FP: Fokker-Planck Transport in Flares

SUMMARY

During solar flares numerous particles (electrons, protons, and ions) are accelerated to high energies at the tops of magnetic flux loops in the solar corona. They travel down the loops interacting with and heating the ambient plasma over the course of their transport. This heating drives upflows of dense material into the magnetic loops, resulting in the intense bursts of radiation that characterize flares. The interaction of these high energy particles with the ambient solar atmosphere is well-represented by the Fokker-Planck kinetic theory, which includes Coulomb force interactions coupled with magnetic and external electric field forces. Mathematically, the Fokker-Planck theory is represented by a nonlinear partial differential equation that must be solved numerically. The particular solution of the Fokker-Planck equation depends on the atmospheric conditions within the magnetic flux loop (e.g., temperature, density, ionization fractions, magnetic flux density) as well as the nonthermal distribution function of the flare-accelerated particles. Thus, particular solutions must be obtained for each flare that is to be analyzed. We have developed the FP package to efficiently perform this computational task for arbitrary loop conditions and nonthermal particle distributions.

FP is a Fortran package developed with MPI for parallel processing and includes a Python and IDL wrapper to facilitate ease of use. It is designed for use by solar and stellar physics researchers. It has been used as input to the radiation hydrodynamic model, RADYN, to model the solar atmosphere's response to the heating and nonthermal ionization produced by impacting flare-accelerated particles. It has also been used as a forward-modeling tool to invert X-rays observed during flares by NASA's Ramaty High Energy Solar Spectroscopic Imager (RHESSI) to determine the nonthermal particle distribution function leaving the acceleration site.

BUILDING FP

Prerequisites

1. FP requires a Fortran 95+ compiler with MPI. It has been tested with GFortran and Intel Fortran. GFortran is a free compiler and is recommended for users without a pre-existing Fortran installation. Information about GFortran and binary packages are available here: https://gcc.gnu.org/fortran/
2. FP is written to take advantage of parallel computing using MPI, so MPI is required and must be built with a compiler compatible with the Fortran compiler. FP has been tested with MPICH, which is available here: https://www.mpich.org/
3. FP uses the binary file format, HDF5, to store its output. Documentation and binary packages are available here: https://www.hdfgroup.org/solutions/hdf5/
4. FP has been tested using Mac OS and Linux distributions (Fedora and Cent OS). It has not been tested with Windows.

Compiling FP

1. The source code for FP is in the "code" subdirectory of the FP package. In that subdirectory there is the "Makefile". Edit this file to:
    a. Set the HDF5_LIB_PATH and HDF5_INCLUDE_PATH variables to point to the library and include paths of the HDF5 installation. Specifically, the directory pointed to by HDF5_LIB_PATH should contain the file, libhdf5_fortran.a and the directory pointed to by HDF5_INCLUDE_PATH should contain the file, hdf5.mod.
    b. Set "F" to be equal to the path to the MPI Fortran compiler executable. If the compiler (e.g., mpif90) is already in the system path, then simply setting "F = mpif90" will work.
    c. Set "FOPT" to be equal to optimization options used by the compiler. GFortran and Intel Fortran options are given as examples.
2. Build FP, by entering "make" in the command line in the code subdirectory.
3. This should produce a binary executable, "fp", in the code directory.

RUNNING FP

FP can be run as a standalone executable but is most easily run using the included IDL or Python 3 interfaces. On Linux systems it is necessary to ensure the that HDF5_LIB_PATH is included in the LD_LIBRARY_PATH. In the bash shell, for example, this can be accomplished by running: "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HDF5_LIB_PATH" where the HDF5_LIB_PATH variable is the same that was set in the Makefile in the "Compiling FP" step.

IDL Interface:

1. Requires IDL version $\geq$ 6.2.
2. Ensure the FP/idl subdirectory is in the IDL path. This can be done by adding the path to the IDL system variable, !PATH.
3. The fp.pro IDL procedure sets up a parameter file input to FP, executes FP, and reads in the output, storing it into an IDL structure.
4. FP is called from IDL using the following syntax:

   ```
   fp, fpout, [set keyword options here]
   ```

   This runs the Fortran code, storing the output into an IDL structure called "fpout". The keyword options control which forces are included, grid sizes, and several other parameters. A full list is included at the end of this document under the heading "Input Parameters."
5. The output structure includes the distribution function obtained from FP and several other output variables. Those are listed at the end of this document under the heading "Output Variables"

Python Interface:

1. Requires Python version $\geq$ 3.4.
2. FP relies on several python packages which must be installed prior to using FP. These are: numpy (numpy.org), scipy (scipy.org), h5py (h5py.org), and matplotlib (matplotlib.org).
3. Ensure that the FP/python interface is in the python search path. This path can be added using the sys.append.path() procedure in the sys module.
4. FP is called from python using the following syntax:

   ```
   import fp
   fpout = fp.solver( [set keyword options here] )
   ```

   This runs the Fortran code, FP, storing the output into a python class called "fpout". The keyword options control which forces are included, grid sizes, and several other parameters. A full list is included below under the heading "Input Parameters."
5. The output class includes the distribution function obtained from FP and several other output variables. Those are listed below under the heading "Output Structure"

Input Parameters

These parameters can be set to control FP. The variable types and default values are listed in parentheses.

nE (nEn in IDL) (Integer: 100): The number of grid cells in the energy grid.
nmu (Integer: 60): The number of grid cells in the pitch-angle grid. Must be greater than 5 unless oneD is set to True.
atmfile (String: 'atm.dat'): File name of the binary file storing the loop stratification.
inc_relativity (Boolean: True): Should the relativistic (if True) or classical (if False) expressions relating energy, speed, and momentum be used.
inc_cc (Boolean: True): If True, the Coulomb collision force will be included.
inc_synchro (Boolean: True): If True, the synchrotron reaction force will be included.
inc_magmirror (Boolean: False): If True, the magnetic mirroring force will be included.
inc_rc (Boolean: True): If True, the return current force will be included.
oneD (Boolean: False): If True, impose a fully 1D (not 1.5D) geometry. This ensures that nmu = 1 and patype = 0.
reflecttop (Boolean: False): If true, particles moving out of the loop top will be reflected back down into the loop.
reflectbottom (Boolean: False): If true, particles moving through the bottom of the loop will be reflected back up into the loop.
maxiter (Integer: 100): Maximum number of iterations.
tolres (double: 1e-3): Requested tolerance of the L2 residual of the FP matrix solver.
toldiff (double: 1e-4): Requested tolerance of the relative difference between successive iterations.
implicit_theta (double: 1.0): Damping parameter between successive iterations. In performing an iteration, $f^{(n+1)}$ = (implicit_theta)* $f^{(n+1)}$ + (1 – implicit_theta)* f^{n}. This can be used to damp out oscillations that develop between iterations.
mbeam (double: electron mass): Mass of the beam particle in eV.
Zbeam (double: -1.0): Charge of the beam particle in units of elementary charge.
Ecut (double: 20.0): Cutoff energy of the injected power law distribution in keV.
dlt (double: 5.0): Power law index of the injected distribution.
Eflux (double: 1e11): Energy flux of the power injected power law distribution in erg cm^{-2} s^{-1}.
patype (integer: 2): Selects the type of injected pitch angle distribution.
    0: All injected particles have pitch angle, theta = 0 (i.e., fully beamed)
    1: Isotropic in the forward hemisphere.
    2: Gaussian pitch-angle distribution.
pasigma (double: 0.05): Sigma width parameter in radians for the pitch angle distribution. Only used when patype = 2.
resist_fact (double: 1.0): Scale factor to apply to the Spitzer resistivity.
Emin (double: 1.0): Low energy edge of the energy grid in keV.
Emax (double: 2e3 * Ecut): High energy edge of the energy grid in keV.

restart (Boolean: False): In IDL if restart = 1 then restart FP using the solution stored in fpout as the new starting solution. In python, to restart from a previous solution, set “restart = fpout” where fpout is the python class returned by a previous iteration of FP.

outfile (String: ‘’): Set this to a file name to hold the HDF5 output produced by FP. If unset then delete the HDF5 file after FP is run.

nthreads (Integer: maximum available): Set this to the number of MPI processes to launch.

Output Structure

This structure in IDL or class in python contains the output obtained after running FP. The data type and array sizes are listed in parentheses. Since python uses row-major and IDL uses column-major indexing multi-dimensional arrays, the order of the indices is reversed between IDL and python. Here we quote the order for python.

nE (nEn in IDL) (integer): The number of grid cells in the energy grid.
nmu (integer): The number of grid cells in the pitch-angle grid.
nz (integer): The number of grid cells in the position grid.
f (double); (nz,nmu,nE): The distribution function obtained by FP (particles cm^{-3} keV^{-1} sr^{-1}).
esvol (double); (nmu, nE): The energy space volume element (keV sr). That is dE * dOmega where dE is the energy differential element and dOmega is the solid angle differential element.
heatrate (double); (nz): The heating rate of the injected particles on the ambient loop as a function of position (erg cm^{-3} s^{-1}).
momrate (double); (nz): The momentum deposition rate of the injected particles on the ambient loop as a function of position (g cm^{-2} s^{-2}).
atmfile (string): File name of the input binary file storing the loop stratification.
atm: In IDL a substructure or in python a dictionary containing the loop stratification of the input loop.
    nIon (integer): The number of ion species in the ambient plasma.
    nNeutral (integer): The number of neutral species in the ambient plasma.
    zin (double); (nz): The loop z-axis measured from the photosphere (cm).
    tg (double); (nz): The gas temperature (K).
    bfield (double); (nz): The magnetic flux density (G).
    dni (double); (nIon, nz): The number densities of the ion species composing the ambient plasma (cm^{-3}).
    dnn (double); (nNeutral, nz): The number densities of the neutral species composing the ambient plasma (cm^{-3}).
    mion (double); (nIon): masses of the ion species (eV).
    Zion (double); (nIon): charges of the ion species (elementary charge).
    Zn (double); (nNeutral): Atomic number of the neutral species
    Enion (double); (nNeutral): Ionization energy of the neutral species (eV).

inputparams: In IDL a substructure or in a python a dictionary storing the input parameters (see above) specified to start the run.
E (double); (nE+1): The energy grid cell boundaries (keV).
Em (double); (nE): The energy grid cell centers (keV).
gma (double); (nE+1): The relativistic Lorentz factor, γ, at the energy grid cell boundaries. If inc_relativity = False, then gma = 1.
gmam (double); (nE): The relativistic Lorentz factor, γ, at energy cell centers. If inc_relativity = False, then gmam = 1.

bta (double); (nE+1): Speed / speed of light at the energy grid cell boundaries.
btam (double); (nE): Speed / speed of light at energy grid cell centers.
theta (double); (nmu+1): The pitch angle grid cell boundaries (radians).
thetam (double); (nmu): pitch angle grid cell centers (radians).
mum (double); (nmu): cos(thetam).
z (double); (nz+1): The position grid cell boundaries measured from the looptop (cm).
zm (double); (nz): The position grid cell centers (cm).
eflux (double); (nz): The energy flux in the z direction as a function of position (erg cm^{-2} s^{-1})
eflux = $\int f \boldsymbol{v}\, E\, dE\, d\Omega$ = sum(f * Em * btam* mum * esvol)
nflux (double); (nz): The number flux in the z direction as a function of position (particles cm^{-2} s^{-1}).
nflux = $\int f \boldsymbol{v}\, dE\, d\Omega$ = sum(f * btam* mum * esvol)
flux (double); (nz, nE): The number flux distribution in the z direction as a function of energy and position (particles cm^{-2} s^{-1} keV^{-1}).
flux = $\int f \boldsymbol{v}\, d\Omega$ = sum(f * btam* mum * dOmega)
flx (double); (nz, nE): The scalar number flux as a function of energy and position (particles cm^{-2} s^{-1} keV^{-1}).
flx = $\int f |\boldsymbol{v}|\, d\Omega =$ sum(f * btam* dOmega)

The following are available only when electrons are selected as the beam particle:
Eph (double); (nE): Energy grid for X-ray bremsstrahlung photons (keV).
brem (double); (nE,nz): The electron/ion X-ray bremsstrahlung as a function of position and energy (ph cm^{-3} s^{-1} keV^{-1}).
totbrem (double); (nE): brem integrated over z (ph cm^{-2} s^{-1} keV^{-1}).

Examples

The following examples are bundled with FP in the “FP/examples” subdirectory. Each example produces a plot, which can be used to ensure FP is working correctly.

1. compare_fp_e78
   IDL: “@compare_fp_e78”
   Python: “import compare_fp_e78”

   This example runs FP to simulate injecting electrons with a power-law distribution into a loop with half-length of 13 Mm and apex coronal temperature of 3.4 MK. First, FP is run in a mode including only Coulomb collisions to compare with the model from Emslie 1978 (ApJ, 224, 241). Then FP is run including the return current to demonstrate its effect. This example produces a two-panel plot. The top panel shows the loop temperature (black line) and electron (red line) and hydrogen (blue line) density stratification. The bottom panel shows the heating rate predicted by FP for the Coulomb collisions only case (black line), the return current case (blue line), and prediction using the model of Emslie 1978 (red line). The plot produced by the python version of this script is shown in Figure 1.

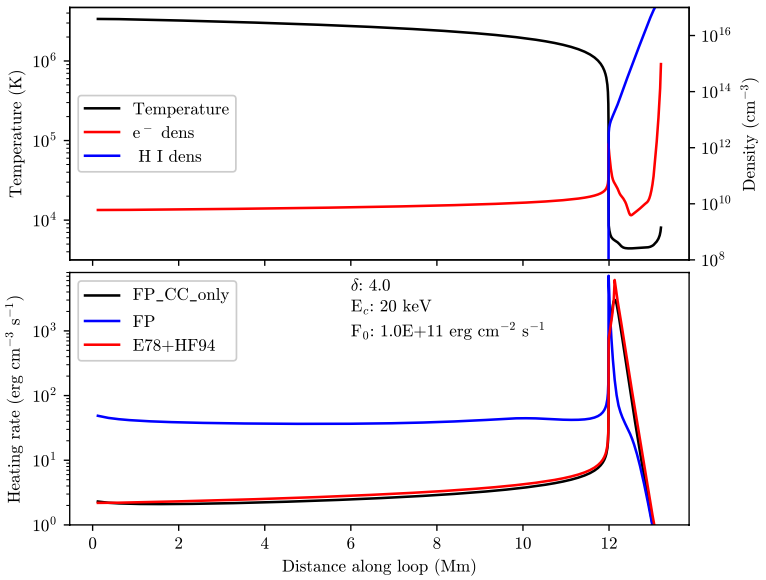

Figure 1: Plot resulting from the example "compare_fp_e78"

2. compare_fp_h12
   IDL: “@compare_fp_h12”
   Python: “import compare_fp_h12”

   This example runs FP in three modes. In each a power-law distribution of electrons is injected in the same loop as presented in the previous example. In the first mode, FP is run including only the return current force and ignoring pitch-angle effects (i.e., 1D not

1.5D; labeled FP_RC_only_1D). The second runs FP also including only the return current force but this time including pitch-angle effects (labeled FP_RC_only). Finally, FP is run with all its forces included (labeled FP). This example produces a two-panel plot. In the top panel the heating rates predicted from each case is compared to the RCCTTM model from Holman 2012 (ApJ, 745, 52). In the bottom panel, the electron flux distribution predicted by each of these models at a position of 7Mm is compared to the prediction from RCCTTM and the injected flux (z=0; dashed black line). The plot produced by the python version of this example is shown in Figure 2.

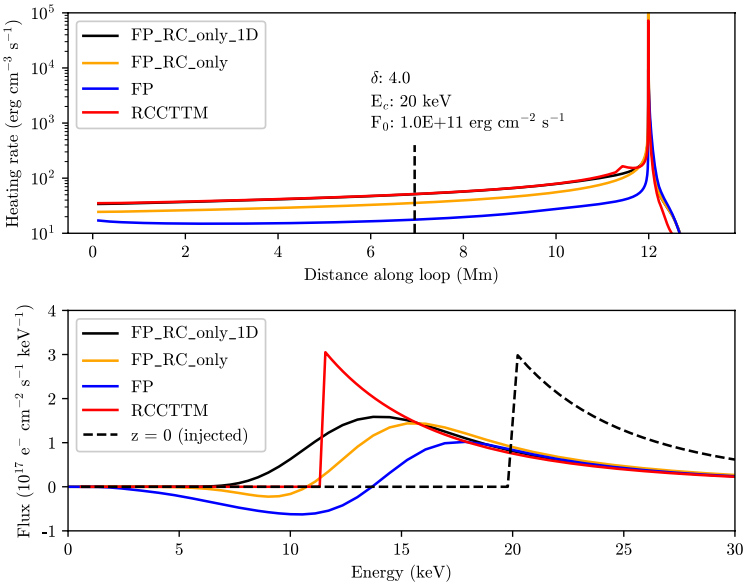

Figure 2: Plot produced by the "compare_fp_h12" example.

3. compare_fp_t86
   IDL: “@compare_fp_t86”
   Python: “import compare_fp_t86”

   This example models the injection of power-law distributions of protons into the same loop as used in the first example. It is run in two cases. In the first, a proton distribution with low-energy cutoff of 100 keV is used, and in the second the low-energy cutoff is 1000 keV. This example compares differences between cold- and warm-target collisions as defined by Tamres et al. 1986 (ApJ,) A two panel plot is produced. The top panel shows the heating rates for the 100 keV case as predicted by FP in a mode including only Coulomb collisions (FP_CC_only; black line), FP (blue line), the cold target approximation (E78; red line), and the warm target (orange line). The bottom panel shows the same

quantities for the 1000 keV cutoff energy case. The plot produced by the python version of this example is shown in Figure 3.

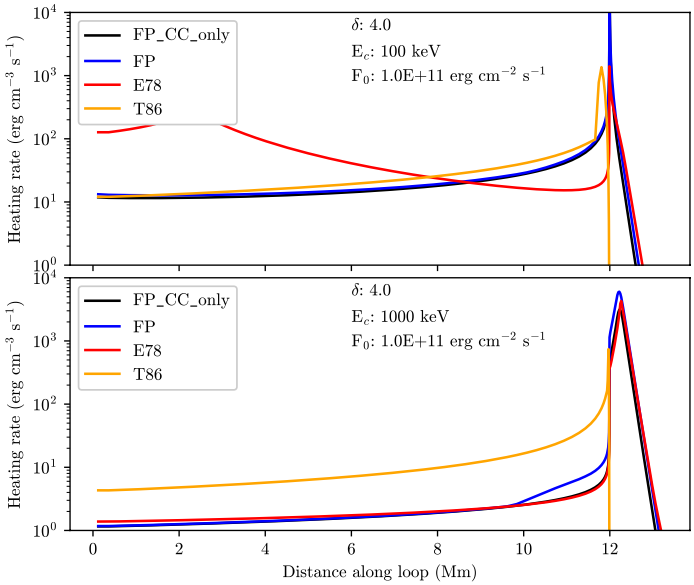

Figure 3: Plot produced by the "compare_fp_t86" example.