data Type: Boolean

Default: false
Notes: See RestartManager parameters for more on how to run from a saved restart.

sys_type

Description: Specify a Vlasov-Poisson (electrostatic) or Vlasov-Maxwell (electrodynamic) system.
Data Type: String
Default: "poisson"
Notes: Must be "poisson" or "maxwell".

number_of_probes

Description: The number of probes to sample.
Data Type: Integer
Default: 1
Notes: Must be >= 1.

probe.n.location

Description: An array of 2 values specifying the location of each probe as a fraction of the configuration space extent in each direction. Probes are numbered starting from 1. Therefore the command to place the first probe at a location .25 the extent of the domain in x and .5 of the extent of the domain in y would look like:
probe.1.location = 0.25 0.5
Data Type: Real
Default: 0.5 0.5

domain_limits

Description: Upper and lower limits of each dimension of configuration space specified as an array of the form:
domain_limits = x_lo x_hi y_lo y_hi
Data Type: Real
Default: 0.0 1.0 0.0 1.0
Notes: Although there is a default it is almost certainly not what you want to use.

periodic_dir

Description: For each dimension of configuration space, if true apply periodic boundary conditions in that dimension. Otherwise do not.
Data Type: Boolean
Default: false false

N

Description: The number of cells in each dimension of configuration space specified as an array of the form:
N = N_x N_y
Data Type: Integer
Default: None. This input is required

Notes: N in each dimension must be at least the stencil width. If the spatial solution order is 4, then N >= 5 in each dimension. If the spatial solution order is 6, then N >= 7 in each dimension.

number_of_species

Description: Specifies the number of kinetic species in the problem.
Data Type: Integer
Default: 1

Parameters Describing Kinetic Species. There is a variety of input associated with each kinetic species in the problem. The input parameters of each kinetic species are defined in separate namespaces. The namespace for the nth kinetic species is kinetic_species.n where n is one based. Therefore to specify the name of the 1st kinetic species as "electron" one would use the following syntax:
kinetic_species.1.name = "electron"

name

Description: The name of the kinetic species.
Data Type: String
Default: None. This input is required.

mass

Description: The mass of the kinetic species.
Data Type: Real
Default: None. This input is required.

charge

Description: The charge of the kinetic species.
Data Type: Real
Default: None. This input is required.

num_external_drivers

Description: Specifies the number of external electric field drivers for this species.
Data Type: Integer
Default: 0

num_collision_operators

Description: Specifies the number of collision operators for this species.
Data Type: Integer
Default: 0

velocity_limits

Description: Upper and lower limits of each dimension of velocity space specified as an array of the form:
kinetic_species.n.velocity_limits = vx_lo vx_hi vy_lo vy_hi
Data Type: Real
Default: None. This input is required.

Nv

Description: The number of cells in each dimension of velocity space specified as an array of the form:
kinetic_species.n.velocity_limits = N_{vx} N_{vy}
Data Type: Integer
Default: None. This input is required.

Parameters Describing Kinetic Species Initial Conditions. There are 5 ways that initial conditions may be specicied. The first 4 are through the "Perturbed Maxwellian", "Landau damping", "Maxwellian with noise", and "Interpenetrating Stream" initial conditions. In these cases the user provides a set of input parameters and the code computes the complete initial condition. The fifth way is through an initial condition called "External 2D". In this case the user precomputes and writes to an HDF5 file the desired 2D configuration space distribution. Input parameters describing the velocity space dependency of the initial distribution are provided by the user and the code combines the externally defined 2D distribution with this velocity dependency to form the complete initial condition.

Each of these initial conditions is factored into configuration space behavior and thermal behavior. The configuration space behavior is unique to each initial condition. The thermal behavior is selected by the user and is either a Maxwellian or a Juttner distribution in velocity space. The Juttner distribution is only valid for relativistic problems which are imbedded in momentum space. The Maxwellian is valid for either relativistic or non-relativistic problems. Both of these thermal behaviors allow for spatial perturbation so they are functions of all coordinates.

The functional form of the Maxwellian thermal behavior in terms of user input is as follows. The x coordinate is x1, y coordinate is x2, vx (or px for relativistic problems) coordinate is x3, and xy (or py for relativistic problems) coordinate is x4.

Maxwellian thermal behavior
$vx = vx0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + vflowinitx$
$vy = vy0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + vflowinity$
$fthermal(x1, x2, x3, x4) = mass/(2.0 * \pi * sqrt(tx * ty)) *$
$exp(-0.5 * ((x3 - vx) * *2/(tx/mass) + (x4 - vy) * *2/(ty/mass)))$

The functional form of the Juttner thermal behavior in terms of user input is as follows. The x coordinate is x1, the y coordinate is x2, the px coordinate is x3, the py coordinate is x4, and x is the speed of light is c.

Juttner thermal behavior
$px = px0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + pflowinitx$
$py = py0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + pflowinity$
$fthermal(x1, x2, x3, x4) = (c * *2)/(2.0 * \pi * (1.0 + (c * *2)) *$
$exp((c * *2) - (c * *2) * sqrt(1.0 + ((x3 - px) * *2 + (x4 - py) * *2)/(c * *2)))$

For any initial condition there is a variety of input that the user must specify. The input parameters of each kinetic species' initial condition are defined in a sparate namespace of that kinetic species. The namespace for the initial condition of the nth kinetic species is kinetic_species.n.ic where n is one based. Therefore to specify the name of the initial condition of the 1st kinetic species as "Perturbed Maxwellian" one would use the following

syntax:
kinetic_species.1.ic.name = "Perturbed Maxwellian"

Perturbed Maxwellian, Landau damping, and Maxwellian with noise. These 3 initial conditions are closely related. As much of their input is common they will be discussed together here.

The functional forms of these initial conditions in terms of their input parameters are below. In each case x1 is the x coordinate, x2 is the y coordinate, x3 is the vx coordinate, x4 is the vy coordinate, fthermal(x1,x2,x3,x4) is either the Maxwellian or Juttner thermal behavior described above, and f is the distribution function.

Perturbed Maxwellian
$f(x1, x2, x3, x4) = frac * fthermal(x1, x2, x3, x4) * (1.0 + A * cos(kx1 * x1 + spatial\_phase) * cos(ky1 * x2 + spatial\_phase) + B * cos(kx2 * x1 + spatial\_phase) + C * cos(ky2 * x2 + spatial\_phase))$

Landau damping
$f(x1, x2, x3, x4) = frac * fthermal(x1, x2, x3, x4) * (1.0 + A * cos(kx1 * x1 + ky1 * x2 + spatial\_phase))$

Maxwellian with noise
Lx = length of configuration space in x dimension
noise = 1.0
do k = 1,number_of_noisy_modes
$noise = noise + noise\_amplitudes(k) * cos(2.0 * \pi * k * (x1 + noise\_phases(k))/Lx + spatial\_phase)$
end do
$f(x1, x2, x3, x4) = noise * frac * fthermal(x1, x2, x3, x4)$

These parameters apply to all 3 of these initial conditions:

name

Description: The name of the initial condition.
Data Type: String
Default: None. This input is required.
Notes: Must be one of "Perturbed Maxwellian", "Landau damping", or "Maxwellian with noise".

maxwellian_thermal

Description: Enables Maxwellian or Juttner thermal behavior.
Data Type: Boolean
Default: true

tx

Description: Temperature in x.
Data Type: Real
Default: 1.0
Notes: This is related to the old input alpha by alpha=sqrt(mass/tx).

ty

Description: Temperature in y.
Data Type: Real
Default: 1.0
Notes: This is related to the old input beta by beta=sqrt(mass/ty).

vx0

Description: Amplitude of spatial perturbation of Maxwellian in vx.
Data Type: Real
Default: 0.0
Notes: Although it is possible to use this to specify a constant x flow velocity by setting x_wave_number, y_wave_number, and phase to 0 it is strongly recommended to use the vflowinitx input which has this intended purpose. vx0 is intended to specify any spatially varying initial x flow velocity.

vy0

Description: Amplitude of spatial perturbation of Maxwellian in vy.
Data Type: Real
Default: 0.0
Notes: Although it is possible to use this to specify a constant y flow velocity by setting x_wave_number, y_wave_number, and phase to 0 it is strongly recommended to use the vflowinity input which has this intended purpose. vy0 is intended to specify any spatially varying initial y flow velocity.

vflowinitx

Description: Inital constant x flow velocity.
Data Type: Real
Default: 0.0

vflowinity

Description: Inital constant y flow velocity.
Data Type: Real
Default: 0.0

frac

Description: Relative weight of this kinetic species.
Data Type: Real
Default: 1.0

x_wave_number

Description: Wave number in x of spatial perturbation of Maxwellian in vx and vy.
Data Type: Real
Default: 0.0

y_wave_number

Description: Wave number in y of spatial perturbation of Maxwellian in vx and vy.
Data Type: Real
Default: 0.0

phase

Description: Phase of spatial perturbation of Maxwellian in vx and vy.
Data Type: Real
Default: 0.0

spatial_phase

Description: Phase of spatial perturbation of Maxwellian in x and y.
Data Type: Real
Default: 0.0

These parameters apply to only "Perturbed Maxwellian" and "Landau damping":

A

Description: One amplitude of spatial perturbation of Maxwellian in x and y. See functional forms above for use in each initial condition type.
Data Type: Real
Default: 0.0

kx1

Description: Wave number in x of spatial perturbation of Maxwellian in x having amplitude A. See function forms above for use in each initial condition type.
Data Type: Real
Default: 0.5

ky1

Description: Wave number in y of spatial perturbation of Maxwellian in y having amplitude A. See function forms above for use in each initial condition type.
Data Type: Real
Default: 0.5

These parameters apply to only "Perturbed Maxwellian":

B

Description: Amplitude of spatial perturbation of Maxwellian in x controlled by kx2. See functional form above for use.
Data Type: Real
Default: 0.0

kx2

Description: Wave number in x of spatial perturbation of Maxwellian in x having amplitude B. See function form above for use.
Data Type: Real
Default: 0.5

C

Description: Amplitude of spatial perturbation of Maxwellian in y controlled by ky2. See functional form above for use.
Data Type: Real
Default: 0.0

ky2

Description: Wave number in y of spatial perturbation of Maxwellian in y having amplitude C. See function form above for use.
Data Type: Real
Default: 0.5

These parameters apply to only "Maxwellian with noise":
number_of_noisy_modes

Description: Number of noise terms.
Data Type: Integer
Default: 0

noise_amplitudes

Description: Amplitudes of each noise term specified as an array.
Data Type: Real
Default: None. If number_of_noisy_modes > 0 this input is required.
Notes: There must be number_of_noisy_modes values in the array.

noise_phases

Description: Phases of each noise term specified as an array.
Data Type: Real
Default: None. If number_of_noisy_modes > 0 this input is required.
Notes: There must be number_of_noisy_modes values in the array.

Interpenetrating Stream. This initial condition defines box or half plane regions in physical space along with the user option of Maxwellian or Juttner thermal behavior. It is useful for setting up problems involving interpenetrating plasma streams as well as structures such as the interior of NIF hohlraums. The half planes may be be arbitarily oriented in configuration space. The boxes are all rectangles in configuation space. Because the natural way to describes these two geometries is different, there are two distinct syntaxes for defining either a half plane boxes. However the means by which this initial condition creates either geometry is similar. The half plane syntax relies on the location of the region's edge and a rotation of that edge. Hence the input syntax for this geometry describes quantites that are transverse and longitudinal to the half plane's edge. The box syntax describes boxes oriented along the x-y axes of configuration space. Hence this input syntax describes quantities relative to the x-y axes of configuration space.

There is a regression test for this initial condition which may be helpful in understanding its use.

name

Description: The name of the initial condition.
Data Type: String
Default: None. This input is required.
Notes: Must be "Interpenetrating Stream".

maxwellian_thermal

Description: Enables Maxwellian or Juttner thermal behavior.
Data Type: Boolean
Default: true

syntax

Description: The input syntax that is being used.
Data Type: String
Default: "half plane".
Notes: Must be either "half plane" or "box".

vflowinitx

Description: Inital constant x flow velocity.
Data Type: Real
Default: 0.0
Notes: For the "half plane" syntax, vflowinitx and vflowinity are converted into flow velocities transverse and longitudinal to the slab based on the orientation of the slab, theta.

vflowinity

Description: Inital constant y flow velocity.
Data Type: Real
Default: 0.0
Notes: For the "half plane" syntax, vflowinitx and vflowinity are converted into flow velocities transverse and longitudinal to the slab based on the orientation of the slab, theta.

These input parameters are used to describe half planes:
tl

Description: The longitudinal plasma temperature.
Data Type: Real
Default: None. This input is required.

tt

Description: The transverse plasma temperature.
Data Type: Real
Default: None. This input is required.

theta

Description: The orientation of the normal to the slab front in radians. A value of 0 indicates that the normal to the slab front is along x while a value of pi/2 indicates that the normal to the slab is along y.
Data Type: Real
Default: None. This input is required.

d

Description: The location of the slab front. This is the distance of the origin in physical space from the slab front. So with a theta of 0 a positive value for d place the slab front at -d in x.
Data Type: Real
Default: None. This input is required.

beta

Description: Slab front steepness parameter and indicator of which side of the front is populated. To form the slab the distribution function transitions from 1 to 0 at the front. The magnitude of this parameter describes the steepness of that transition. A large beta impliies a steeper slab front. The sign of this parameter indicates which side of the front is populated (on which side the distribution function is 1 vs 0). A positive value for beta populates the left side of the front and a negative value populates the right side of the front. A value of 0 for beta does not make sense.
Data Type: Real
Default: None. This input is required.

floor

Description: Relative minimum weight of the species. The slab is shaped by a Heaviside function which has a value of frac (see next parameter) in the slab region and floor outside the slab region. In some cases the default of 0 for the floor may not be what is needed. One specific use case is for the electrons in a problem in which there are different ions in adjoining slabs. In this case the Heaviside for the electrons is really shaping 2 adjoining slabs of different densities. One slab has density frac and the other has density floor.
Data Type: Real
Default: 0.0

frac

Description: Relative weight of the species.
Data Type: Real
Default: None. This input is required.

two_sided

Description: If true, a mirror image slab is also created. This slab front is located at -d with -beta and -vl0 transverse drift velocity. This is useful to specifiy two interpenetrating streams of the same species (electrons for example). Using this option halves the computational load when compared to specifying 2 species, one for each stream (slab). Note that you must specify d, beta, and vl0 that are self-consistent in specifying one of the slabs. It does not matter which slab you specify as the code will construct the other one for you from the input you have provided.
Data Type: Boolean
Default: false

Notes: When true, frac2 below must be specified. Can not be specified in conjunction with centered.

frac2

Description: Relative weight of the species in the mirror image slab when two_sided is true. Two different relative weights are useful for example when setting up a problem with two different ions and their electons in slabs at opposite ends of the physical domain.
Data Type: Real
Default: None. This input is required when two_sided is true.

centered

Description: If true, the slab centered in the physical domain so that there are 2 fronts, one at d and the other at -d.
Data Type: Boolean
Default: false
Notes: Can not be specified in conjunction with two_sided.

These input parameters are used to describe boxes:
beta

Description: Box edge steepness parameter. Each edge of each box has the same steepness. Larger values of beta result in steeper edges. A value of 0 does not make sense. As this parameter is used on each edge the sign is not significant. The code will take the absolute value of any value entered.
Data Type: Real
Default: The domain's upper x edge.
Notes: This is optional depeding upon where the box is to be located.

tx

Description: The plasma temperature in x.
Data Type: Real
Default: None. This input is required.

ty

Description: The plasma temperature in y.
Data Type: Real
Default: None. This input is required.

num_boxes

Description: The number of boxes being specified.
Data Type: Integer
Default: None. This input is required.
Notes: The boxes are described by the following 5 parameters. To specifiy any of these 5 parameters for the nth box use box.n.parameter = value.

x_hi

Description: The location of the box's upper x edge.

Data Type: Real
Default: The domain's upper x edge.
Notes: This is optional depeding upon where the box is to be located.

x_lo

Description: The location of the box's lower x edge.
Data Type: Real
Default: The domain's lower x edge.
Notes: This is optional depeding upon where the box is to be located.

y_hi

Description: The location of the box's upper y edge.
Data Type: Real
Default: The domain's upper y edge.
Notes: This is optional depeding upon where the box is to be located.

y_lo

Description: The location of the box's lower y edge.
Data Type: Real
Default: The domain's lower y edge.
Notes: This is optional depeding upon where the box is to be located.

frac

Description: Relative weight of this species in this box.
Data Type: Real
Default: None. This input is required.

External 2D. As described above, the user will precompute the configuation space dependence of the distribution function and write it to an HDF5 file. The code applies the thermal behavior selected by the user. This will be either the Maxwellian or Juttner thermal behavior described above. In the examples directory there is a problem set up to use this initial condition. The 2D distribution is generated with the Matlab script supplied with the problem. There is also a regression test of this initial condition which shows more details of its use.

The functional form of this initial condition is given below. Here x1 is the the x coordinate, x2 is the y coordinate, x3 is the vx (or px for relativistic problems) coordinate, x4 is the vy (or py for relativistic problems) coordinate, fthermal(x1,x2,x3,x4) is either the Maxwellian or Juttner thermal behavior described above, the externally specified 2D distribution is extDist(x1, x2), and f is the full 4D distribution function.

$f(x1, x2, x3, x4) = fthermal(x1, x2, x3, x4) * extDist(x1, x2)$

name

Description: The name of the initial condition.
Data Type: String
Default: None. This input is required.
Notes: Must be "External 2D".

maxwellian_thermal

Description: Enables Maxwellian or Juttner thermal behavior.
Data Type: Boolean
Default: true

file_name

Description: The name of the external 2D initial condition.
Data Type: String
Default: None. This input is required.

tx

Description: Temperature in x.
Data Type: Real
Default: 1.0
Notes: This is related to the old input alpha by alpha=sqrt(mass/tx).

ty

Description: Temperature in y.
Data Type: Real
Default: 1.0
Notes: This is related to the old input beta by beta=sqrt(mass/ty).

vx0

Description: Amplitude of spatial perturbation of Maxwellian in vx.
Data Type: Real
Default: 0.0
Notes: Although it is possible to use this to specify a constant x flow velocity by setting x_wave_number, y_wave_number, and phase to 0 it is strongly recommended to use the vflowinitx input which has this intended purpose. vx0 is intended to specify any spatially varying initial x flow velocity.

vy0

Description: Amplitude of spatial perturbation of Maxwellian in vy.
Data Type: Real
Default: 0.0
Notes: Although it is possible to use this to specify a constant y flow velocity by setting x_wave_number, y_wave_number, and phase to 0 it is strongly recommended to use the vflowinity input which has this intended purpose. vy0 is intended to specify any spatially varying initial y flow velocity.

vflowinitx

Description: Inital constant x flow velocity.
Data Type: Real
Default: 0.0

vflowinity

Description: Inital constant y flow velocity.

Data Type: Real
Default: 0.0

x_wave_number

Description: Wave number in x of spatial perturbation of Maxwellian in vx and vy.
Data Type: Real
Default: 0.0

y_wave_number

Description: Wave number in y of spatial perturbation of Maxwellian in vx and vy.
Data Type: Real
Default: 0.0

phase

Description: Phase of spatial perturbation of Maxwellian in vx and vy.
Data Type: Real
Default: 0.0

Parameters Describing Kinetic Species Electric Field Drivers. Multiple electric field drivers are permitted for each kinetic species. The input parameters for each electric field driver for a given kinetic species are defined in separate namespaces of that kinetic species. The input parameters for the nth electric field driver for the mth kinetic species will be specified in the namespace kinetic_species.m.external_driver.n where m and n are one based. Therefore to specify the name of the 2nd electric field driver of the 1st kinetic species one would use the following syntax:
kinetic_species.1.external_driver.2.name = "Shaped Ramped Cosine Driver"

Unlike the external potential driver, each electric field driver applies to a specific kinetic species. This is reflected in how the electric field drivers are specified. They are defined in a namespace of the kinetic species to which they apply.

At the present time there is only one electric field driver, the Shaped Ramped Cosine Driver. The code is designed with an abstract driver class so that new drivers derived from the abstract driver may be easily added. The only change to existing code needed is to add the new class to the driver factory.

The driver envelope, f(t) below, was originally controlled by 2 parameters, t_ramp and t_off. It has been generalized and is now contolled by 3 parameters, t_rampup, t_hold, and t_rampdown. The new syntax is described below. However the old syntax is still recognized by the code and is mapped to the new syntax.

The functional form of the Shaped Ramped Cosine Driver is given here.
if (t0 <= t < t0+t_rampup)
$f(t) = 0.5 + 0.5 * tanh(4.0 * (2.0 * (t - t0)/t\_rampup - 1.0))$
else if (t0+t_rampup <= t < t0+t_rampup+t_hold)
$f(t) = 0.5 + 0.5 * tanh(4.0)$
else if (t0+t_rampup+t_hold <= t < t0+t_ramp+t_hold+t_rampdown)
$f(t) = 0.5 - 0.5 * tanh(4.0 * (2.0 * (t - t0 - t\_rampup - t\_hold)/t\_rampdown - 1.0))$

else
$f(t) = 0$
For Sinusoidal shaping:
if (abs(x-x0) < lwidth/2)
$g(x) = 1 - x\_shape * (sin(\pi * (x - x0)/lwidth)) * *2$
else
$g(x) = 1 - x\_shape$
Exponential shaping depends on the sign of lwidth.
For non-negative lwidth:
if (x <= x0)
$g(x) = 1.0$
else
$g(x) = 1.0 - x\_shape * (1.0 - exp(-(x - x0)/lwidth))$
For negative lwidth:
if (x <= x0)
$g(x) = 1.0 - x\_shape * (1.0 - exp(-(x0 - x)/lwidth))$
else
$g(x) = 1.0$
if (abs(y) < ywidth/2)
$h(y) = 1 - shape * (sin(\pi * y/ywidth)) * *2$
else
$h(y) = 1 - shape$
$Ex\_ext = E\_0 * f(t) * g(x) * h(y) * cos(\pi * x/xwidth - omega * (t - t0) + phase - 0.5 * alpha * (t - t0 - t\_res) * *2)$
$Ey\_ext = 0.0$

name

Description: Name of driver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "Shaped Ramped Cosine Driver"

t0

Description: Time at which driver turns on.
Data Type: Real
Default: 0.0

t_rampup

Description: Ramp up time. Driver ramps up from t0 to t0+t_rampup.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

t_hold

Description: Hold time. Driver remains constant from t0+t_rampup to t0+t_rampup+t_hold.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

t_rampdown

Description: Ramp down time. Driver ramps down from t0+t_rampup+t_hold to t0+t_rampup+t_hold+t_rampdown.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

E_0

Description: Amplitude of driver.
Data Type: Real
Default: 0.01

xwidth

Description: Half of a wavelength of modulation in x.
Data Type: Real
Default: 0.5
Notes: Again, the default is almost certainly not what you want to use.

omega

Description: Frequency of external driver.
Data Type: Real
Default: 1.0
Notes: Again, the default is almost certainly not what you want to use.

alpha

Description: Frequency like term in external driver. Controls t**2 modulation of driver.
Data Type: Real
Default: 0.0

t_res

Description: Together with t0, determines center of t**2 modulation of driver. Center of t**2 modulation is at t0+t_res.
Data Type: Real
Default: 0.0

x0

Description: Center of shape function in x.
Data Type: Real
Default: 0.0

lwidth

Description: For sinusoidal shaping, the half wavelength of the shape function in x. For exponential shaping, the inverse rate of decay of the shape function. A non-negative lwidth applies the exponential shape to the right of x0 while a negagive value applies the exponential shape to the left of x0.
Data Type: Real
Default: 0.5
Notes: Again, the default is almost certainly not what you want to use.

x_shape

Description: This is the amplitude of the shape function in x but in practice it is more of a flag to shape between -xwidth/2 and xwidth/2 or not. A value of 0 results in no shaping in x. A value of 1 results in modulation between x0-lwidth/2 and x0+lwidth/2 and a constant 0 outside.
Data Type: Real
Default: 0.0

shape_type

Description: This is the type of the shape function in x. The available shape types are the square of a sinusoid or an exponential. The allowed inputs are "sin2" for the sinusoidal and "exp" for the exponential.
Data Type: String
Default: "sin2"

ywidth

Description: Half of a wavelength of shape function in y.
Data Type: Real
Default: 0.5
Notes: Again, the default is almost certainly not what you want to use.

shape

Description: This is the amplitude of the shape function in y but in practice it is more of a flag to shape between -ywidth/2 and ywidth/2 or not. A value of 0 results in no shaping in y. A value of 1 results in modulation between -ywidth/2 and ywidth/2 and a constant 0 outside.
Data Type: Real
Default: 1.0

phase_decay_time_steps

Description: If you wish to apply a random phase to the driver this and fwhm must be specified. These control the calculation of the random phase and it is best to consult the source as to how these are used.
Data Type: Real
Default: 0.0

fwhm

Description: If you wish to apply a random phase to the driver this and phase_decay_time_steps must be specified. These control the calculation of the random phase and it is best to consult the source as to how these are used.
Data Type: Real
Default: 0.0

Parameters Describing Kinetic Species Collision Operators. Multiple collision operators are permitted for each kinetic species. The input parameters for each collision operator for a given kinetic species are defined in separate namespaces of that kinetic species. The input parameters for the nth collision operator for the mth kinetic species will be specified in the namespace kinetic_species.m.collision_operator.n where m and n are one based. Therefore to specify the name of the 2nd collision operator of the 1st kinetic species one would use the following syntax: kinetic_species.1.collision_operator.2.name = "Pitch Angle Collision Operator"

There are currently 3 collision operators, Pitch Angle Collision Operator, Original Rosenbluth Collision Operator, and Rosenbluth Collision Operator. The Pitch Angle Collision Operator is the infinite mass limiting case of the Rosenbluth Operator. As such it is most suitable for collisions of electrons off of ions, the heavier the ions the better. As it assumes that the target has infinite mass there is no interspecies communication and the kernel is less computationally intensive than the the Rosenbluth Operator. The Original Rosenbluth Collision Operator is our initial attempt at this and it only supports self collisions with or without back reactions. The Rosenbluth Collision Operator supports both self and interspecies collisions with or without back reactions. Note that the Rosenbluth Collision Operator with interspecies collisions is very resource intensive. This is not only computationally intensive but requires significant communication. Also note that the distribution function for an ion is extremely narrow in comparison to that of electrons. This makes differentiation of an ion distribution function on the electron mesh essentially impossible for any practical electron mesh resolution. For this reason one should not attempt to use the Rosenbluth operator for electon/ion interactions. Use the Pitch Angle operator to model the effect of the ions on the electrons and ignore the small effect of the electrons on the ions.

The Vlasov-Fokker-Planck equation, which describes a weakly coupled plasma charged species a, is

$$\frac{\partial f_a}{\partial t} + \frac{\partial f_a}{\partial \boldsymbol{x}} + \boldsymbol{a}\left(\boldsymbol{x}, \boldsymbol{v}, t\right) \cdot \frac{\partial f_a}{\partial \boldsymbol{v}} = \left.\frac{\partial f_a}{\partial t}\right|_c \tag{1}$$

$$= -\sum_b C_{ab}(f_b, f_a). \tag{2}$$

Here, f_a is the density in the $\boldsymbol{x} - \boldsymbol{v}$ phase space, where $\boldsymbol{x}$ is the position, $\boldsymbol{v}$is the velocity, t is the time, $\boldsymbol{a} = \boldsymbol{a}\left(\boldsymbol{x}, \boldsymbol{v}, t\right)$ is the acceleration due to electric and magnetic fields. Each C_{ab} is a Landau-Fokker-Planck collision operator, which describes Coulomb collisions and

is given by

$$\begin{aligned} C_{ab}(f_b, f_a) &= \gamma_{ab} m_b \frac{\partial}{\partial \boldsymbol{v}} \cdot \int d\boldsymbol{v}' \overleftrightarrow{U}\left(\boldsymbol{v}-\boldsymbol{v}'\right) \cdot \left(\frac{1}{m_b}\frac{\partial}{\partial \boldsymbol{v}'} - \frac{1}{m_a}\frac{\partial}{\partial \boldsymbol{v}}\right) f_a\left(\boldsymbol{v}\right) f_b\left(\boldsymbol{v}'\right) \\ \overleftrightarrow{U}\left(\boldsymbol{v}-\boldsymbol{v}'\right) &= \frac{1}{u}\left(\overleftrightarrow{I} - \hat{\boldsymbol{u}}\hat{\boldsymbol{u}}\right) \\ \boldsymbol{u} &= \boldsymbol{v}-\boldsymbol{v}' \\ \gamma_{ab} &= 2\pi \frac{\ln \Lambda_{ab}}{m_a m_b} \frac{q_a^2}{4\pi\epsilon_0} \frac{q_b^2}{4\pi\epsilon_0} \\ \Lambda_{ab} &= \frac{r_{\max}}{r_{\min}};\ r_{\max} = \lambda_{\mathrm{D}} \end{aligned}$$

Here, m_s and q_s denote the mass and charge of species s, $r_{\max}$ is an effective screening lenth typically taken to be the Debye length, and $r_{\min}$ is a "distance of closest approach." For classical Coulomb collisions, this is typically taken to be the distance at which the thermal and electrostatic energies of colliding particles balance. In the following, we will use cgs units for which $4\pi\epsilon_0 \to 1$. In Loki, $t \to \omega_{pe}t$, $v \to v/v_{te}$, $x \to x/\lambda_{De}$ and thus the coefficient becomes for Rosenbluth Collision operator becomes:

$$RosenbluthCoef = 2\pi {q_a}^2 {q_b}^2 \frac{\ln \Lambda_{ab}}{m_a m_b}\left[\frac{N_e}{V_{te}^3 \omega_{pe}}\right] \tag{3}$$

where the normalization factors are in square brackets. For collisions of electrons from ions, the Pitch Angle Collisions operator is useful for which the ion distribution details are irrelevant, only the density, charge, and mass are needed. It should be an excellent approximation for electrons in both EPW and in IAWs. This neglects the small loss of energy of the electrons to to ions in the scattering process. Neglecting the ion velocity in the relative velocity, $\boldsymbol{U_{i,j}} = \boldsymbol{V_{i,j}} = (v^2\delta_{i,j} - v_i v_j)/v^3$ where v is the magnitude of the electron velocity. Using this in $C_{a,b}$, the integral with respect to v' is done with the result

$$C_{e,i} = -\gamma_{ab}\frac{m_b}{m_a} N_b \frac{\partial}{\partial \boldsymbol{v}} \cdot \boldsymbol{V_{i,j}} \cdot \frac{\partial}{\partial \boldsymbol{v}} f_a \tag{4}$$

The result can be compared to the Rosenbluth operator by using Eq. (3) with $q_a = e^2$, $q_b = Z_b e^2$, $m_a = m_e$, and $m_b = A_b m_p$. The pitch angle collision coefficient is:

$$PitchAngleCoef = 2\pi \frac{Z_b^2 e^4 N_b}{m_e^2} \ln \Lambda_{eb}\left[\frac{1}{V_{te}^3 \omega_{pe}}\right] \tag{5}$$

We have used this for the ion-ion coefficients in the IAW detrapping simulations and in the ion temperature equilibration study with apparent success.

Pitch Angle Collision Operator. These parameters apply to the Pitch Angle Collision Operator:

name

Description: Name of collision operator.
Data Type: String
Default: None. This input is required.

Notes: Must be "Pitch Angle Collision Operator".

collision_vfloor

Description: The collision coefficient is scaled by:
$(vthermal/v) * *3$
To avoid the singularity at v=0 vfloor is the smallest allowed value for v.
Data Type: Real
Default: None. This input is required.

collision_vthermal_dt

Description: The thermal velocity of the lighter Kinetic Species involved in this collision operator.
Data Type: Real
Default: None. This input is required.

collision_nuCoeff

Description: Unmodulated collision coefficient.
Data Type: Real
Default: None. This input is required.

collision_conservative

Description: Controls use of conservative algorithm. 1 means use conservative algorithm. Other than 1 means use non-conservative algorithm.
Data Type: Integer
Default: 1
Notes: If running with 6th order spatial accuracy, this must be 1.

collision_vel_range_lo

Description: Lower end of x and y velocity space in which collision coefficient is unmodulated. This a an array of 2 values, vxlo and vylo respectively. The collision coefficient is ramped from 1 at this velocity to 0 3 (in 4th order) or 4 (in 6th order) cells inside the velocity lower bound. This works in tandem with collision_vel_range_hi.
Data Type: Real
Default: None. This input is required.

collision_vel_range_hi

Description: Upper end of x and y velocity space in which collision coefficient is unmodulated. This a an array of 2 values, vxhi and vyhi respectively. The collision coefficient is ramped from 1 at this velocity to 0 3 (in 4th order) or 4 (in 6th order) cells inside the velocity upper bound. This works in tandem with collision_vel_range_lo.
Data Type: Real
Default: None. This input is required.

Original Rosenbluth Collision Operator. These parameters apply to the Original Rosenbluth Collision Operator:

name

Description: Name of collision operator.
Data Type: String
Default: None. This input is required.
Notes: Must be "Original Rosenbluth Collision Operator".

collision_vthermal_method

Description: Means by which vthermal is determined. One may simply enter a value, compute it via a global calculation of the mean distribution function, or compute it via a processor local calculation of the mean distribution function.
Data Type: String
Default: "input vthermal"
Notes: Must be one of "input vthermal", "local vthermal", or "global vthermal". If "input vthermal" is selected then there must be an entry for collision_vthermal and collision_alpha. If "local vthermal" or "global vthermal" are selected then there must not be an entry for collision_vthermal or collision_alpha as this is contradictory.

collision_vthermal

Description: The thermal velocity of the Kinetic Species associated with this collision operator.
Data Type: Real
Default: None. This input is required if collision_vthermal_method is "input vthermal".

collision_alpha

Description: This should be the inverse of the square of collision_vthermal.
Data Type: Real
Default: None. This input is required.

collision_nuCoeff

Description: Unmodulated collision coefficient.
Data Type: Real
Default: None. This input is required.

collision_back_reaction

Description: If true, compute back reaction terms.
Data Type: Boolean
Default: false

collision_massR

Description: Mass ratio of collided and colliding species.
Data Type: Real
Default: None. This input is required if collision_back_reaction is true.

collision_kernel_alg

Description: Collision kernel algorithm.
Data Type: String
Default: "primitive"
Notes: There are 4 collision kernel algorithms available, The "primitive" is the most robust followed by "asinc". Two others are available but both are considerably less robust: "log" and "orig".

collision_vel_range_lo

Description: Lower end of x and y velocity space in which collision coefficient is unmodulated. This a an array of 2 values, vxlo and vylo respectively. The collision coefficient is ramped from 1 at this velocity to 0 3 (in 4th order) or 4 (in 6th order) cells inside the velocity lower bound. This works in tandem with collision_vel_range_hi.
Data Type: Real
Default: None. This input is required.

collision_vel_range_hi

Description: Upper end of x and y velocity space in which collision coefficient is unmodulated. This a an array of 2 values, vxhi and vyhi respectively. The collision coefficient is ramped from 1 at this velocity to 0 3 (in 4th order) or 4 (in 6th order) cells inside the velocity lower bound. This works in tandem with collision_vel_range_lo.
Data Type: Real
Default: None. This input is required.

Rosenbluth Collision Operator. These parameters apply to the Rosenbluth Collision Operator:

name

Description: Name of collision operator.
Data Type: String
Default: None. This input is required.
Notes: Must be "Rosenbluth Collision Operator".

collision_back_reaction

Description: If true, compute back reaction terms.
Data Type: Boolean
Default: false

collision_interspecies

Description: If true, compute collisions with other species in the problem. Note that if interspecies collisions are on, each species must specify symmetric collision operators. All species must participate when any species wishes to do an interspecies collision.
Data Type: Boolean

Default: false

collision_nuCoeffInterspecies

Description: Collision coefficients with other species specified as an array. The entries in the array are the coefficients for the other species in the order of those species indices in the input deck.
Data Type: Real
Default: None. This input is required if collision_interspecies is true.

collision_self

Description: If true, compute collisions with self.
Data Type: Boolean
Default: false

collision_nuCoeffSelf

Description: Collision coefficent for self collisions.
Data Type: Real
Default: None. This input is required if collision_self is true.

modulate_collision_rates

Description: Vary collision rates (coefficient) by N/vthermal.
Data Type: Boolean
Default: False.
Notes: If true, the collision rates (coefficients) should exclude the N/vthermal factor.

collision_kernel_alg

Description: Collision kernel algorithm.
Data Type: String
Default: "primitive"
Notes: There are 4 collision kernel algorithms available, The "primitive" is the most robust followed by "asinc". Two others are available but both are considerably less robust: "log" and "orig".

collision_vel_range_lo

Description: Lower end of x and y velocity space in which collision coefficient is unmodulated. This a an array of 2 values, vxlo and vylo respectively. The collision coefficient is ramped from 1 at this velocity to 0 3 (in 4th order) or 4 (in 6th order) cells inside the velocity lower bound. This works in tandem with collision_vel_range_hi.
Data Type: Real
Default: None. This input is required.

collision_vel_range_hi

Description: Upper end of x and y velocity space in which collision coefficient is unmodulated. This a an array of 2 values, vxhi and vyhi respectively. The collision coefficient is ramped from 1 at this velocity to 0 3 (in 4th order) or 4 (in

6th order) cells inside the velocity lower bound. This works in tandem with collision_vel_range_lo.
Data Type: Real
Default: None. This input is required.

collision_diagnostic

Description: If true, compute collisional diagnostics.
Data Type: Boolean
Default: false

Parameters Describing Kinetic Species Krook Layer. Each kinetic species can define a krook layer. The input parameters of each kinetic species' krook layer are defined in separate namespaces. The namespace for the krook layer of the nth kinetic species is kinetic_species.n.krook where n is one based. Therefore to specify the power of the krook layer of the 1st kinetic species one would use the following syntax:
kinetic_species.1.krook.power = 2

power

Description: Exponent in the evaluation of each layer.
Data Type: Real
Default: 3.0
Notes: Must be > 0.

coefficient

Description: Scaling in the evaluation of each layer.
Data Type: Real
Default: 1.0
Notes: Must be > 0.

x1a

Description: The coordinate of the lower end of layer in x. If specified, a krook layer at the lower end of x will be applied otherwise it will not.
Data Type: Real
Default: No krook layer at lower end of x.
Notes: Must be > value for xmin in the domain_limits parameter. If specified, the value for the x periodic_dir must be false.

x1b

Description: The coordinate of the upper end of layer in x. If specified, a krook layer at the upper end of x will be applied otherwise it will not.
Data Type: Real
Default: No krook layer at upper end of x.
Notes: Must be > value for xmax in the domain_limits parameter. If specified, the value for the x periodic_dir must be false.

x2a

Description: The coordinate of the lower end of layer in y. If specified, a krook layer at the lower end of y will be applied otherwise it will not.
Data Type: Real
Default: No krook layer at lower end of y.
Notes: Must be > value for ymin in the domain_limits parameter. If specified, the value for the y periodic_dir must be false.

x2b

Description: The coordinate of the upper end of layer in y. If specified, a krook layer at the upper end of y will be applied otherwise it will not.
Data Type: Real
Default: No krook layer at upper end of y.
Notes: Must be > value for ymax in the domain_limits parameter. If specified, the value for the y periodic_dir must be false.

Parameters Describing Electrostatic (Poisson) Algorithm. Input parameters for the electrostatic algorithm are defined in the poisson namespace. Therefore to specify that the noise particles should be reflected at the domain boundaries one would use the following syntax:
poisson.reflect_noise_source_particles = true

apply_external_potential

Description: It is possible to specify an externally applied potential. Unlike the electric field drivers, the external potential is applied to each kinetic species. If an external potential is to be specified, then this should be set to true.
Data Type: Boolean
Default: false

tracking_particle_file

Description: It is possible to track charged particles during a simulation. The tracking particles respond to the net self consistent E field generated by the kinetic species present in the problem plus any externally applied electric fields. The initial 4D physical space location of each particle is specified in an HDF5 file. In addition, a time to start tracking each particle is specified. The name of the HDF5 file should be supplied with this input parameter. An example Matlab script to generate such a tracking particle input file is in the examples directory. The phase space coordinates of each particle are saved as part of the time history output. Note that these particles are unrelated to the kinetic species in the problem. They simply allow one to visualize the respose of a given charged particle. Tracking particle locations are also saved as restart data. When restarting a problem with tracking particles, do not change this entry in the input deck.
Data Type: String
Default: Empty string (no tracking particles specified).

noise_source_particle_file

Description: It is possible to add noise to the distribution function with noise particles. The contribution of the noise particles is added to the distribution function to determine the net charge density before solving for the electic potential and resulting electric field. The noise particles respond to the net self consistent E field generated by the kinetic species and noise particles, and any externally applied electric fields present in the problem. Input specification of noise particles is similar to tracking particles except for 2 points. First, instead of a start time there is a ramp time. This allows the strength of the particle to ramp up rather than being applied abruptly at t=0. Also there is one more parameter for each particle, a weighting factor applied to that particle's charge when determining its contribution to the net charge density. Due to the large number of noise particles expected to be used in a problem there is no time history output of them. As with tracking particles restart data is saved and you should not modify this entry in the input deck when restarting a problem.
Data Type: String
Default: Empty string (no noise source particles specified)

reflect_noise_source_particles

Description: For noise source particles you may force the particles to be reflcted at the domain boundaries. Set this flag to true to do so.
Data Type: Boolean
Default: False

number_of_processors

Description: The electrostatic portion of the algorithm is 2D and is distributed over different processors than are the 4D kinetic species. You may control this distribution through this parameter. The original intent was to allow faster solves of Poisson's equation by distributing over multiple processors and using a parallel Poisson solve. The code currently does not have access to a solver that shows speed up in parallel. However, a communication bottleneck can occur when there is a single Poisson processor. The electric field must be communicated from the Poisson processor(s) to the Vlasov processors and the charge density must be communicated from the Vlasov processors to the Poisson processor(s). With a single Poisson processor and a highly partitioned configuration space this will result in many messages being processed by the single Poisson processor. Using more Poisson processors can aleviate this problem. The testDist utility will suggest an appropriate number of Poisson processors to use.
Data Type: Integer
Default: 1

Parameters Describing Poisson Solvers. There are currently 2 Poisson solvers to choose from. One is a direct solve using the SuperLU package. The other is a FFT based solver using the FFTW package. These are intrinsically serial solvers. Therefore even when running with multiple Poisson processors each processor does the full serial

solve. At this time we do not have access to a parallel solver which shows good scaling. The performance of the SuperLU solve will become noticable poorer as the problem size increases. In particular, the factorization at the first solve will become onerous for large enough systems. The FFT based solver will also slow down with problem size but it is so fast that in practice one does not notice the slowdown. SuperLU is the default. The input parameters for the solvers are defined in a separate namespace, poisson.solver. Therefore to specify the name of the solver one would use the following syntax:
poisson.solver.name = "superlu"

SuperLU Solver. There are no control parameters for the SuperLU solver. The only input is to specify the name of the solver.

name

Description: The name of the solver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "superlu"

FFT Solver. There is only one control parameters for the FFT solver.

name

Description: The name of the solver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "fft"

discretization

Description: Nature of the discretization. There are 2 options, "order" which gives a solution at the spatial order of the problem, and "spectral" which gives a solution at machine precision.
Data Type: String
Default: "order"

Parameters Describing Electrostatic (Poisson) Algorithm External Potential Driver. The input parameters for the external potential are defined in a separate namespace, poisson.external_potential. Therefore to specify the amplitude of the electrostatic algorithm external potential driver one would use the following syntax:
poisson.external_potential.amp = 0.01

Note that, unlike the electric field drivers, the external potential is not specific to a particular kinetic species. It is applied to each of the species in the problem.

At the present time there is only one potential driver, the Shaped Ramped Cosine Potential Driver. The code is designed with an abstract driver class so that new drivers derived from the abstract driver may be easily added. The only change to existing code needed is to add the new class to the driver factory.

The driver envelope, f(t) below, was originally controlled by 2 parameters, t_ramp and t_off. It has been generalized and is now contolled by 3 parameters, t_rampup, t_hold, and

t_rampdown. The new syntax is described below. However the old syntax is still recognized by the code and is mapped to the new syntax.

The functional form of the Shaped Ramped Cosine Potential Driver is largely the same as for the Electric Field Driver:

if (t0 <= t < t0+t_rampup)
$f(t) = 0.5 + 0.5 * tanh(4.0 * (2.0 * (t - t0)/t\_rampup - 1.0))$
else if (t0+t_rampup <= t < t0+t_rampup+t_hold)
$f(t) = 0.5 + 0.5 * tanh(4.0)$
else if (t0+t_rampup+t_hold <= t < t0+t_ramp+t_hold+t_rampdown)
$f(t) = 0.5 - 0.5 * tanh(4.0 * (2.0 * (t - t0 - t\_rampup - t\_hold)/t\_rampdown - 1.0))$
else
$f(t) = 0$
if (abs(x-x0) < lwidth/2)
$g(x) = 1 - x\_shape * (sin(\pi * (x - x0)/lwidth)) * *2$
else
$g(x) = 1 - x\_shape$
if (abs(y) < ywidth/2)
$h(y) = 1 - shape * (sin(\pi * y/ywidth)) * *2$
else
$h(y) = 1 - shape$
$phi\_ext = amp * f(t) * g(x) * h(y) * cos(\pi * x/xwidth - omega * (t - t0))$

name

Description: Name of driver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "Shaped Ramped Cosine Potential Driver"

t0

Description: Time at which driver turns on.
Data Type: Real
Default: 0.0

t_rampup

Description: Ramp up time. Driver ramps up from t0 to t0+t_rampup.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

t_hold

Description: Hold time. Driver remains constant from t0+t_rampup to t0+t_rampup+t_hold.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

t_rampdown

Description: Ramp down time. Driver ramps down from t0+t_rampup+t_hold to t0+t_rampup+t_hold+t_rampdown.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

amp

Description: Amplitude of driver.
Data Type: Real
Default: 0.01

xwidth

Description: Half of a wavelength of modulation in x.
Data Type: Real
Default: 0.5
Notes: Again, the default is most likely not what you want to use.

omega

Description: Frequency of external driver.
Data Type: Real
Default: 1.0
Notes: Again, the default is most likely not what you want to use.

x0

Description: Center of shape function in x.
Data Type: Real
Default: 0.0

lwidth

Description: Half of a wavelength of shape function in x.
Data Type: Real
Default: 0.5
Notes: Again, the default is most likely not what you want to use.

x_shape

Description: This is the amplitude of the shape function in x but in practice it is more of a flag to shape between -xwidth/2 and xwidth/2 or not. A value of 0 results in no shaping in x. A value of 1 results in modulation between x0-lwidth/2 and x0+lwidth/2 and a constant 0 outside.
Data Type: Real
Default: 0.0

ywidth

Description: Half of a wavelength of shape function in y.
Data Type: Real
Default: 0.5

Notes: Again, the default is most likely not what you want to use.

shape

Description: This is the amplitude of the shape function in y but in practice it is more of a flag to shape between -ywidth/2 and ywidth/2 or not. A value of 0 results in no shaping in y. A value of 1 results in modulation between -ywidth/2 and ywidth/2 and a constant 0 outside.
Data Type: Real
Default: 1.0

Parameters Describing Electrodynamic (Maxwell) Algorithm. Input parameters for the electrodynamic algorithm are defined in the maxwell namespace. To specify the number of current drivers for the electrodynamic algorithm one would use the following syntax:
maxwell.num_current_drivers = 1

num_current_drivers

Description: It is possible to specify current drivers (sources) to be used in the evaluation of the RHS of Maxwell's equations. This indicates how many current drivers there will be.
Data Type: Integer
Default: 0

supergrid1a

Description: It is possible to specify a supergrid for the Maxwell algorithm. This is a box outside of which the electromagnetic fields are damped. The supergid is similar to and specified the same way as a Krook layer. By default the supergrid box is the entire physical domain so no damping of the electromagnetic fields occurs. This parameter is the x lower bound of the supergrid.
Data Type: Real
Default: The domain x lower bound.

supergrid1b

Description: This parameter is the x upper bound of the supergrid.
Data Type: Real
Default: The domain x upper bound.

supergrid2a

Description: This parameter is the y lower bound of the supergrid.
Data Type: Real
Default: The domain y lower bound.

supergrid2b

Description: This parameter is the y upper bound of the supergrid.
Data Type: Real
Default: The domain y upper bound.

tracking_particle_file

Description: It is possible to track charged particles during a simulation. The tracking particles respond to the dynamic electromagnetic field plus any externally applied electric fields. The initial 4D physical space location of each particle is specified in an HDF5 file. In addition, a time to start tracking each particle is specified. The name of the HDF5 file should be supplied with this input parameter. An example Matlab script to generate such a tracking particle input file is in the examples directory. The phase space coordinates of each particle are saved as part of the time history output. Note that these particles are unrelated to the kinetic species in the problem. They simply allow one to visualize the respose of a given charged particle. Tracking particle locations are also saved as restart data. When restarting a problem with tracking particles, do not change this entry in the input deck.
Data Type: String
Default: Empty string (no tracking particles specified).

noise_source_particle_file

Description: It is possible to add noise to the distribution function current density with noise particles. The contribution of the noise particles is added to the distribution function current density and any externally applied currents to determine the net current density before solving Maxwell's equations. The noise particles respond to the dynamic electrodynamic field plus any externaly applied electric field. Input specification of noise particles is similar to tracking particles except that there is one more parameter for each particle, a weighting factor applied to that particle's charge when determining its contribution to the net current density. Due to the large number of noise particles expected to be used in a problem there is no time history output of them. As with tracking particles restart data is saved and you should not modify this entry in the input deck when restarting a problem.
Data Type: String
Default: Empty string (no noise source particles specified)

reflect_noise_source_particles

Description: For noise source particles you may force the particles to be reflcted at the domain boundaries. Set this flag to true to do so.
Data Type: Boolean
Default: False

number_of_processors

Description: The electrodynamic portion of the algorithm is 2D and is distributed over different processors than are the 4D kinetic species. You may control this distribution through this parameter. The intent is to distribute the evaluation of the RHS of Maxwell's equations over multiple processors. This is most useful not for the speed up of the solution of Maxwell's equations but to avoid communication bottlenecks to and from a single Maxwell processor and the Vlasov processors. If, as one would find in a many wavelength problem, there are many partitions of x

and y then a single Maxwell processor will need to communicate with many Vlasov processors creating a bottleneck. Using more Maxwell processors can aleviate this problem. The testDist utility will suggest an appropriate number of Maxwell processors to use.

Data Type: Integer
Default: 1

Parameters Describing Electrodynamic (Maxwell) Algorithm Current Drivers. Multiple current drivers are permitted for the electrodynamic algorithm. The input parameters for each current driver for the electrodynamic algorithm are defined in separate namespaces of the electrodynamic algorithm. The input parameters for the nth current driver will be specified in the namespace maxwell.current_driver.n where n is one based. Therefore to specify the name of the 2nd current driver of the electrodynamic algorithm one would use the following syntax:
maxwell.current_driver.2.name = "Shaped Ramped Cosine Current Driver"

At the present time there are three current drivers, the Shaped Ramped Cosine Current Driver, the Unidirectional Current Driver, and the Gaussian Current Driver. The code is designed with an abstract current driver class so that new drivers derived from the abstract current driver may be easily added. The only change to existing code needed is to add the new class to the current driver factory.

The driver envelope, f(t) below, was originally controlled by 2 parameters, t_ramp and t_off. It has been generalized and is now contolled by 3 parameters, t_rampup, t_hold, and t_rampdown. The new syntax is described below. However the old syntax is still recognized by the code and is mapped to the new syntax.

Shaped Ramped Cosine Current Driver. The Shaped Ramped Cosine Current Driver is a delta function in x or y. It is applied only at x=x0 or y=x0. Its functional form is largely the same as for the Electric Field Driver:
if (t0 <= t < t0+t_rampup)
$f(t) = 0.5 + 0.5 * tanh(4.0 * (2.0 * (t - t0)/t\_rampup - 1.0))$
else if (t0+t_rampup <= t < t0+t_rampup+t_hold)
$f(t) = 0.5 + 0.5 * tanh(4.0)$
else if (t0+t_rampup+t_hold <= t < t0+t_rampup+t_hold+t_rampdown)
$f(t) = 0.5 - 0.5 * tanh(4.0 * (2.0 * (t - t0 - t\_rampup - t\_hold)/t\_rampdown - 1.0))$
else
$f(t) = 0$
if (x == x0)
$g(x) = 1$
else
$g(x) = 0$
if (abs(y) < width/2)
$h(y) = 1 - shape * (sin(\pi * y/width)) * *2$
else
$h(y) = 1 - shape$

$J\_ext = J\_0 * f(t) * g(x) * h(y) * cos(omega * (t - t0))$

name

Description: Name of driver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "Shaped Ramped Cosine Current Driver"

apply_dir

Description: Indicates which component of the current this driver controls.
Data Type: String
Default: None. This input is required.
Notes: Must be either "x" or "X" if y0 is specified, "y" or "Y" if x0 is specified or "z", or "Z".

t0

Description: Time at which driver turns on.
Data Type: Real
Default: 0.0

t_rampup

Description: Ramp up time. Driver ramps up from t0 to t0+t_rampup.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

t_hold

Description: Hold time. Driver remains constant from t0+t_rampup to t0+t_rampup+t_hold.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

t_rampdown

Description: Ramp down time. Driver ramps down from t0+t_rampup+t_hold to t0+t_rampup+t_hold+t_rampdown.
Data Type: Real
Default: 10.0
Notes: Although there is a default it is almost certainly not what you want to use.

J_0

Description: Amplitude of driver.
Data Type: Real
Default: 0.01

omega

Description: Frequency of external driver.
Data Type: Real

Default: 1.0
Notes: Although there is a default it is almost certainly not what you want to use.

x0

Description: Location in x of current source.
Data Type: Real
Default: 0.0
Notes: Either x0 or y0 must be specified. Use x0 for a y-z current sheet.

y0

Description: Location in y of current source.
Data Type: Real
Default: 0.0
Notes: Either x0 or y0 must be specified. Use y0 for an x-z current sheet.

width

Description: Half of a wavelength of modulation in x or y.
Data Type: Real
Default: None. This input is required.

shape

Description: This is the amplitude of the shape function in x or y but in practice it is more of a flag to shape between -(x/y)width/2 and (x/y)width/2 or not. A value of 0 results in no shaping. A value of 1 results in modulation between -(x/y)width/2 and (x/y)width/2 and a constant 0 outside.
Data Type: Real
Default: 1.0

Unidirectional Current Driver. The Unidirectional Current Driver propagates an EM field in one direction only. It is described as a segment of a current sheet. The sheet is infinite in the z direction and has finite user defined extent in the x and y directions. The user specifies the two end points of the sheet in the x-y plane via a "from" and a "to" point. The resultant field propagates to the right when looking in the direction of the "from" point to the "to" point. Thus the order of these points is important. Note that the points may extend beyond the problem domain thereby creating a current sheet covering the entire problem domain. It is also worth noting that current sheets which do not cover the entire problem domain will exhibit diffraction at their end points.

The direction of the current in the sheet is controlled by 2 rotations. The current is initially taken to be oriented along y. The first rotation is a user specified polar angle in the y-z plane relative to the y axis. This controls the amount of out of plane (along the z direction) current. The out of plane current is proportional to sin(polar angle) and the in plane current is proportional to cos(polar angle). The second rotation is implied by the user specified end points of the sheet. The in plane current is appotioned into x and y currents based on the angle of this implied rotation. These rotations along with the user defined magnitude of the current fully specify the x, y, and z current components.

There is a spatial and temporal modulation of the current amplitude. It is spatially modulated by a Gaussian of the perpendicular distance from the current sheet. The user has control over the width of this Gaussian. The temporal modulation is the same tanh based modulation as is in the Shaped Ramped Cosine Current Driver described above. As with that driver, the older t_ramp, t_hold control of the envelope has been replaced with a 3 parameter control using t_rampup, t_hold, and t_rampdown. The older syntax is still recognized and is mapped into the new syntax.

The magnitude of the generated E field is equal to the current magnitude, the J_0 user input. This implies that the magnitude of the generated B field is equal to J_0/speed of light.

name

Description: Name of driver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "Unidirectional Current Driver"

omega

Description: Frequency of external driver.
Data Type: Real
Default: None. This input is required.

J_0

Description: Amplitude of driver.
Data Type: Real
Default: None. This input is required.

t0

Description: Time at which driver turns on.
Data Type: Real
Default: None. This input is required.

t_rampup

Description: Ramp up time. Driver ramps up from t0 to t0+t_rampup.
Data Type: Real
Default: None. This input is required.

t_hold

Description: Hold time. Driver remains constant from t0+t_rampup to t0+t_rampup+t_hold.
Data Type: Real
Default: None. This input is required.

t_rampdown

Description: Ramp down time. Driver ramps down from t0+t_rampup+t_hold to t0+t_rampup+t_hold+t_rampdown.
Data Type: Real

Default: None. This input is required.

polar_angle

Description: Controls out of plane current by sin(polar_angle).
Data Type: Real
Default: None. This input is required.

from_pt

Description: One of the end points of the current sheet. This is an array of 2 real values. The first is the x coordinate of the point and the second is the y coordinate.
Data Type: Real
Default: None. This input is required.
Notes: Which point is identified as the from vs the to point is important as this determines the propagation direction of the EM field as discussed above.

to_pt

Description: One of the end points of the current sheet. This is an array of 2 real values. The first is the x coordinate of the point and the second is the y coordinate.
Data Type: Real
Default: None. This input is required.
Notes: Which point is identified as the from vs the to point is important as this determines the propagation direction of the EM field as discussed above.

fwhm

Description: Specifies the full width at half maximum of the spatial Gaussian current amplitude modulation.
Data Type: Real
Default: None. This input is required.

Gaussian Current Driver. The Gaussian Current Driver generates an oscillating current in z which is a delta function in x or y. It is applied only at x=x0 producing a y-z current sheet or at y=y0 producing a x-z current sheet. The resultant wave propagates along x for a y-z current sheet and along y for a x-z current sheet. The amplitude of the current sheet varies along y for a y-z current sheet and along x for a x-z current sheet. In addition, the phase of the oscillation varies in a similar manner. These variations in amplitude and phase produce a wave with a focal point a user selected distance from x0 or y0. The wave propagates outwardly and symmetrically from the point at which the driver is applied. The formulation of this driver assumes the paraxial beam approximation. This requires that fwhm >> lambda. If this condition is not met then beam focusing will be poor or non-existant.

name

Description: Name of driver.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "Gaussian Current Driver"

x0

Description: Location in x of current source.
Data Type: Real
Default: None. This input is required.
Notes: Either x0 or y0 must be specified. Use x0 for a y-z current sheet.

y0

Description: Location in y of current source.
Data Type: Real
Default: None. This input is required.
Notes: Either x0 or y0 must be specified. Use y0 for an x-z current sheet.

focal_pt

Description: Distance from x0/y0 for best focus of wave.
Data Type: Real
Default: None. This input is required.

fwhm

Description: The full width at half maximum of the wave at the location of best focus.
Data Type: Real
Default: None. This input is required.
Notes: This parameter and omega must be chosen such that fwhm >> lambda.

nenc

Description: Reference density giving the appropriate refractive index for a homogenous system.
Data Type: Real
Default: None. This input is required.

omega

Description: Frequency of external driver.
Data Type: Real
Default: None. This input is required.
Notes: This parameter and fwhm must be chosen such that fwhm >> lambda.

J_0

Description: Amplitude of driver.
Data Type: Real
Default: None. This input is required.

t0

Description: Time at which driver turns on.
Data Type: Real
Default: None. This input is required.

t_rampup

Description: Ramp up time. Driver ramps up from t0 to t0+t_rampup.

Data Type: Real
Default: None. This input is required.

t_hold

Description: Hold time. Driver remains constant from t0+t_rampup to t0+t_rampup+t_hold.
Data Type: Real
Default: None. This input is required.

t_rampdown

Description: Ramp down time. Driver ramps down from t0+t_rampup+t_hold to t0+t_rampup+t_hold+t_rampdown.
Data Type: Real
Default: None. This input is required.

Parameters Describing Electrodynamic (Maxwell) Algorithm Field Initial Conditions. It is possible to initialize EM waves using this syntax. The field initial conditions are optional. For each wave to be initialized in this manner it is necessary to specify initial conditions on both E and B. The input parameters for each of these initializers are defined in separate namespaces of the electromagnetic algorithm. The input parameters for the nth field initializer (n=1,2 for E and B) will be specified in the namespace maxwell.em_ic.n. Therefore if there are 2 waves to initialize the specification of the xamps of the 2nd field initializer of the electrodynamic algorithm would use the following syntax:
maxwell.em_ic.2.xamp = 0.0001 0.002

At the present time there is only one form of field initializer, the SimpleEMIC. The code is designed with an abstract field initializer class so that new initializers derived from the abstract initializer may be easily added. The only change to existing code needed is to add the new class to the field initializer factory.

The SimpleEMIC is really simple. It just initializes E or B to a standing wave:
$Ex = xamp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$
$Ey = yamp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$
$Ez = zamp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$

name

Description: Name of field initializer.
Data Type: String
Default: None. This input is required for each initial condition specified.
Notes: Currently must be "SimpleEMIC"

field

Description: Indicated which field, E or B, is being initialized.
Data Type: String
Default: None. This input is required for each initial condition specified.
Notes: Must be "e", "b", "E", or "B".

xamp

Description: Specifies the x amplitude of the field being initialized for each wave. This input is an array of number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

yamp

Description: Specifies the y amplitude of the field being initialized for each wave. This input is an array of number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

zamp

Description: Specifies the z amplitude of the field being initialized for each wave. This input is an array of number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

x_wave_number

Description: Wave number in x of the field being initialized for each wave. This input is an array of number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

y_wave_number

Description: Wave number in y of the field being initialized for each wave. This input is an array of number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

phase

Description: Phase of the field being initialized for each wave. This input is an array of number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

Parameters Describing Electrodynamic (Maxwell) Algorithm Drift Velocity Initial Conditions. It is possible to initialize the transverse drift velocity of each species with this syntax. The transverse drift velocity initial conditions are optional but, if specified, there must be an intializer for each species. The input parameters for each of these initializers are defined in separate namespaces of the electromagnetic algorithm. The input parameters for the transverse drift velocity of the nth kinetic species will be specified in the namespace maxwell.vel_ic.n where n is one based and indicates the kinetic species. Each parameter is an array whose length is the number of waves initialized with the EM field initialization. Therefore to specify the amp of the 2nd kinetic species' transverse drift velocity initializer of the electrodynamic algorithm when 2 waves have been initialized one

would use the following syntax:
maxwell.vel_ic.2.amp = 0.0001 0.003

At the present time there is only one transverse drift velocity initializer, the SimpleVELIC. The code is designed with an abstract transverse drift velocity initializer class so that new initializers derived from the abstract initializer may be easily added. The only change to existing code needed is to add the new class to the transverse drift velocity initializer factory.

The SimpleVELIC is really simple. It just initializes vz to a standing wave:
$vz = amp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$

name

Description: Name of initializer.
Data Type: String
Default: None. This input is required.
Notes: Currently must be "SimpleVELIC"

amp

Description: Specifies the amplitude of the transverse drift velocity being initialized by each wave. This input is an array number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

x_wave_number

Description: Wave number of the transverse drift velocity in x being initialized by each wave. This input is an array number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

y_wave_number

Description: Wave number of the transverse drift velocity in y being initialized by each wave. This input is an array number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

phase

Description: Phase of the transverse drift velocity being initialized by each wave. This input is an array number of waves values.
Data Type: Real
Default: None. This input is required for each initial condition specified.

Parameters Describing RestartManager. Several parameters control reading and writing restart files. They are defined in the restart namespace. Therefore to specify the restart write_directory one would use the following syntax:
restart.write_directory = "my_dump_file_directory"

step_interval

Description: This specifies the frequency in time steps at which restart dumps will be written. Restart dumps are written into the directory specified by the write_directory input parameter.
Data Type: Integer
Default: -1 which implies that the frequency for writing restart dumps is not determined by time steps.
Notes: You must specify one of step_interval or time_interval. Both may not be specified. Must be > 0.

time_interval

Description: This specifies the frequency in simulation time at which restart dumps will be written. Restart dumps are written into the directory specified by the write_directory input parameter.
Data Type: Real
Default: -1.0 which implies that the frequency for writing restart dumps is not determined by simulation time.
Notes: You must specify one of step_interval or time_interval. Both may not be specified. Must be > 0.

write_directory

Description: Indicates the directory into which all restart dumps are written.
Data Type: String
Default: "."
Notes: Although there is a default it is likely not what you want.

read_directory

Description: If start_from_restart is true then you must specify where to find the restart files with this input.
Data Type: String
Default: "."
Notes: Although there is a default it is likely not what you want. This will be what write_directory was on the run from which you want to restart. If start_from_restart is false this input is not needed.

Advanced Parameters. There are a few parameters that, although they may be set by the user, they should be used with great care if at all. Some of these parameters control untested capabilities. Some are hooks that may be needed for future capabilities. And others are algorithmic knobs intended primarily for developmental experimentation. Although they are described here for completeness, we generally suggest that that not be set in user input decks.

General Simulation Parameters

do_new_algorithm

Description: If true, use derivative based calculations. Otherwise use original flux based calculations.
Data Type: Boolean

Default: true

avWeak

Description: Weak artificial dissipation term in evaluation of RHS of Maxwell's equations.
Data Type: Real
Default: 0.0

avStrong

Description: Strong artificial dissipation term in evaluation of RHS of Maxwell's equations.
Data Type: Real
Default: 1.0

Output Files

Loki generates both restart files plot files. The restart files contain the state data necessary to restart a run from the point at which the file was written. Essentially this is the distribution functions and simulation time. These files are written to the directory that was specified in the restart.write_directory input. The names of the plot files and the frequency at which they are written are controlled by the save_data, restart.write_directory, save_times, and sequence_write_times input parameters. All output files generated by Loki are HDF5 files. Any of the HDF5 binaries may be used to examine the files or you may write your own HDF5 application to manipulate them. The matlab HDF5 interface may also be used to analyze these files.

It is important to note that Loki strictly enforces output file write times. The code will adjust the simulation time step in order to ensure that plot files are written at the exact time interval specified by the user. It is therefore safe to Fourier transform Loki output as the the time intervall of this data is constant.

Post-Processing Output Files. Once the post-processor has been run, several new files will be generated. For each restart dump a serialized output of the distribution functions will be written. If the prefix arg to the post-processor was xxx then for the n restart dumps xxx_dist_n.hdf and xxx_dist_n.hdf.g0 will be generated. The first file contains meta-data and the latter contains the bulk data, essentially the distribution functions.

All the plot files will be serialized into 2 new files. Again, if the arg to the post-processor was xxx then xxx_fields.hdf and xxx_timeSeries.hdf will be generated. The _fields file contains all the 2D fields generated by Loki. The _timeSeries file contains all the 0D time histories generated by Loki.

2D Field Output. The 2D fields produced by Loki depend whether a Vlasov-Poisson or Vlasov-Maxwell system has been run. In the case of a Vlasov-Poisson system, the following plots are generated:

(1) Ex

(2) Ey
If plot_ke_fluxes is true then for each Kinetic Species:
(3) Kinetic energy flux at vx low velocity boundary
(4) Kinetic energy flux at vx high velocity boundary
(5) Kinetic energy flux at vy low velocity boundary
(6) Kinetic energy flux at vy high velocity boundary

Therefore, if there are Ns species, there will be 2 2D fields generated by default and $4*Ns+2$ 2D fields generated if plot_ke_fluxes is true for a Vlasov-Poisson System.

In the case of a Vlasov-Maxwell system, the following plots are generated:

(1) Ex
(2) Ey
(3) Ez
(4) Bx
(5) By
(6) Bz
If plot_ke_fluxes is true then for each Kinetic Species:
(7) Transverse drift velocity
(8) Kinetic energy flux at vx low velocity boundary
(9) Kinetic energy flux at vx high velocity boundary
(10) Kinetic energy flux at vy low velocity boundary
(11) Kinetic energy flux at vy high velocity boundary

Therefore, if there are Ns species, there will be 6 2D fields generated by default and $5*Ns+6$ 2D fields generated if plot_ke_fluxes is true for a Vlasov-Maxwell system.

0D Time Histories. The 0D time histories produced by Loki also depend on the system that has been run. In the case of a Vlasov-Poisson system, the following time histories are generated:

(1) E_max, the maximal value of $\sqrt{(E_x^2 + E_y^2)}$
(2) norm E, the integral of $\sqrt{(E_x^2 + E_y^2)}$
(3) Ex_max, the maximal value of $abs(E_x)$
(4) Ey_max, the maximal value of $abs(E_y)$
(5) field_energy, the integral of $0.5 * (E_x^2 + E_y^2)$
For each probe:
(6) Ex_probeN where N is the 0-based probe index, E_x at the probe
(7) Ey_probeN where N is the 0-based probe index, E_y at the probe
For each tracking particle:
(8) particleN_x where N is the 0-based particle index, particle x position
(9) particleN_y where N is the 0-based particle index, particle y position
(10) particleN_vx where N is the 0-based particle index, particle x velocity
(11) particleN_vy where N is the 0-based particle index, particle y velocity
For each kinetic species:

(12) speciesname_ke, the integrated species total kinetic energy
(13) speciesname_ke_x, the integrated species kinetic energy in x
(14) speciesname_ke_y, the integrated species kinetic energy in y
(15) speciesname_px, the integrated species x momentum
(16) speciesname_py, the integrated species y momentum
(17) speciesname_xlo_flux, KE flux through x low physical boundary
(18) speciesname_xhi_flux, KE flux through x high physical boundary
(19) speciesname_ylo_flux, KE flux through y low physical boundary
(20) speciesname_yhi_flux, KE flux through y high physical boundary
(21) speciesname_vxlo_flux, KE flux through vx low velocity boundary
(22) speciesname_vxhi_flux, KE flux through vx high velocity boundary
(23) speciesname_vylo_flux, KE flux through vy low velocity boundary
(24) speciesname_vyhi_flux, KE flux through vy high velocity boundary
(25) speciesname_ke_e_dot, sum of the field driver energy transfer to species
(26) speciesname_driver_time_envel, sum of the field driver time envelopes

Therefore, if there are Ns species, Np probes, and Nt tracking particles there will be $5 + 2 * Np + 4 * Nt + 15 * Ns$ 0D time histories generated for a Vlasov-Poisson system.

In the case of a Vlasov-Maxwell system, the following plots are generated:

(1) E_max, the maximal value of $\sqrt{(Ex^2 + Ey^2 + Ez^2)}$
(2) norm E, the integral of $\sqrt{(Ex^2 + Ey^2 + Ez^2)}$
(3) Ex_max, the maximal value of $abs(E_x)$
(4) Ey_max, the maximal value of $abs(E_y)$
(5) Ey_max, the maximal value of $abs(E_z)$
(6) E_tot, the integral of $0.5 * (Ex^2 + Ey^2 + Ez^2)$
(7) B_max, maximal value of $\sqrt{(Bx^2 + By^2 + Bz^2)}$
(8) norm B, the integral of $\sqrt{(Bx^2 + By^2 + Bz^2)}$
(9) Bx_max, the maximal value of $abs(B_x)$
(10) By_max, the maximal value of $abs(B_y)$
(11) Bz_max, the maximal value of $abs(B_z)$
(12) B_tot, the integral of $0.5 * (Bx^2 + By^2 + Bz^2)$

For each probe:

(13) Ex_probeN where N is the 0-based probe index, E_x at the probe
(14) Ey_probeN where N is the 0-based probe index, E_y at the probe
(15) Ez_probeN where N is the 0-based probe index, E_z at the probe
(16) Bx_probeN where N is the 0-based probe index, B_x at the probe
(17) By_probeN where N is the 0-based probe index, B_y at the probe
(18) Bz_probeN where N is the 0-based probe index, B_z at the probe
(19) vz_probeN_species_M where N is the 0-based probe index and M is the 0-based species index, for each kinetic species: vz at the probe

For each tracking particle:

(20) particleN_x where N is the 0-based particle index, particle x position
(21) particleN_y where N is the 0-based particle index, particle y position

(22) particleN_vx where N is the 0-based particle index, particle x velocity
(23) particleN_vy where N is the 0-based particle index, particle y velocity
For each kinetic species:
(24) speciesname_ke, the integrated species total kinetic energy
(25) speciesname_ke_x, the integrated species kinetic energy in x
(26) speciesname_ke_y, the integrated species kinetic energy in y
(27) speciesname_px, the integrated species x momentum
(28) speciesname_py, the integrated species y momentum
(29) speciesname_xlo_flux, KE flux through x low physical boundary
(30) speciesname_xhi_flux, KE flux through x high physical boundary
(31) speciesname_ylo_flux, KE flux through y low physical boundary
(32) speciesname_yhi_flux, KE flux through y high physical boundary
(33) speciesname_vxlo_flux, KE flux through vx low velocity boundary
(34) speciesname_vxhi_flux, KE flux through vx high velocity boundary
(35) speciesname_vylo_flux, KE flux through vy low velocity boundary
(36) speciesname_vyhi_flux, KE flux through vy high velocity boundary
(37) speciesname_ke_e_dot, sum of the field driver energy transfer to species
(38) speciesname_driver_time_envel, sum of the field driver time envelopes

Therefore, if there are Ns species, Np probes, and Nt tracking particles there will be $12 + Np * (6 + Ns) + 4 * Nt + 15 * Ns$ 0D time histories generated for a Vlasov-Maxwell system.

testDist Utility

This is a serial utility that takes 2 arguments, a proposed number of Vlasov processors and the path to an input deck. The utility determines the decomposition of each kinetic species in the problem and, for each species, prints out the smallest partition in each dimension. If a partition is smaller that the required minimum, the stencil width, a warning will be printed.

There are some general guidelines for choosing a good processor count. The code will assign processors to each kinetic species based on its proportion of the total Vlasov work load. The work load of a species is simply the product of its 4 phase space dimensions. Thus, if there are 2 species of equal size each will recieve 1/2 the Vlasov processors. Similarly, if there are 2 species and one is 2x the size of the other the smaller species will receive 1/3 the Vlasov processors and the other 2/3. In general, the ith species will receive n_i/m of the total number of Vlasov processors. It is best if the total number of Vlasov processors, P_v, contains a factor of m so that $P_v = m * P_0$. This means that the ith kinetic species will receive $n_i * P_0$ processors. Further, making P_0 a power of 2 gives the partitioning algorithm maximum flexibility in partitioning each dimension thereby lessening the possibility of overdecomposing a dimension. For example 2 factors of 2 allows the algorithm to divide 2 comparably sized dimensions once each or one very large dimension twice. However one factor of 5 provides no flexibility to the algorithm. It can only divide the largest dimension by 5 which may be too much.

Test Infrastructure

The test infrastructure is a Makefile driven set of tests. It is largely a regression test suite in which baselines are located in /usr/gapps/valhalla/LOKI/BASELINES/your_test_name. Each test resides in a sub-directory of the test directory. The necessary components of a test are the input file; Makefile.in which specifies how to run and difference the test; and a set of files to control checkTests, the differencing tool. Adding a new test consists of creating a new sub-directory of the test directory for the new test to live in, adding the files mentioned above to this new directory, and editing test/Makefile.in to add the new test to the list of existing tests. With the obvious exception of the input file, the rest of this can be done by copying and adjusting the files associated with an existing test.

The intent of the test infrastructure is to place control of how to run a test and how to verify its results in the hands of the developer. The input file is entirely up to the developer and specifies the physics and code features to be exercised. The Makefile is where you specify how the problem will be decomposed by specifying the number of processors on which it will run. The expectation, although this is not required, is that following execution of Loki the post-processor will be run followed by checkTests which will verify consistency with vetted baseline results.

The tool, checkTests, will do the following:

(1) Difference developer specified species distribution functions reporting any which differ from the baseline by a developer specified relative tolerance.
(2) Difference developer specified fields reporting any which differ from the baseline by a developer specified relative tolerance.
(3) Difference developer specified timeseries reporting any which differ from the baseline by a developer specified relative tolerance.

To do this checkTests reads information from a collection of files all of which are specific to the test being analyzed and under the control of the developer. One of these files is a master file which is taken as the tool's single argument. This in turn contains the names of the other files referenced by checkTests. Note that all strings in all of these files should contain no white space. The information in the master file is as follows and all entries appear together on a single line:

(1) Name with no spaces of the file containing information about which species distribution functions are to be differenced and each threshold relative tolerance.
(2) Name with no spaces of the file containing information about which fields are to be differenced and each threshold relative tolerance.
(3) Name with no spaces of the file containing information about which timeseries are to be differenced and each threshold relative tolerance.
(4) Name with no spaces of the baseline directory.
(5) Base name with no spaes of the the files produced by the test. This is essentially restart.write_directory from the test input file.
(6) The index of the distribution function dump to examine.
(7) A 1 if collision diagnostics fields and time series should be examined. Enter 0 otherwise.

(8) If the collision diagnostics value is 1 you must include the name with no white spaces of the file containing information about which collision diagnostic timeseries are to be differenced and each threshold relative tolerance.
(9) If the collision diagnostics value is 1 you must include the name with no white spaces of the file containing information about which collision diagnostic fields are to be differenced and each threshold relative tolerance.

The format of each of the auxilliary files mentioned above is the same. Each contains a series of entity name and threshold relative tolerance pairs. Each name and tolerance appear on separate lines. As the names of the fields and timeseries generated by Loki have embedded spaces, these names may include spaces and must match one of Loki's outputs. For example, should a test contain 2 species, "electron" and "ion", and you wish to difference both, the file describing the species to difference will look like:
electron
1.0e-15
ion
1.0e-13
If there are entities which are either not of interest or are not highly reproducable then they may simply be omitted from the appropriate control file. Should you not wish to difference anything from a given class (fields for example) simply keep that file empty.