Radhyd Initialization
Outline

- **Initialization**

  1. Call **read_pack_array_dimensions(n_dim_data)** $\underline{(\textbf{my\_rank = 0})}$
     n_dim_data (output) integer array of array dimension data
     (a) Open file **array_dimensions.d** in directory **Data3/Initial_Data**
     (b) Call **read_array_dimensions(nread,nprint,iskipp,nx,ny,nz,nez,nnu,nnc,n_proc)**
     Reads in array dimenisons and number of processors assigned to the run.
     nread: (input) unit number to read from.
     nprint: (input) unit number to print diagnostics.
     iskipp: (input) read in echo flag.
     nx: (output) x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
     ny: (output) y-array (angular) dimension.
     nz: (output) z-array (azimuthal) dimension.
     nez: (output) Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
     nnu: Neutrino flavor array dimension. For the time being, set to 3.
     nnc: Number of nuclear species not in NSE.
     n_proc: Number of processors assigned to the run.
     (c) Close file array_dimensions.d
     (d) Open file **reset.d** in directory **Data3/Initial_Data**
     (e) Call **read_pack_init(nread, c_init_data, i_init_data, nrst, nouttmp)**
     nread: (input) unit number from which to read.
     c_init_data: (output) character array of initial data containing header.
     i_init_data: (output) integer array of initial data containing nrst.
     nrst: (output) cycle number to start simulation.
     nouttmp: (output) unit number to get restart data if nrst /= 0 (obsolete).
     (f) Close file **reset.d**
     (g) IF (nrst == 0)
        i. Open file **radhyd_keys.d** in directory **Data3/Initial_Data**
        ii. Call **read_model_dimensions(nread, nprint, iskipp, imin, imax, jmin, jmax, kmin, kmax)**
        nread: (input) unit number from which to read.
        nprint: (input) unit number to print to.
        iskip: (input) echo data read flag.
        imin: (output) inner x-array index.
        imax: (output) outer x-array index.
        jmin: (output) inner y-array index.
        jmax: (output) inner x-array index.
        kmin: (output) inner z-array index.
        kmax: (output) outer z-array index.
        iii. Close file **radhyd_keys.d**
     (h) ELSE
        i. Open file **restart.d** in directory **Data3/Restart**

ii. Call **read_model_dimensions(nread, nprint, iskipp, imin, imax, jmin, jmax, kmin, kmax)**
nread: (input) unit number from which to read.
nprint: (input) unit number to print to.
iskip: (input) echo data read flag.
imin: (output) inner x-array index.
imax: (output) outer x-array index.
jmin: (output) inner y-array index.
jmax: (output) inner x-array index.
kmin: (output) inner z-array index.
kmax: (output) outer z-array index.

iii. Close file **restart.d**

(i) END IF

(j) Check array dimensions and processor number for compatibility.

(k) Compute **i_ray_dim** and **j_ray_dim**.

(l) Pack array dimensions in integer array **n_dim_data**

2. **Broadcast array_dimensions**
3. Call **unpack_array_dimenisons(n_dim_data,nx,ny,nz,nez,nnu,nnc,n_proc, i_ray_dim, j_ray_dim)**
   Unpacks array dimensions and makes them available to each processor. (**all_ranks**)
   (a) **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
   (b) **ny**: y-array (angular) dimension.
   (c) **nz**: z-array (azimuthal) dimension
   (d) **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
   (e) **nnu**: Neutrino flavor array dimension. For the time being, set to 3.
   (f) **nnc**: Number of nuclear species not in NSE.
   (g) **n_proc**: Number of processors assigned to the run.
   (h) **i_ray_dim**: Number of radial rays per processor.
   (i) **j_ray_dim**: Number of angular rays per processor.
4. Call **load_array_module(nx,ny,nz,nez,nnu,nnc,n_proc,i_ray_dim, j_ray_dim, max_12)**
   Loads array dimensions into **array_module**. (**all_ranks**)
   (a) **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
   (b) **ny**: y-array (angular) dimension.
   (c) **nz**: z-array (azimuthal) dimension
   (d) **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
   (e) **nnu**: Neutrino flavor array dimension. For the time being, set to 3.
   (f) **nnc**: Number of nuclear species not in NSE.
   (g) **n_proc**: Number of processors assigned to the run.
   (h) **i_ray_dim**: Number of radial rays per processor.
   (i) **j_ray_dim**: Number of angular rays per processor.

(j) **max_12**: MAX(nx, ny, nz) + 12

5. Call **initialize**

   (a) Call **dimension_arrays(nx,ny,nz,nez,nnu,nnc,max_12,n_proc,i_ray_dim, j_ray_dim)**
   Dimensions and initializes the module arrays. (**all_ranks**)
   nx: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
   ny: y-array (angular) dimension.
   nz: z-array (azimuthal) dimension
   nez Neutrino energy array dimension. Must be ≥ number of active neutrino energy zones.
   nnu: Neutrino flavor array dimension. For the time being, set to 3.
   nnc: Number of nuclear species not in NSE.
   max_12: MAX(nx, ny, nz) + 12
   n_proc: Number of processors assigned to the run.
   i_ray_dim: Number of radial rays per processor.
   j_ray_dim: Number of angular rays per processor.

      i. Call **dimension_radhyd_arrays(nx,ny,nz,nez,nnu,nnc)**

         A. Call **dimension_radhyd_variable_arrays(nx,ny,nz,nez,nnu,nnc)**
         Allocates on each processor the dimensions and initializes the master radhyd arrays

         B. Call **dimension_radhyd_ray_arrays(nx,ny,nz,i_ray_dim,nez,nnu,nnc)**
         Allocates on each processor the dimensions and initializes the primary radial arrays

         C. Call **dimension_angular_ray_arrays(ny,j_ray_dim,nez,nnu,nnc)**
         Allocates on each processor the dimensions and initializes the primary angular arrays

         D. Call **dimension_prb_cntl_ray_arrays(nnu)**
         Allocates on each processor the dimensions and initializes the problem controls

         E. Call **dimension_t_cntrl_arrays(nx,nnu)** Allocates on each processor the dimensions and time step controls

      ii. Call **dimension_hydro_arrays(nx,ny,nz,nez,nnu,nnc)**

         A. Call **dimension_boundary_arrays(nx)**
         Allocates on each processor the dimensions and initializes the boundary conditions

         B. Call **dimension_convect_arrays(nx)**
         Allocates on each processor the dimensions and initializes the parameters for mixing-length convection

         C. Call **dimension_mgfld_remap_arrays(max_12,nez,nnu,nnc)**
         Allocates on each processor the dimensions and initializes the parameters for remapping the material variables

         D. Call **dimension_shock_arrays(nx,i_ray_dim)**
         Allocates on each processor the dimensions and initializes the parameters for tracking shocks

         E. Call **dimension_evh1_sweep_arrays(nx,ny,nz)**
         Allocates on each processor the dimensions and initializes the parameters for the evh1 padded variables

F. Call **dimension_evh1_zone_arrays(nx,ny,nz)**
Allocates on each processor the dimensions and initializes the parameters for global evh1 variables

G. Call **dimension_evh1_bound_arrays(nnc)**
Allocates on each processor the dimensions and initializes the hydro boundary conditions

iii. Call **dimension_mgfld_arrays(nx,nez,nnu,i_ray_dim)**

A. Call **dimension_abem_arrays(nx,nez,nnu,i_ray_dim)**
Allocates on each processor the dimensions and initializes the absorption and emission arrays

B. Call **dimension_brem_arrays(nx,nez,nnu,i_ray_dim)**
Allocates on each processor the dimensions and initializes the nucleon-nucleon neutrino bremsstrahlung arrays

C. Call **dimension_incrmnt_arrays(nx,nez,nnu,i_ray_dim)**
Allocates on each processor the dimensions and initializes the increment arrays

D. Call **dimension_mdl_cnfg_arrays(nx)**
Allocates on each processor the dimensions and initializes the mgfld model configuration arrays

E. Call **dimension_nu_dist_arrays(nx,nez,nnu,,i_ray_dim)**
Allocates on each processor the dimensions and initializes the mgfld model configuration arrays

F. Call **dimension_nu_energy_grid_arrays(nez,nnu)**
Allocates on each processor the dimensions and initializes the neutrino distribution arrays

G. Call **dimension_pair_arrays(nx,nez,nnu,i_ray_dim)**
Allocates on each processor the dimensions and initializes the electron-positron annihilation arrays

H. Call **dimension_scat_a_arrays(nx,nez,nnu,i_ray_dim)**
Allocates on each processor the dimensions and initializes the Haxton neutrino nucleus inelastic scattering arrays

I. Call **dimension_scat_e_arrays(nx,nez,nnu,i_ray_dim)**
Allocates the dimensions and initializes the neutrino electron scattering arrays on a processor

J. Call **dimension_scat_i_arrays(nx,nez,i_ray_dim)**
Allocates on each processor the dimensions and initializes the neutrino-nucleon and nucleus isoenergetic scattering arrays

K. Call **dimension_scat_n_arrays(nx,nez,nnu,i_ray_dim)**
Allocates the dimensions and initializes the neutrino nucleon elastic scattering arrays on a processor

L. Call **dimension_scat_nn_arrays(nx,nez,nnu,i_ray_dim)**
Allocates on each processor the dimensions and initializes the neutrino-nucleus inelastic scattering arrays

iv. Call **dimension_edit_arrays(nx,nez,nnu)** Allocates on each processor the dimensions and initializes the edit arrays

v. Call **dimension_eos_arrays(nx, ny, i_ray_dim, j_ray_dim, nnc)**

A. Call **dimension_eos_snc_arrays(nx, i_ray_dim, nnc)**
   Allocates on each processor the dimensions and initializes the "cube" eos machinery for the radial arrays
B. Call **dimension_eos_snc_y_arrays(ny, j_ray_dim, nnc)**
   Allocates on each processor the dimensions and initializes the "cube" eos machinery for the angular arrays
C. Call **dimension_eos_bck_arrays(nx)**
   Allocates on each processor the dimensions and initializes the BCK eos arrays
D. Call **dimension_eos_ls_arrays(nx)**
   Allocates on each processor the dimensions and initializes the LS eos arrays

vi. Call **dimension_nucbrn_arrays(nx,nnc)**
    Allocates on each processor the dimensions and initializes the nuclear network arrays

vii. Call **dimension_e_advct_arrays(nx,nez,nnu)**
     Allocates on each processor the dimensions and initializes the neutrino energy advection arrays

(b) Call **initialize_variables(nx,ny,nz,nez,nnu,nnc)**
   i. Initializes variables not initialized in the dimension variable calls
   ii. **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
   iii. **ny**: y-array (angular) dimension.
   iv. **nz**: z-array (azimuthal) dimension
   v. **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
   vi. **nnu**: Neutrino flavor array dimension. For the time being, set to 3.
   vii. **nnc**: Number of nuclear species not in NSE.
   viii. Call **initialize_global_var**
   ix. Call **initialize_cycle_arrays**
   x. Call **initialize_it_tol_arrays**
   xi. Call **initialize_bomb_arrays**
   xii. Call **initialize_rezone_arrays**

(c) Call **problem_read(c_init_data, i_init_data,c_radhyd_data,c_eos_data, c_nuc_data, i_radhyd_data,i_trans_data,i_e_advct_data,i_edit_data, i_hydro_data, i_nuc_data, i_model_data,d_radhyd_data, d_eos_data, d_trans_data, d_e_advct_data, d_edit_data,d_hydro_data, d_nuc_data, d_model_data)**
   (my_rank = 0)
   i. nread = 11
   ii. nprint = 41
   iii. Open **Data3/Initial_Data/reset.d**
   iv. Open **Data3/Run_Log/superdump.d**
   v. Call **read_pack(c_init_data, i_init_data, c_radhyd_data,c_eos_data, c_nuc_data, i_radhyd_data,i_trans_data,i_e_advct_data,i_edit_data, i_hydro_data, i_nuc_data, i_model_data, d_radhyd_data, d_eos_data, d_trans_data,d_e_advct_data, d_edit_data, d_hydro_data, d_nuc_data, d_model_data, nrst)**
      Read and broadcast initial data and run keys

A. Call **read_pack_init(nrrstp,c_init_data, i_init_data, nrst, nouttmp)**
Reads 'head" and "nrst"

B. If nrst = 0 ....
Call **radhyd_read(c_radhyd_data,i_radhyd_data,d_radhyd_data, nrst)**
.....Open Data3/Initial_Data/radhyd_keys.d
.....Call **read_radhyd_keys(nreadp,nprint,iskip,c_radhyd_data,i_radhyd_data,d_radh**
nrst)
.....Close Data3/Initial_Data/radhyd_keys.d
Call **eos_read(c_eos_data,d_eos_data,nrst)**
.....Open Data3/Initial_Data/eos_keys.d
.....Call **read_pack_eos_keys(nreadp,nprint,iskip,c_eos_data,d_eos_data,**
nrst) .....Close Data3/Initial_Data/eos_keys.d
Call **transport_read)(i_trans_data,d_trans_data, nrst)**
.....Open Data3/Initial_Data/transport_keys.d
.....Call **read_pack_transport_keys(nreadp,nprint,iskip,nez,nezp1,nnu,**
i_trans_data,d_trans_data, nrst)
.....Close Data3/Initial_Data/transport_keys.d
Call **e_advct_read(i_e_advct_data,d_e_advct_data, nrst)**
.....Open Data3/Initial_Data/e_advct_keys.d
.....Call **read_pack_e_advct_keys(nreadp,nprint,iskip,nnu,i_e_advct_data,**
d_e_advct_data, nrst)
.....Close Data3/Initial_Data/e_advct_keys.d
Call **edit_read(i_edit_data,d_edit_data, nrst)**
.....Open Data3/Initial_Data/edit_keys.d
.....Call **read_pack_edit_keys(nreadp,nprintp,iskip,nez,nnu,i_edit_data,**
d_edit_data, nrst)
.....Close Data3/Initial_Data/edit_keys.d
Call **hydro_read(i_hydro_data,d_hydro_data, nrst)**
.....Open Data3/Initial_Data/hydro_keys.d
.....Call **read_pack_hydro_keys(nreadp,nprint,iskip,nx,i_hydro_data,**
d_hydro_data, nrst)
.....Close Data3/Initial_Data/hydro_keys.d
Call **nuc_read(c_nuc_data, i_nuc_data,d_nuc_data, nrst)**
.....Open Data3/Initial_Data/nuclear_keys.d
.....Call **read_pack_nuclear_keys(nreadp,nprint,iskip,nx,nnc,c_nuc_data,**
i_nuc_data, d_nuc_data, nrst)
.....Close Data3/Initial_Data/nuclear_keys.d
Call **model_read(i_model_data,d_model_data, nrst)**
.....Open Data3/Initial_Data/initial_model.d
.....Call **read_pack_initial_model(nread, nprint, iskipp, nx, nez, nnu,**
i_model_data,
d_model_data, nrst)
.....Close Data3/Initial_Data/initial_model.d

C. ELSE ....
Open Data3/Restart/restart.d
Call **read_pack_radhyd_keys(n_restart, nprint, iskip, c_radhyd_data,**
i_radhyd_data, d_radhyd_data, nrst)

        Call **read_pack_eos_keys(n_restart, nprint, iskip, c_eos_data, d_eos_data, nrst)** Call **read_pack_transport_keys(n_restart, nprint, iskip, nez, nezp1, nnu**
        i_trans_data, d_trans_data, nrst)
        Call **read_pack_e_advct_keys(n_restart, nprint, iskip, nnu, i_e_advct_data, d_e_advct_data, nrst)**
        Call **read_pack_edit_keys(n_restart, nprint, iskip, nez, nnu, i_edit_data, d_edit_data, nrst)**
        Call **read_pack_hydro_keys(n_restart, nprint, iskip, nx, i_hydro_data, d_hydro_data, nrst)**
        Call **read_pack_nuclear_keys(n_restart, nprint, iskip, nx, nnc, c_nuc_data, i_nuc_data, d_nuc_data, nrst)**
        Call **read_pack_initial_model(n_restart, nprint, iskip, nx, nez, nnu, i_model_data, d_model_data, nrst)**
        Close Data3/Restart/restart.d

    vi. Close **Data3/Initial_Data/reset.d**
    vii. Open **Data3/Initial_Data/reset_initial.d**
    viii. Write **reset.d** used
    ix. Close **Data3/Initial_Data/reset_initial.d**

(d) Call **data_check(c_radhyd_data, i_radhyd_data)**
    Check consistency of data.

(e) Call **rezone(c_radhyd_data,i_radhyd_data,d_radhyd_data,i_model_data, d_model_data,l_rezone_data,d_rezone_data,i_nuc_data, d_nuc_data)**
    Unpack lagr, rezn, ngeomy, ngeomz, jm, and r. Set courant, xmin, and xmax. Initialize ymin, ymax, zmin, zmax.

    i. set **courant** and **lagrangian**
    ii. If rezn = 'ye'
        A. if lagrangian = true, Call **lagregrid**
        B. if lagrangian = false, Call **eulregrid**
    iii. If rezn = 'no'
        A. imax = jm - 1
        B. load nse(j) in eos_snc_module
        C. Load quantities from MGFLD to RadHyd variables.
        D. Load EVH1 boundary conditions from MGFLD.
    iv. if ndim ≥ 2, build a j grid. Set ymin and ymax,
    v. if ndim = 3, build a k grid. Set zmin and zmax,
    vi. Pack **lagrangian, imax, xmin, xmax, ymin, ymax, zmin, zmax, courant, y-coordinates, z-coordinates**

(f) Broadcast packed data to all nodes

(g) Call **Call unpack_arrays(c_init_data, i_init_data, c_radhyd_data, c_eos_data, c_nuc_data, i_radhyd_data, i_trans_data, i_e_advct_data, i_edit_data, i_hydro_data, i_nuc_data, i_model_data, l_rezone_data, d_radhyd_data, d_eos_data, d_trans_data, d_e_advct_data, d_edit_data, d_hydro_data, d_nuc_data, d_model_data, d_rezone_data)**
    (all_ranks)

i. Call **unpack_init(c_init_data)**

ii. Call **unpack_radhyd_keys(c_radhyd_data, i_radhyd_data, d_radhyd_data)**

iii. Call **unpack_radhyd_ray_keys(c_radhyd_data, i_radhyd_data, d_radhyd_data)**

iv. Call **unpack_rezone_arrays(l_rezone_data, d_rezone_data)**

v. Call **unpack_eos_keys(c_eos_data, d_eos_data)**

vi. Call **unpack_transport_keys(nez, nezp1, nnu, i_trans_data, d_trans_data)**

vii. Call **unpack_e_advct_keys(nnu, i_e_advct_data, d_e_advct_data)**

viii. Call **upack_edit_keys(i_ray_dim, nez, nnu, i_edit_data, d_edit_data)**

ix. Call **unpack_hydro_keys(nx, i_hydro_data, d_hydro_data)**

x. Call **unpack_nuclear_keys(nx, nnc, i_ray_dim, i_nuc_data, d_nuc_data)**

xi. Call **unpack_initial_model(nx, nez, nnu, i_ray_dim, i_model_data, d_model_data)**

(h) Call **problem_setup(nx, nez, nnu, i_ray_dim, nnc)**

i. IF (nrst = 0) <u>**Loop over i_ray from 1 to i_ray_dim**</u>
Call **mgfld_setup(imin, imax, i_ray, i_ray_dim, nx, nez, nnu, nnc, rho_c, t_c, ye_c, x_e, dx_c, u_c, xn_c, be_nuc_rep_c, a_nuc_rep_c, z_nuc_rep_c)**
imin: (input) inner physical x-zone index
imax: (input) outer physical x-zone index
i_ray: (input) index denoting a specific radial ray
i_ray_dim: (input) number of rays assigned to a processor
nx: (input) x-array extent
nez: (input) neutrino energy array extent
nnu: (input) neutrino flavor extent
nnc: (input) composition array extent
rho_c(:,:,:): (input) density (g/cm^3)
t_c(:,:,:): (input) temperature (MeV)
ye_c(:,:,:): (input) electron fraction
x_e(:): (input) radial coordinate (face) (cm)
dx_c(:): (input) radial coordinate thickness (cm)
u_c(:,:,:): (input) radial velocity (face) (cm/s)
xn_c(:,:,:): (input) abundance mass fractions
be_nuc_rep_c(:,:): (input) binding energies
a_nuc_rep_c(:,:): (input) nuclear mass numbers
z_nuc_rep_c(:,:): (input) nuclear charge numbers

A. Initialize radial array index boundaries (set jm, jmin, jmax, jmaxp, jnumax, jnumaxp)

B. Set quatities at inner edge of configuration

C. Transfer zone-centered independent variables to mgfld arrays

D. Transfer zone-edgeed independent variables to mgfld arrays

E. Compute Newtonian rest masses

F. Call **pblmst1** Modify problem before eos table setup (if appropriate)

G. Call **esrgnz** Load equation of state

H. Call **eqstz**

I. Call **gammaz**

J. Set m-1 and m+1 values of independent variables

K. Call **genst_rel** Compute GR quantities if irelhy = 1

L. Call **agr_cal** Time dilation factors
M. Call **gamgr_nu_cal** Put GR gammas in neutrino variables
N. Call **gamgra_nu_cal** Put updated GR gammas in neutrino variables
O. Call **agr_nu_cal** Put time dilation factors in neutrino variables
P. Call **agra_nu_cal** Put updated time dilation factors in neutrino variables
Q. Call **e_zone** Compute neutrino group energies at infinity
R. Call **enu_cal** Compute GR neutrino energy arrays
S. Call **pre_trans** Compute quantities needed for neutrino transport
T. Call **pblmst2** Modify problem given the neutrino energies (if appropriate)
U. Call **gennur** Read in and regrid Wick's neutrino interaction rates
V. Call **abemset** Compute absorption and emission opacities on table corners
W. Call **scataset** Compute Wick's scattering opacities on table corners
X. Call **scateset** Compute neutrino-electron scattering opacities on table corners
Y. Call **scatiset** Compute isoenergetic scattering opacities on table corners
Z. Call **pairset** Compute pair annihilation opacities on table corners
A. Call **bremset** Compute nucleon-nucleon bremsstrahlung opacities on table corners
B. Call **scatnset** Compute neutrino-nucleon elastic scattering opacities on table corners
C. Call **scatnnset** Compute neutrino-nucleon inelastic scattering opacities on table corners
D. Call **abemrate** Interpolate absorption and emission opacities
E. Call **sctarate** Interpolate Wick's opacities
F. Call **scterate** Interpolate neutrino-electron scattering opacities
G. Call **sctirate** Interpolate isoenergetic scattering opacities
H. Call **pairrate** Interpolate pair annihilation opacities
I. Call **bremrate** Interpolate nucleon-nucleon bremsstrahlung opacities
J. Call **sctnrate** Interpolate neutrino-nucleon elastic scattering opacities
K. Call **sctnnrate** Interpolate neutrino-nucleon inelastic scattering opacities
L. Call **nu_number** Compute the neutrino number and energy
M. Call **mfp_cal** Compute neutrino inverse mean free paths
N. Call **nu_sphere** Compute location of neutrinospheres
O. Call **diffc** Compute neutrino diffusion coefficients
P. Call **nu_stress** Compute neutrino stresses
Q. Call **eddington** Compute neutrino flux and eddington
R. Call **nu_U** Compute neutrino energy density

ii. If (nrst $\neq 0$) Call **genrst** Reinitialize problem from restart data

(i) Call **load_radhyd_ray_arrays(nx,nnu,nnc,n_ray)**
(j) Call **load_evh1_arrays**
(k) Call **time_step_check(n_ray)**
Checks that the given time step is not larger than the minimum time step given by the Courant condition.

6. Call **radhyd_to_edit(j_ray_min,j_ray_max,i_edit)**

- (a) **j_ray_min**: (input) minimum ray index
- (b) **j_ray_max**: (input) maximum ray index
- (c) **i_edit**: (input) edit flag
- (d) <u>**Loop over j_ray from j_ray_min to j_ray_max**</u>
    - i. **CALL mgfld_edit_in(is,ie,idim,j_ray_min,j_ray_max,n_ray,rho_ci,rho_c,t_c,ye_c, x_e,u_c,psi0_c,psi1_e,dtnph,time,i_editp,ncycle,xn_c,be_nuc_c,a_nuc_c,z_nuc_c, nse_c, nedc,nedmi,nedma,nedh,nedps,nedu,nedy,nedsc,nedn,nedng)**
        - A. Load edit counters into edit_module
        - B. Load state variables into mdl_cnfg_module
        - C. Load neutrino distribution functions into nu_dist_module
        - D. Load composition variables into eos_snc_module and nucbrn_module
        - E. **Call eqstz(jmin,jmaxp,j_ray)**
        - F. **Call gammaz(jmin,jmaxp,j_ray)**
        - G. **Call mgfld_edit(j_ray,i_editp,first)**
    - ii. **Call mgfld_edit_out(nedc,nedmi,nedma,nedh,nedps,nedu,nedy,nedsc, nedn,nedng)**
      Bring back updated edit counters
- (e) <u>**End Loop over j_ray**</u>

Cycling MGFLD Transport

- Call **cycle**
  Updates cycle number, opens **Data3/cycle.d** prints cycle number, closes **Data3/cycle.d**
- Initialize increment arrays
- Initialize **svel**
- <u>**Loop over j_ray from 1 to n_ray**</u>
    1. **Call store_int_radhyd_var(j_ray)**
       Stores initial values of state variables in **radhyd_variable_module** (variables end with an "i")
    2. **Call radhyd_to_evh1_x_lagr(nx,j_ray,n_ray)**
        - (a) **Call evh1_x_lagr(imin,imax,nx,j_ray,n_ray,x_e,dx_c,x_c, y_e,dy_c,y_c,z_e, dz_c,z_c,rho_c,t_c,ye_c,ei_c,u_c,v_c,w_c,nu_str_c,time,dtime)**
            - i. Set **nmin, nmax, ntot**
            - ii. Load padded arrays for Lagrangian update
            - iii. Load initial values in mgfld_remap_module
            - iv. Initialize **dt**
            - v. **Call etotal(.false.)**
            - vi. **Call sweepx(j_ray)**
                - A. **Call tgvndeye_sweep(nmin,nmax,j_ray,r0i,r0i)**
                  Update **t, p, s, gc, ge**
                - B. **Call sweepbc(nleftx,nrightx,j_ray)**
                  Fill ghost zones

C. **Call volume (ngeomx)**

D. **Call paraset(ntot,zparax,dx,xa,nmin-4,nmax+4)**

E. **Call e_compose(xa,dx,ntot,zparax)**

F. **Call ppm(ngeomx,ntot,zparax,j_ray)**

G. **Call tgvndeye_sweep(nmin,nmax,j_ray,r,r0i)**

vii. Put variables advanced by Lagrangian hydro step back in radhyd ray arrays