HYbrid and MHD simulation code (HYM)

The nonlinear 3-D simulation code (HYM) has been originally developed at PPPL to study the macroscopic stability properties of FRCs. In the HYM code, three different physical models have been implemented: (a) a 3-D nonlinear resistive MHD model; (b) a 3-D nonlinear hybrid scheme with fluid electrons and particle ions; and (c) a 3-D nonlinear hybrid MHD/particle model where a fluid description is used to represent the thermal background plasma, and a kinetic (particle) description is used for the low-density energetic beam ions. The nonlinear delta-f method has been implemented in order to reduce numerical noise and optimize the particle simulations. The modeling capabilities of the HYM code include the options of a two-fluid (Hall-MHD) description of the thermal plasma, and the effects of finite electron pressure.

The HYM code employs the delta-f particle simulation method and a full-ion-orbit description in toroidal geometry. The use of the delta-f method has been found to be crucial for understanding the underlying physics of macroscopic instabilities in the kinetic regime. In this method, the equilibrium ion distribution function f_0 is assumed to be known analytically, and the equation for the perturbed distribution function $\delta f = f - f_0$ is integrated along the particle trajectories. Each simulation particle is assigned a weight $w = \delta f / f$, which evolved in time using the equation

$$f_0 \frac{dw}{dt} = -(1-w)\frac{df_0}{dt}. \qquad (1)$$

The particle weights are used to calculate the perturbed ion density and current density according to $\delta n(\mathbf{x},t) = \sum_m w_m S(\mathbf{x}_m - \mathbf{x})$, and $\boldsymbol{\delta}\mathbf{J}(\mathbf{x},t) = \sum_m w_m \mathbf{v}_m S(\mathbf{x}_m - \mathbf{x})$, where *m* is the simulation particle index, and *S(***x***)* is a shape function. This method simplifies linearization of the Vlasov equation, and it allows for a detailed study of the linear phase of instability. The delta-f method, since it is fully nonlinear, is also essential for the study of weakly nonlinear effects, such as the saturation of instabilities at low amplitudes. On the other hand, the delta-f method is found to be inaccurate in strongly nonlinear regimes. Therefore, the numerical scheme in the HYM code has been extended to allow for a dynamical switch from the delta-f method to the regular PIC method, when the perturbation amplitude becomes sufficiently large.

The initial equilibrium used in the HYM code is calculated using a Grad-Shafranov solver. The equilibrium solver allows the computation of MHD equilibria including the effects of toroidal flows. In addition, a kinetic version of the Grad-Shafranov solver calculates equilibria with a non-Maxwellian and anisotropic fast ion distribution function.

CODE DESCRIPTION

The HYM code is a Fortran 90 code, which is written using a pre-processor language MPPL. The HYM code was written to run on distributed memory computers at NERSC. An MPI (Message Passing Interface) version of the HYM code has been developed using 1D, 2D or 3D domain decomposition of the field grid depending on the problem size. The code uses general orthogonal coordinates. For FRC and tokamak applications, fields are advanced on cylindrical grid, and particles are pushed on a Cartesian grid.

The Grad-Shafranov equilibrium code, FRCIN (Fortran 90), has been developed for FRC and spheromak merging studies. The MPI version of this code allows the calculation of FRC kinetic equilibria for a five-dimensional particle phase space. The calculation of the kinetic equilibrium requires typically less than 5 minutes of wall-clock time, using 16 processors. The Grad-Shafranov parallel equilibrium code, TKIN has been developed to calculate self-consistent equilibria for NSTX(-U) and DIII-D simulations including the energetic beam ion component [Phys. Plasmas 10, 3240, 2003].

Most of the post-processing, data analysis, and visualization are performed using the IDL-based code, GUI1.

Equilibrium codes
There are two main codes: frcin.p (FRC) and tkin.p (tokamak), both written using the MPPL pre-processor which will generate F90 codes, frcin.f and tkin.f respectively. They are used to generate initial conditions and include Grad-Shafranov (GS) solver. ‘frcin.p’ is a serial code, it can be run on a single processor, and it can be used to generate FRC-like or spheromak-like initial configurations. ‘tkin.p’ and ‘tkin_read.p’ are MPI codes.

The equilibrium code requires two input files: frc.i (tk.i) and hmin.i. The frc.i (tk.i) file specifies equilibrium parameters for GS solver, ie FRC/spheromak equilibrium. The hmin.i file contains simulations parameters which specify size of simulation region, boundary conditions, and the perturbation amplitude (rest of hmin.i parameters are related to particle simulations, and should be ignored in MHD version).

MPPL
MPPL is Fortran pre-processor developed long time ago at LLNL as part of their larger code-developing system. MPPL has a powerful macro capability, including both built-in and user-defined macros with arguments, which is the main reason for its use in HYM. The macros are used extensively in the HYM for switching between different simulation models, defining global parameters, allowing easy-to-read code, and etc. NOTE, all user-defined MPPL macros in HYM are in capital letters to distinguish them from Fortran functions and variables.

Examples:
1.
```
    define NID    257          ! Number of data i positions.
    define NJD    127          ! Number of data j positions.
    define NKD     16          ! Number of data k positions
```
In generated F90 code, these will be substituted as numbers everywhere in the hybm.f.

2.

```
    define LMPI      T        # =T use MPI calls.
    define LMHD     T        # mhd (no particles) option; needs nbs=1
```

These are ‘logical’ macros, therefore having values of T (true) or F(false) , and used to switch between parallel (MPI) – serial code versions, and MHD – kinetic numerical models, respectively.

3.

ifelse($,$,[ ],[ ]) is a built-in macro, which is used to switch between different parts of the code. For example,

```
ifelse(LMPI,T,[
      parallel code
],[
      serial code
])
```

allows to generate different F90 code (ie parallel or serial), depending on value of LMPI macro.

4.

V macro (is used to expand vector in HYM):

```
      define(V,[v1[]$1,v2[]$1,v3[]$1])
```

Example: V(a2) will be expanded in hybm.f into

```
    v1a2,v2a2,v3a2
```

ie three components of a2 (vector potential at time level 2).

5.

DIM_s and DIM_v macros are used to dimension vectors and scalars in subroutines.

6.

ADD_vvv, SET_vv and etc macros are used for simple vector operations. Example:

```
      SUB_vvv( V(tmp), V(tmpa), V(tmpb) )    will be expanded into

      v1tmp = v1tmpa - v1tmpb
      v2tmp = v2tmpa - v2tmpb
      v3tmp = v3tmpa - v3tmpb
```

7.

SDO […] SENDDO macros expand into

```
      do  k= 1,nk
        do  j= 1,nj
          do  i= 1,ni
            […]
          enddo i
        enddo j
      enddo k
```

All macro definitions can be found in HYM, and most include comments or are self-explanatory.

Normalization
All physics parameters are normalized in a way that resulting normalized equations don't have numerical coefficients in them. Note that most MHD codes use MHD time scale to normalize time, but since HYM has both hybrid and MHD versions in the same source code, I had to use particle-code normalization for all.

Code normalized units are as follows:
Length is normalized to: V_A/omega_ci = ion skin depth
Magnetic field normalized to value of external field: B_0
Time: 1/omega_ci
Pressure: B_0/4*pi
Everything else can be expressed in these units, ie
current_unit= B_unit/Length_unit and etc.

Viscosity and resistivity are a bit more complicated, you'll have to divide code values by device radius in code units (ie Rc=28.2) in order to get 1/Re and 1/S, where Re and S are Reynolds and Lunquist numbers respectively.

For conventional MHD-like normalization, normalized viscosity and resistivity equal to inverse of Reynolds number (mu= 1/Re) and inverse Lundquist number (eta=1/S), respectively. However, note that the HYM code uses different normalizations:
ie values of 'vis' and 'res' in 'hybm.i' file are normalized based on L_0 (ion skin depth) length scale instead on global R_c scale. For example, for viscosity, the code normalization is: L_0*V_A*m_i*n_0. Again, this allows to avoid numerical coefficients in the code equations, but one have to divide the code values of 'res' and 'vis' by the normalized R_c in order to get the 'MHD values'.

HYM naming convention (variables)
There are rules for variables and subroutines names used in the HYM code. The purpose for naming convention is to make it easier to write, read, debug and understand the code.

The HYM is 3D code, and three dimensions are labeled with numbers 1,2,3, integer indexes i, j, k, and letters q, r, s respectively. In cylindrical geometry (q, r, s) directions correspond to (z, r, phi), whereas in box geometry, they could correspond to (x,y,z) etc. For example, in cylindrical geometry, z-component of E (electric field vector) would be named: v1e and it would be 3D array of this shape: v1e(1:ni, 1:nj, 1:nk), where ni, nj, nk are number of grid points in q, r, s (ie z, r, phi) directions. Phi component of magnetic field is v3b(ni,nj,nk), and values of phi in the grid points are given by s(1:nk) array.

All scalar physical parameters (ie plasma pressure, density etc) are 3D arrays named starting with letter ‘s’, ie

spb(i,j,k) - is bulk plasma pressure at grid point (i,j,k);
srhob1(i,j,k) - is bulk plasma density at time level 1
[“Bulk” means thermal plasma, and for the MHD version it is just total plasma density etc.]

All arrays for vector fields, ie **A, B, E, V**, momentum and etc start with ‘v1’, ‘v2’ or ‘v3’. First component (ie q-component) of any vector will be named v1*, and second (r-component) – v2*, and etc. For example, 3D arrays corresponding to components of electric field will be v1e, v2e, and v3e. Description of all HYM vector fields and scalars is in module cm_fld (COMMON_FLD macro) in hybm.p file.

HYM naming convention (subroutines)
All names of subroutines acting on fields or scalar arrays follow this format:

```
swi_ss( sa, sb)
curl_vv( V(b), V(a), VBC_b)
sum_rs( rsum, sa )
```

where indexes r, s, v stand for real, scalar, and vector respectively, and indicate the type of the subroutine arguments. For example,

```
    call  swi_ss( sa, sb)
```

will switch the values of two scalar 3D arrays sa and sb;

```
    call curl_vv( V(b), V(a), VBC_b)
```

will calculate curl of vector V(a) [=v1a,v2a,v3a] and put the result in V(b) vector.

```
    call sum_rs( rsum, sa )
```

will calculate sum of all elements of 3D scalar array sa and put the result in rsum, which is defined as Fortran Real.

```
    call cros_vvv( V(c), V(a), V(b) )
```

calculates cross product of vectors V(a) and V(b) and put the result in V(c). Note that result is always a first argument ie **C**=**A** x **B**

```
   call grad_vs( V(tmpb), spb, VBC_no )
```

calculates gradient of a scalar spb (bulk pressure) and put the result in vector V(tmpb), ie V(tmpb)= grad(spb).

Grid
The HYM code uses 4-th order finite-difference scheme in all three dimensions. In general, NID, NJD and NKD are sizes of the HYM 'data' grid. There are extra 4 grid points in each dimension (ghost points) for boundary conditions (ie two ghost points at both ends of each direction). The time evolution equations (like momentum in 'stepmo') are solved only on 'data' points, and values at 'ghost' points are found after that using fix_v or fix_s subroutines. So, the output usually includes 'data' points, and not the 'ghost' points. The only exception is r-direction in cylindrical geometry. In this case, r(1)=0 and r(2)=dr are technically 'ghost' points, because they are not stepped in time, but since it is

really a part of simulation region, these two points are included in outputs. So the output in radial direction is NJD+2.

Note that SDO/SENDDO macro will make do-loop over whole range of indexes (including ‘ghost’ grid points), whereas SDO_dat/SENDDO macro will make do-loop over ‘data’ points only.

Note that I use powers of 2 for NKD values because of FFT in phi direction, which is used for diagnostics.

Boundary Conditions (cylindrical geometry, ie LCYL=T.)
For n=0 mode (ie axisymmetric) both V_r and V_phi components of any vector should be =0 at r=0. But V_z, V_x and V_y can be treated as scalars, ie continuous across r=0. The way it is done in HYM in 'fix_v' subroutine, the V_x and V_y are calculated for first few j points (j=3,5) from V_r and V_phi. Then V_x and V_y are treated as scalars using 'fix_s' subroutine. After boundary values for V_x, V_y are found, these are used to find V_r and V_phi for j=1,2. Note that uniform V_x or V_y (ie n=0) corresponds to n=1 in terms of V_r and V_phi.

Re b3.dat file: this is magnetic field or fluid velocity data. Vector components are always Vz, Vr, Vphi -in this order. In HYM all vectors stored as cylindrical components.

Using IDL post-processing codes (gui1.pro**)**
To check **initial conditions** generated by the frcin.p or tkin.p code use ‘psi.dat’ file. Run IDL code ‘gui1.pro’
choose 'Equilibrium' from menu
choose ‘Others’ if not an FRC equilibrium
choose 'load new data' from 'data' menu
load 'psi.dat' file
Click on psi1, psi4, psi5 to check out the equilibrium plots.
To check particle loading, load ‘rhoi.dat’ file first, and use ‘rhoi’ option in ‘Equilibrium menu’. Gui1.pro will also plot the distribution function in energy vs p_phi.