```
module plasma

  use constants_and_parameters
  implicit none

  ! derived data
  type, public                                          :: thermo    !  *  *
  !  define the inputs
  real ( dp ),      allocatable, dimension ( : ) :: alpha_list, j_list
  real ( dp ),      allocatable, dimension ( : ) :: LOS_list, pressure_
  integer ( lint ), allocatable, dimension ( : ) :: LOS_indices

  integer ( lint )                                      :: num_intervals

  real ( dp )                                           :: temperature_max =
  real ( dp )                                           :: pressure_max    =
  real ( dp )                                           :: density_max     =

  ! these variables are used to match computed temperature and density
  real ( dp )                                           :: density_intcpt
  real ( dp )                                           :: temperature_intcpt

  real ( dp )                                           :: temperature_index_
  real ( dp )                                           :: density_index_max

  real ( dp )                                           :: boundary_left   =
  real ( dp )                                           :: boundary_right  =
  real ( dp )                                           :: boundary_length =


  real ( dp )                                           :: map_slope, map_int

  contains

    ! functions
    procedure, public                                   :: toy
    procedure, public                                   :: g
```

```
38      procedure, public                                   :: X
39      procedure, public                                   :: Y
40
41      ! subroutines
42      procedure, public                                   :: create_mesh
43      procedure, public                                   :: populate_temperatu
44      procedure, public                                   :: populate_pressure
45      procedure, public                                   :: populate_density
46
47    end type                                                thermo    !  *  *
48
49  ! subroutines
50    private                                               :: create_mesh_sub
51    private                                               :: populate_temperatu
52    private                                               :: populate_pressure_
53    private                                               :: populate_density_s
54
55  ! functions
56    private                                               :: toy_fcn
57    private                                               :: g_fcn
58    private                                               :: X_fcn
59    private                                               :: Y_fcn
60
61    contains
62
63  !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
64
65      subroutine create_mesh_sub ( self, num_intervals, boundary_right,
66
67        class   ( thermo ), target                        :: self
68
69        real    ( dp ),   intent ( in )                   :: boundary_right, bo
70        integer ( lint ), intent ( in )                   :: num_intervals
71        real    ( dp ),   dimension ( : ), pointer        :: LOS, density, pres
72        integer ( lint ), dimension ( : ), pointer        :: indices
73        integer ( lint )                                  :: k
74        integer ( lint )                                  :: alloc_status
```

```
 75
 76       character ( 64 )                              :: err_msg_allocate
 77
 78 !     ------------------------------------------------------------------
 79
 80       ! check for valid number of intervals
 81       if ( num_intervals > 0_lint ) then
 82         self % num_intervals = num_intervals
 83       else
 84         write ( * , * ) 'Error in specifying the number of intervals'
 85         write ( * , * ) 'Value must exceed 0: requested value is ', nu
 86         stop 'FAIL: input data error'
 87       end if
 88
 89       ! check boundary values
 90       if ( boundary_right == boundary_left ) then
 91         write ( * , * ) 'Error in specifying the size of the domain in
 92         write ( * , * ) 'Right and left values are the same: ', bounda
 93         stop 'FAIL: input data error'
 94       end if
 95
 96       ! enforce Archimedean ordering
 97       if ( boundary_right > boundary_left ) then
 98         self % boundary_right = boundary_right
 99         self % boundary_left  = boundary_left
100       else
101         self % boundary_right = boundary_left
102         self % boundary_left  = boundary_right
103       end if
104
105       ! prepare the mapping from capsule space to interval number
106       self % boundary_length = self % boundary_right - self % boundary_
107       self % map_slope       = self % boundary_length / self % num_int
108       self % map_intpt       = self % boundary_left
109
110 !     ------------------------------------------------------------------
111
```

```
112 !      indices     => self % LOS_indices
113 !      LOS         => self % LOS_list
114 !      density     => self % density_list
115 !      pressure    => self % pressure_list
116 !      temperature => self % temperature_list
117
118       allocate ( self % LOS_indices ( 0 : num_intervals ), stat = allo
119       if ( alloc_status /= 0 ) then
120         write ( *, * ) 'failure to allocate integer lint array LOS_ind
121         write ( *, * ) 'allocation status variable = ', alloc_status
122         write ( *, * ) 'error message              = ', err_msg_alloca
123         stop 'memory allocation failure for Line Of Sight indices'
124       end if
125
126       allocate ( self % LOS_list ( 0 : num_intervals ), stat = alloc_s
127       if ( alloc_status /= 0 ) then
128         write ( *, * ) 'failure to allocate real dp array LOS_list ( 0
129         write ( *, * ) 'allocation status variable = ', alloc_status
130         write ( *, * ) 'error message              = ', err_msg_alloca
131         stop 'memory allocation failure for Line Of Sight mesh'
132       end if
133
134       allocate ( self % pressure_list ( 0 : num_intervals ), stat = al
135       if ( alloc_status /= 0 ) then
136         write ( *, * ) 'failure to allocate real dp array pressure_lis
137         write ( *, * ) 'oddly, the preceding allocation for LOS_list w
138         write ( *, * ) 'allocation status variable = ', alloc_status
139         write ( *, * ) 'error message              = ', err_msg_alloca
140         stop 'memory allocation failure for list of pressure values on
141       end if
142
143       allocate ( self % density_list ( 0 : num_intervals ), stat = all
144       if ( alloc_status /= 0 ) then
145         write ( *, * ) 'failure to allocate real dp array density_list
146         write ( *, * ) 'curiously, the preceding allocations for LOS_l
147         write ( *, * ) 'allocation status variable = ', alloc_status
148         write ( *, * ) 'error message              = ', err_msg_alloca
```

```
149         stop 'memory allocation failure for list of density values on r
150       end if
151
152       allocate ( self % temperature_list ( 0 : num_intervals ), stat =
153       if ( alloc_status /= 0 ) then
154         write ( *, * ) 'failure to allocate real dp array temperature_
155         write ( *, * ) 'strangely, the preceding allocations for LOS_li
156         write ( *, * ) 'allocation status variable = ', alloc_status
157         write ( *, * ) 'error message              = ', err_msg_alloca
158         stop 'memory allocation failure for list of temperature values
159       end if
160
161       allocate ( self % alpha_list ( 0 : num_intervals ), stat = alloc_
162       if ( alloc_status /= 0 ) then
163         write ( *, * ) 'failure to allocate real dp array alpha_list (
164         write ( *, * ) 'preceding allocations for LOS_list, pressure_li
165         write ( *, * ) 'allocation status variable = ', alloc_status
166         write ( *, * ) 'error message              = ', err_msg_alloca
167         stop 'memory allocation failure for list of alpha values (opac
168       end if
169
170       allocate ( self % j_list ( 0 : num_intervals ), stat = alloc_stat
171       if ( alloc_status /= 0 ) then
172         write ( *, * ) 'failure to allocate real dp array j_list ( 0 :
173         write ( *, * ) 'all preceding allocations for real dp arrays su
174                        LOS_list, pressure_list, density_list, temperat
175         write ( *, * ) 'allocation status variable = ', alloc_status
176         write ( *, * ) 'error message              = ', err_msg_alloca
177         stop 'memory allocation failure for list of j values (emissivi
178       end if
179
180
181     end subroutine create_mesh_sub
182
183 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
184 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
185 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
186
187     subroutine populate_temperature_sub ( self )           ! ideal gas
188
189       class ( thermo )                          :: self
190
191       ! ideal gas law EOS
192       self % temperature_list = self % pressure_list / self % density_
193
194       self % temperature_max = maxval ( self % temperature_list )
195       self % temperature_min = minval ( self % temperature_list )
196
197 !     map computed slope to HELIOS indices
198       self % temperature_slope  = ( self % temperature_index_max - sel
199                                   ( self % temperature_max - self % te
200       self % temperature_intcpt = self % temperature_slope
201
202     end subroutine populate_temperature_sub
203
204 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
205 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
206 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
207
208     subroutine populate_j_sub ( self )
209
210       class ( thermo )                            :: self
211
212       real ( dp ), intent ( in )                  :: xi   ! wall thicknes
213       integer ( lint )                            :: k
214
215       do k = 0, self % num_intervals
216         self % j_list ( k ) = dot ( amplitudes ( self ),
217                                     basis ( self, self % temperature_l
218       end do
219
220     end subroutine populate_j_sub
221
222 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
223 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
224 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
225
226     subroutine populate_pressure_sub ( self, xi, eta )           ! para
227
228       class ( thermo )                          :: self
229
230       real ( dp ), intent ( in )                :: xi   ! wall thicknes
231       real ( dp ), intent ( in )                :: eta  ! well depth
232       integer ( lint )                          :: k
233
234       do k = 0, self % num_intervals
235         self % pressure_list ( k ) = toy_fcn ( self, k, xi, eta )
236       end do
237 !      self % pressure_list = toy_fcn ( self, self % LOS_indices, xi, e
238
239       self % pressure_max = maxval ( self % pressure_list )
240       self % pressure_min = minval ( self % pressure_list )
241
242     end subroutine populate_pressure_sub
243
244 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
245 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
246 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
247
248     subroutine populate_density_sub ( self, xi, eta )           ! paral
249
250       class ( thermo )                        :: self
251
252       real ( dp ), intent ( in )              :: xi   ! wall thickness
253       real ( dp ), intent ( in )              :: eta  ! well depth
254       integer ( lint )                                :: k
255
256 !      self % density_list = toy_fcn ( self, indices, xi, eta )
257       do k = 0, self % num_intervals
258         self % density_list ( k ) = toy_fcn ( self, k, xi, eta )
259       end do
```

```
260
261       self % density_max = maxval ( self % density_list )
262       self % density_min = minval ( self % density_list )
263
264 !     could a pointer to a pressure structure clean up the typing
265 !     map computed slope to HELIOS indices
266       self % density_slope  = ( self % density_index_max - self % dens
267                               ( self % density_max - self % density_mi
268       self % density_intcpt = self % density_slope
269
270     end subroutine populate_density_sub
271
272 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
273 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
274 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
275
276     elemental function toy_fcn ( self, k, xi, eta ) result ( y )
277
278       class ( thermo ), intent ( in )        :: self
279
280       integer ( lint ), intent ( in )        :: k      !  in capsule c
281       real ( dp ),      intent ( in )        :: xi     !  wall thickne
282       real ( dp ),      intent ( in )        :: eta    !  well depth
283       real ( dp )                            :: x      !  in capsule c
284       real ( dp )                            :: delta  !  avoid divisi
285       real ( dp )                            :: y
286
287       delta = 0.000000953674316406 25_dp                ! 2^(-20)  avoi
288
289       x = g_fcn ( self, k )                             ! index -> caps
290
291       if ( x < ( one - xi ) ) then
292         y = ( eta / ( one - xi ) ** 2 ) * x ** 2 + ( one - eta )  !  q
293       else
294         y = ( ( delta - one ) / xi ) * ( x - one ) + delta        !  l
295       end if
296
```

```
297     end function toy_fcn
298
299 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
300 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
301 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
302
303     elemental function g_fcn ( self, k ) result ( x )
304
305       class ( thermo ), intent ( in )        :: self
306
307       real ( dp )                            :: x
308       integer ( lint ), intent ( in )        :: k
309
310       x = self % map_slope * k + self % map_intpt
311
312     end function g_fcn
313
314
315 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
316 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
317 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
318
319     elemental function X_fcn ( self, unscaled ) result ( scaled )
320
321       class ( thermo ), intent ( in )        :: self
322
323       real ( dp ), intent ( in )             :: unscaled
324       real ( dp )                            :: scaled
325
326       scaled = self % temperature_slope * unscaled + self % temperatu
327
328     end function X_fcn
329
330 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
331 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
332 !   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
333
```

```
334     elemental function Y_fcn ( self, unscaled ) result ( scaled )
335
336       class ( thermo ), intent ( in )        :: self
337
338       real ( dp ), intent ( in )             :: unscaled
339       real ( dp )                            :: scaled
340
341       scaled = self % density_slope * unscaled + self % density_slope
342
343     end function Y_fcn
344
345 end module plasma
```