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 1D 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.

Building Loki

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

Date: August 18, 2017.
LLNL-SM-737619.

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 except for A++P++ and Overture have a traditional notion of separate build and install locations. Thus, for all packages except these two you may untar and build wherever you wish. The installation process places the necessary components into the requested installation location. A++P++ and Overture must be built in-place. Hence you must untar and build these packages exactly where they will be installed.

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) To avoid confusion should you end up with multiple builds of Overture that need different versions of METIS and PARMETIS it is useful to do 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 $INST_DIR/PETSc_3.4.5
(5) ./configure –configModules=PETSc.Configure
   –optionsModule=PETSc.compilerOptions –PETSC_ARCH=linux-gnu-opt
   –with-mpi-dir="/usr/tce/packages/mvapich2/mvapich2-2.2-gcc-4.9.3"
   –with-debugging=0 –with-matlab=0
   –with-superlu-include="$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) cd $PETSC_INST
   ln -s ../SUPERLU_4.3 SUPERLU
   ln -s ../SUPERLU_DIST_3.3-p1 SUPERLU_DIST

Build A++P++:
Unlike the other external packages, A++P++ does not install the necessary pieces of it's distribution and your build of it (libs and includes) to a separate installation directory. Instead it does an "in place" installation copying these pieces to a location embedded in the distribution itself. So you need to untar and build this package where you want it installed.

(1) setenv A++P++_DIR $INST_DIR/A++P++-0.8.0
(2) cd $INST_DIR
(3) tar xvzf A++P++-0.8.0.tar.gz
(4) cd A++P++-0.8.0
(5) Set the MPI_ROOT environment variable to wherever your MPI installation lives. We needed:
    setenv MPI_ROOT /usr/tce/packages/mvapich2/mvapich2-2.2-gcc-4.9.3
(6) ./configure –enable-PXX –prefix="$A++P++_DIR" –enable-SHARED_LIBS
    –with-mpi-include="-I's needed for mpi"
    –with-mpi-lib-dirs="-L's and -Wl's needed for mpi"
    –with-mpi-libs="-l's needed for mpi"
    –without-PADRE –disable-mpirun-check
    You may want –with-mpirun and not –disable-mpirun-check depending on the nature of your machine.
(7) make
(8) make install

Build Overture:
You need a version of Overture patched to support 6th order solution accuracy. Like A++P++ Overture does not have a traditional notion of an installation location so you will need to untar and build Overture where you want it installed. Also, Overture requires HDF5 which was already built on all machines that we have used. If this is not the case where you are building Loki then you will need to download and build parallel HDF5. Most of these instructions are a repetition of those found in Overture's Installation Instructions document.

(1) cd $INST_DIR
(2) tar xvzf OV.tar.gz
(3) mv Overture_dist Overture.6thorderfix
(4) cd Overture.6thorderfix
(5) Assuming that HDF5 is located in $HDF5_DIR, that OpenGL libs and headers are found in lib64 and include subdirectories of OPENGL_DIR set up the following soft links:
    (a) ln -s $A++P++_DIR/P++/install A++
    (b) ln -s $HDF5_DIR HDF
    (c) ln -s $OPENGL_DIR OpenGL
    (d) ln -s $PETSC_INST PETSc

(6) Set up the following environment variables. These are the settings for a TOSS3 machine:
   (a) setenv MPI_ROOT /usr/tce/packages/mvapich2/mvapich2-2.2-gcc-4.9.3
   (b) setenv XLIBS /usr
   (c) setenv OpenGL /usr
   (d) setenv MOTIF /usr
   (e) setenv HDF $HDF5_DIR
   (f) setenv APlusPlus $A++P++_DIR/P++/install
   (g) setenv LAPACK /usr/lib64
   (h) setenv Overture $INST_DIR/Overture.6thorderfix
   (i) setenv LD_LIBRARY_PATH /usr/lib64:$HDF/lib:
   $INST_DIR/Overture.6thorderfix/lib:$A++P++_DIR/P++/install/lib
   (j) setenv PPlusPlus $A++P++_DIR/P++/install
   (k) setenv PETSC_ARCH linux-gnu-opt
   (l) setenv PETSC_DIR $PETSC_INST
   (m) setenv PETSC_LIB $PETSC_INST/lib
(7) ./configure opt parallel debugFlag=”-g” precision=double CC=g++ cc=gcc FC=gfortran
(8) We found it necessary to edit DataBase/Makefile to put -I$(HDF)/include before $(CCFLAGS). We also found it necessary to edit Ogshow/Makefile to put -I$(HDF)/include before $(CCFLAGS) and $(CFLAGS). Finally, we found it necessary to edit bin/Makefile to get rid of -lmpio These may be fixed in newer versions of Overture.
(9) make

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. Thus to post-process a run that generated output with the root name planeEPW, one would execute one of the following 2 commands:

```
vlasovPoisson4D -prefix=planeEPW
```
or
```
vlasovPoisson4D -prefix=planeEPW -skip_dists
```

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 = 3.14159265358979323846 26;
$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.

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 controlled by the show_file_name input parameter below.
Data Type: Real
Default: 1.0

save_times

Description: Save 2D configuration space arrays at multiples of this simulation time. This data is written to files whose naming is controlled by the show_file_name input parameter below.
Data Type: Real
Default: 1.0
Note: 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.

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

show_file_name

Description: Gives the root naming of plot files.
Data Type: String
Default: ex.show
Notes: If you intend to use any Overture visualization or run the post-processing tool this should be of the form xxx.show. If the post-processing tool is to be run, xxx should be the same as what is specified in the restart.write_directory input.

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 = $\mathrm{N_{vx}}$ $\mathrm{N_{vy}}$
Data Type: Integer
Default: None. This input is required.

Parameters Describing Kinetic Species Initial Conditions. There are 4 ways that initial conditions may be specicied. The first 3 are through the "Perturbed Maxwellian", "Landau damping", and "Maxwellian with noise" initial conditions. In this case the user provides a set of input parameters and the code computes the complete initial condition. The fourth 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.

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, and f is the distribution function. vflowx and vflowy are the vflowinitx and vflowinity input parameters for the species being computed.

Common Definitions
$vx = vx0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + vflowx$
$vy = vy0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + vflowy$

Perturbed Maxwellian
$f(x1, x2, x3, x4) = frac*alpha*beta/(2.0*\pi)*exp(-0.5*((alpha*(x3-vx))**2+(beta*(x4-vy))**2))*(1.0+A*cos(kx1*x1+spatial\_phase)*cos(ky1*x2+spatial\_phase)+B*cos(kx2*x1+spatial\_phase)+C*cos(ky2*x2+spatial\_phase))$

Landau damping
$f(x1, x2, x3, x4) = frac*alpha*beta/(2.0*\pi)*exp(-0.5*((alpha*(x3-vx))**2+(beta*(x4-vy))**2))*(1.0+A*cos(kx1*x1+ky1*x2+spatial\_phase))$

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
$f(x1, x2, x3, x4) = noise * frac * alpha * beta/(2.0 * \pi) * exp(-0.5 * ((alpha * (x3 - vx)) * *2 + (beta * (x4 - vy)) * *2))$

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".

alpha

Description: Factor to control width of Maxwellian in vx.

Data Type: Real
Default: 1.0

beta

Description: Factor to control width of Maxwellian in vy.
Data Type: Real
Default: 1.0

vx0

Description: Amplitude of spatial perturbation of Maxwellian in vx.
Data Type: Real
Default: 0.0

vy0

Description: Amplitude of spatial perturbation of Maxwellian in vy.
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: 0.0

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.

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 velocity space dependence which will be described below. 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.

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 coordinate, x4 is the vy coordinate, the externally specified 2D distribution is extDist(x1, x2), and f is the full 4D distribution function. vflowx and vflowy are the vflowinitx and vflowinity input parameters for the species being computed.

$vx = vx0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + vflowx$
$vy = vy0 * cos(x\_wave\_number * x1 + y\_wave\_number * x2 + phase) + vflowy$
$f(x1, x2, x3, x4) = alpha * beta/(2.0 * \pi) * exp(-0.5 * ((alpha * (x3 - vx)) * *2 + (beta * (x4 - vy)) * *2)) * extDist(x1, x2)$

name

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

file_name

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

alpha

Description: Factor to control width of Maxwellian in vx.
Data Type: Real
Default: 1.0

beta

Description: Factor to control width of Maxwellian in vy.
Data Type: Real
Default: 1.0

vx0

Description: Amplitude of spatial perturbation of Maxwellian in vx.

Data Type: Real
Default: 0.0

vy0

Description: Amplitude of spatial perturbation of Maxwellian in vy.
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 functional form of the Shaped Ramped Cosine Driver is given here.

```
if (t0 <= t < t0+t_ramp)
    $f(t) = 0.5 + 0.5 * tanh(4.0 * (2.0 * (t - t0)/t\_ramp - 1.0))$
else if (t0+t_ramp <= t < t0+t_ramp+t_off)
    $f(t) = 0.5 - 0.5 * tanh(4.0 * (2.0 * (t - t0 - t\_off)/t\_ramp - 1.0))$
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
    $h(y) = 1 - shape$
$Ex\_ext = E\_0 * f(t) * g(x) * h(y) * cos(\pi * x/xwidth - omega * (t - t0) + phase - 0.5 * alpha * (t - t0 - t\_res) * *2)$
$Ey\_ext = 0.0$
```

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_ramp

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

t_off

Description: Ramp down time. Driver ramps down from t0+t_ramp to t0+t_ramp+t_off.
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: Half of a wavelength of shape function in x.
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

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 2 collision operators, Pitch Angle Collision Operator and Rosenbluth Collision Operator.

These parameters apply to both collision operators:

name

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

collision_vceil

Description: Upper velocity limit used to compute collision rate. The collision rate for velocities >= collision_vceil is modulated by:
$1.0 - sin^2(\pi/2 * ((v - collision\_vceil)/(vmax - collision\_vceil)))$
Data Type: Real
Default: None. This input is required.

collision_vfloor

Description: Lower velocity limit used to compute collision rate. All velocities below collision_vfloor are set to collision_vfloor.
Data Type: Real
Default: None. This input is required.

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. If "local vthermal" or "global vthermal" are selected then there must not be an entry for collision_vthermal 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.

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

This parameter applies only to "Pitch Angle Collision Operator":
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.

These parameters apply only to "Rosenbluth Collision Operator":
collision_alpha

Description: This is the same as 1/vthermal.
Data Type: Real
Default: None. This input is required.

Notes: This is redundant. It is the same as 1/vthermal.

collision_back_reaction

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

collision_massR

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

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 the method for the Poisson solve one would use the following syntax:
poisson.solver_method = "superlu direct"

solver_method

Description: There are 3 different solvers, one direct and 2 iterative. The iterative algorithms are older and are still in the code for reproducibility of older runs. We recommend using the direct solver which is significantly faster.
Data Type: String
Default: "superlu direct"
Notes: Must be "original iterative", "overture best iterative", or "superlu direct".

solver_tolerance

Description: This parameter controls the solver tolerance which only applies to the iterative solvers.
Data Type: Real
Default: If using "original iterative" default is 1.0e-6, if using "overture best iterative" default is 1.0e-12.

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.

Data Type: String

Default: Empty string (no tracking particles specified).

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

```
if (t0 <= t < t0+t_ramp)
    $f(t) = 0.5 + 0.5 * tanh(4.0 * (2.0 * (t - t0)/t\_ramp - 1.0))$
else if (t0+t_ramp <= t < t0+t_ramp+t_off)
    $f(t) = 0.5 - 0.5 * tanh(4.0 * (2.0 * (t - t0 - t\_off)/t\_ramp - 1.0))$
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_ramp

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

t_off

Description: Ramp down time. Driver ramps down from t0+t_ramp to t0+t_ramp+t_off.
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 speed of light for the electrodynamic algorithm one would use the following syntax:
maxwell.light_speed = 10.0

light_speed

Description: Light speed.
Data Type: Real
Default: 1.0

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

num_em_ics

Description: Indicates the number of electromagnetic field initial conditions.
Data Type: Integer
Default: None. This input is required but 2 is the required input.
Notes: This must be 2. Not sure why we're making you put this in your deck since you must initialize both E and B.

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.
Data Type: String
Default: Empty string (no tracking particles specified).

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 is only one current driver, the Shaped Ramped Cosine Current 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 Shaped Ramped Cosine Current Driver is a delta function in x. It is applied only at x=x0. Its functional form is largely the same as for the Electric Field Driver:

```
if (t0 <= t < t0+t_ramp)
    $f(t) = 0.5 + 0.5 * tanh(4.0 * (2.0 * (t - t0)/t\_ramp - 1.0))$
else if (t0+t_ramp <= t < t0+t_ramp+t_off)
    $f(t) = 0.5 - 0.5 * tanh(4.0 * (2.0 * (t - t0 - t\_off)/t\_ramp - 1.0))$
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”, ”y”, ”z”, ”X”, ”Y”, or ”Z”.

t0

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

t_ramp

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

t_off

Description: Ramp down time. Driver ramps down from t0+t_ramp to t0+t_ramp+t_off.
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

width

Description: Half of a wavelength of modulation in y.
Data Type: Real

Default: None. This input is required.

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 Field Initial Conditions. 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 will be specified in the namespace maxwell.em_ic.n where n is one based. Therefore to specify the xamp of the 2nd field initializer of the electrodynamic algorithm one would use the following syntax:
maxwell.em_ic.2.xamp = 0.0001

At the present time there is only one 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:
$Ex = xamp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$
$Ey = yamp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$
$Ey = zamp * cos(x\_wave\_number * x + y\_wave\_number * y + phase)$

name

Description: Name of field initializer.
Data Type: String
Default: None. This input is required.
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.
Notes: Must be "e", "b", "E", or "B".

xamp

Description: Specifies the x amplitude of the field being initialized.
Data Type: Real
Default: None. This input is required.

yamp

Description: Specifies the y amplitude of the field being initialized.
Data Type: Real

Default: None. This input is required.

zamp

Description: Specifies the z amplitude of the field being initialized.
Data Type: Real
Default: None. This input is required.

x_wave_number

Description: Wave number of standing wave in x.
Data Type: Real
Default: None. This input is required.

y_wave_number

Description: Wave number of standing wave in y.
Data Type: Real
Default: None. This input is required.

phase

Description: Phase of standing wave.
Data Type: Real
Default: None. This input is required.

Parameters Describing Electrodynamic (Maxwell) Algorithm Drift Velocity Initial Conditions. It is necessary to specify initial condition on the transverse drift velocity of each kinetic 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. Therefore to specify the amp of the 2nd kinetic species transverse drift velocity initializer of the electrodynamic algorithm one would use the following syntax:
maxwell.vel_ic.2.amp = 0.0001

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.

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

x_wave_number

Description: Wave number of standing wave in x.
Data Type: Real
Default: None. This input is required.

y_wave_number

Description: Wave number of standing wave in y.
Data Type: Real
Default: None. This input is required.

phase

Description: Phase of standing wave.
Data Type: Real
Default: None. This input is required.

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. If the post-processing tool is to be run, this needs to be consistent with the show_file_name input. See the comments for that input.

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

Parameters Describing Kinetic Species

number_of_processors

Description: Each kinetic species is 4D and distributed over a set of processors. If you wish to control the number of processors that a kinetic species is distributed over you would specify it here. Otherwise Overture will determine the distribution based on the work load of each species. It is best to let Overture determine the distribution. This has not been tested much if at all.
Data Type: Integer
Default: None. Overture determines.

vflowinitx

Description: Specify an initial flow velocity in the vx direction.
Data Type: Real
Default: 0.0

vflowinity

Description: Specify an initial flow velocity in the vy direction.
Data Type: Real
Default: 0.0

Parameters Describing Electrostatic (Poisson) Algorithm
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 intent is 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. Therefore, you should set this to 1 for now.
Data Type: Integer
Default: 1

Parameters Describing Electrodynamic (Maxwell) Algorithm
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. In practice this has not proven to be needed and has not been tested much if at all. Therefore, you should set this to 1.
Data Type: Integer
Default: 1

avWeak

Description: Weak artificial dissipation term in evaluation of RHS of Maxwell's equations.
Data Type: Real
Default: 0.1

avStrong

Description: Strong artificial dissipation term in evaluation of RHS of Maxwell's equations.
Data Type: Real
Default: 0.1

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, show_file_name, 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 1D 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
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 $4 * Ns + 2$ 2D fields generated 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
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 $5 * Ns + 6$ 2D fields generated for a Vlasov-Maxwell system.

1D Time Histories. The 1D 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) maximal value of $\sqrt{(E_x^2 + E_y^2)}$
(2) integral of $\sqrt{(E_x^2 + E_y^2)}$
(3) maximal value of $abs(E_x)$
(4) maximal value of $abs(E_y)$
(5) integral of $0.5 * (E_x^2 + E_y^2)$
For each probe:
(6) E_x at the probe
(7) E_y at the probe
For each tracking particle:
(8) particle x position
(9) particle y position
(10) particle x velocity
(11) particle y velocity
For each kinetic species:
(12) integrated kinetic energy
(13) kinetic energy flux through x low physical boundary
(14) kinetic energy flux through x high physical boundary
(15) kinetic energy flux through y low physical boundary
(16) kinetic energy flux through y high physical boundary

Therefore, if there are Ns species, Np probes, and Nt tracking particles there will be $5 + 2 * Np + 4 * Nt + 5 * Ns$ 1D time histories generated for a Vlasov-Poisson system.

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

(1) maximal value of $\sqrt{(Ex^2 + Ey^2 + Ez^2)}$
(2) integral of $\sqrt{(Ex^2 + Ey^2 + Ez^2)}$
(3) maximal value of abs(Ex)
(4) maximal value of abs(Ey)
(5) maximal value of abs(Ez)
(6) integral of $0.5 * (Ex^2 + Ey^2 + Ez^2)$
(7) maximal value of $\sqrt{(Bx^2 + By^2 + Bz^2)}$
(8) integral of $\sqrt{(Bx^2 + By^2 + Bz^2)}$
(9) maximal value of abs(Bx)
(10) maximal value of abs(By)
(11) maximual value of abs(Bz)
(12) integral of $0.5 * (Bx^2 + By^2 + Bz^2)$
For each probe:
(13) Ex at the probe
(14) Ey at the probe
(15) Ez at the probe

(16) Bx at the probe
(17) By at the probe
(18) Bz at the probe
(19) for each kinetic species: vz at the probe
For each tracking particle:
(20) particle x position
(21) particle y position
(22) particle x velocity
(23) particle y velocity
For each kinetic species:
(24) integrated kinetic energy
(25) kinetic energy flux through xlo physical boundary
(26) kinetic energy flux through xhi physical boundary
(27) kinetic energy flux through ylo physical boundary
(28) kinetic energy flux through yhi physical boundary

Therefore, if there are Ns species, Np probes, and Nt tracking particles there will be $12 + Np * (6 + Ns) + 4 * Nt + 5 * Ns$ 1D 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.