NSCool User's guide

Structure of the Code

Dany Page

Instituto de Astronomía
Universidad Nacional Autónoma de México

The problem to be solved

The equations to be solved are described in the NSCool_Guide_1 Introduction. They are:

1) Structure of the star: the TOV equations.
2) Thermal evolution of the star.

How to use the TOV solver is described in NSCool_Guide_3_TOV. Meanwhile, several pre-built stars are available in the directory `TOV/Profile`.

For the thermal evolution equations, the star is cut at an outer boundary, with radius r_b and density ρ_b (typically $\rho_b = 10^{10}$ gm cm^{-3}): at $\rho > \rho_b$ matter is strongly degenerate and thus the structure of the star does not change with time:

The star's structure is calculated before the cooling and not modified thereafter.

(Almost: `NSCool` allows for small density changes in the outer part of the star, if required)

Only the energy balance and transport equations are solved as a function of time:

- two first order partial differential equations to get $L(r,t)$ and $T(r,t)$ with
- an initial L and T profile: $L(r,t=0)$ and $T(r,t=0)$
- two boundary conditions, at $r=0$ and $r=r_b$.

Note: the heat transport is a diffusion equation and numerically unstable if treated improperly. Numerical stability is achieved using an implicit scheme ("Henyey scheme") similar to the textbook Crank-Nicholson.

Rewriting the thermal evolution equations

The equations to solve:

Energy balance

$$\frac{d(Le^{2\Phi})}{dr} = -\frac{4\pi r^2 e^{\Phi}}{\sqrt{1-2Gm/c^2 r}}\left(C_v\frac{dT}{dt} + e^{\Phi}(Q_\nu - Q_h)\right)$$

Energy transport

$$\frac{d(Te^{\Phi})}{dr} = -\frac{1}{\lambda}\cdot\frac{Le^{\Phi}}{4\pi r^2\sqrt{1-2Gm/c^2 r}}$$

Use red-shifted functions: $\mathcal{T} \equiv e^{\Phi}T$ and $\mathcal{L} \equiv e^{2\Phi}L$

and the Lagrangian coordinate *a* (baryon number) $da = 4\pi r^2 dl\, n_B = \dfrac{4\pi r^2 n_B\, dr}{\sqrt{1-2Gm/c^2 r}}$

to get: $$\frac{d\mathcal{L}}{da} = -\frac{C_v}{n_B}\frac{d\mathcal{T}}{dt} - e^{2\Phi}\frac{Q_\nu - Q_h}{n_B} \quad \text{or} \quad \frac{d\mathcal{T}}{dt} = -e^{2\Phi}\frac{Q_\nu - Q_h}{C_v} - \frac{n_B}{C_v}\frac{d\mathcal{L}}{da}$$

and: $$\frac{d\mathcal{T}}{da} = -\frac{1}{\lambda}\frac{\mathcal{L}}{(4\pi r^2)^2 n_B\, e^{\Phi}} \quad \text{or} \quad \mathcal{L} = -\lambda\,(4\pi r^2)^2 n_B\, e^{\Phi}\,\frac{d\mathcal{T}}{da}$$

which we write as: $$\frac{d\mathcal{T}}{dt} = F\left(\mathcal{T}, \frac{d\mathcal{L}}{da}\right) \quad \text{and} \quad \mathcal{L} = G\left(\mathcal{T}, \frac{d\mathcal{T}}{da}\right)$$

(the $\mathcal{T}$ dependance of *F* and *G* comes from Q_v, Q_h, C_v, and λ)

Finite differencing the equations

For finite differencing these equations one divides the star into shells, at radii $r_0=0$, r_1, ... , r_i , ... r_{imax}. L, being a flux, is defined at the shell interfaces while T is understood as the average in the interior of each shell: it is common to write then L_i and $T_{i+½}$ to emphasize this.

Since fortran does not like loop indices with half integer values I used:

$\mathcal{L}$ is defined at $i = 0, 2, 4, \ldots , i_{max}-1$

$\mathcal{T}$ is defined at $i = 1, 3, 5, \ldots , i_{max}$

$$\frac{d\mathcal{T}}{dt} = F\left(\mathcal{T}, \frac{d\mathcal{L}}{da}\right) \quad \longrightarrow \quad \frac{d\mathcal{T}_i}{dt} = F\left(\mathcal{T}_i, \left.\frac{d\mathcal{L}}{da}\right|_i\right) \quad \text{with} \quad \left.\frac{d\mathcal{L}}{da}\right|_i = \frac{\mathcal{L}_{i+1} - \mathcal{L}_{i-1}}{da_{i-1} + da_i}$$

for i = 1, 3, 5, ...

$$\mathcal{L} = G\left(\mathcal{T}, \frac{d\mathcal{T}}{da}\right) \longrightarrow \mathcal{L}_i = G\left(\mathcal{T}|_i, \left.\frac{d\mathcal{T}}{da}\right|_i\right) \quad \text{with} \quad \mathcal{T}|_i = \frac{\mathcal{T}_{i+1} + \mathcal{T}_{i-1}}{2} \quad \text{and} \quad \left.\frac{d\mathcal{T}}{da}\right|_i = \frac{\mathcal{T}_{i+1} - \mathcal{T}_{i-1}}{da_{i-1} + da_i}$$

for i = 2, 4, 6, ...

where da_i is the number of baryons between r_{i-1} and r_i

Stepping forward in time

Assuming we know the profiles of $\mathcal{T}$ and $\mathcal{L}$ at time *t*: $\mathcal{T}^{\text{old}}$ and $\mathcal{L}^{\text{old}}$

we can write for $\mathcal{T}$ and $\mathcal{L}$ at time *t'=t+dt*:

Explicit scheme

$$\frac{d\mathcal{T}}{dt} = F\left(\mathcal{T}, \frac{d\mathcal{L}}{da}\right) \quad \longrightarrow \quad \mathcal{T} = \mathcal{T}^{\text{old}} + dt \cdot F\left(\mathcal{T}^{\text{old}}, \frac{d\mathcal{L}^{\text{old}}}{da}\right)$$

$$\mathcal{L} = G\left(\mathcal{T}, \frac{d\mathcal{T}}{da}\right) \quad \longrightarrow \quad \mathcal{L} = G\left(\mathcal{T}^{\text{old}}, \frac{d\mathcal{T}^{\text{old}}}{da}\right)$$

this is very easy to integrate BUT:

it is numerically unstable unless *dt* is very small (Courant *dixit*)

Better: evaluate *F* and *G* at the new values of $\mathcal{T}$ and $\mathcal{L}$:

Implicit scheme

$$\frac{d\mathcal{T}}{dt} = F\left(\mathcal{T}, \frac{d\mathcal{L}}{da}\right) \quad \longrightarrow \quad \mathcal{T} = \mathcal{T}^{\text{old}} + dt \cdot F\left(\mathcal{T}, \frac{d\mathcal{L}}{da}\right)$$

$$\mathcal{L} = G\left(\mathcal{T}, \frac{d\mathcal{T}}{da}\right) \quad \longrightarrow \quad \mathcal{L} = G\left(\mathcal{T}, \frac{d\mathcal{T}}{da}\right)$$

this is numerically stable (and allows large *dt*) BUT:

extracting the new $\mathcal{T}$ and $\mathcal{L}$ is tough

(particularly $\mathcal{T}$ because it is inside Q_v, Q_h, C_v, and λ)

Solving the implicit equations by iterations

Assuming we know the profiles of $\mathcal{T}$ and $\mathcal{L}$ at time *t*: $\mathcal{T}^{\text{old}}$ and $\mathcal{L}^{\text{old}}$
we can find the new $\mathcal{T}$ and $\mathcal{L}$ at time *t'=t+dt* by successive approximations

$$(\mathcal{T}^{(0)}, \mathcal{L}^{(0)}) \rightarrow (\mathcal{T}^{(1)}, \mathcal{L}^{(1)}) \rightarrow (\mathcal{T}^{(2)}, \mathcal{L}^{(2)}) \rightarrow (\mathcal{T}^{(3)}, \mathcal{L}^{(3)}) \rightarrow \ \ldots$$

As an initial guess for $(\mathcal{T}^{(0)}, \mathcal{L}^{(0)})$ one can take $(\mathcal{T}^{(0)}, \mathcal{L}^{(0)}) = (\mathcal{T}^{\text{old}}, \mathcal{L}^{\text{old}})$
or extrapolate from $(\mathcal{T}^{\text{old}}, \mathcal{L}^{\text{old}})$ and the previous values $(\mathcal{T}^{\text{older}}, \mathcal{L}^{\text{older}})$.

Evaluate the functions F and G with $\mathcal{T}_i^{(k)}$ and $\mathcal{L}_i^{(k)}$ to obtain $\mathcal{T}_i^{(k+1)}$ and $\mathcal{L}_i^{(k+1)}$:

$$\mathcal{T}_i^{(k+1)} = \mathcal{T}_i^{\text{old}} + dt \cdot F\left(\mathcal{T}_i^{(k)}, \left.\frac{d\mathcal{L}}{da}\right|_i^{(k)}\right) \qquad \mathcal{L}_i^{(k+1)} = G\left(\mathcal{T}|_i^{(k)}, \left.\frac{d\mathcal{T}}{da}\right|_i^{(k)}\right)$$

then plug back $\mathcal{T}_i^{(k+1)}$ and $\mathcal{L}_i^{(k+1)}$ into F and G to obtain $\mathcal{T}_i^{(k+2)}$ and $\mathcal{L}_i^{(k+2)}$
and so on until some K when $\mathcal{T}_i^{(K+1)} \cong \mathcal{T}_i^{(K)}$ and $\mathcal{L}_i^{(K+1)} \cong \mathcal{L}_i^{(K)}$

(All $\mathcal{T}_i^{(K)}$ and $\mathcal{L}_i^{(K)}$ are successive approximations *at the same time t'*. $\mathcal{T}_i^{\text{old}}$ does not change, it is at time t !)

As long as the initial guess $(\mathcal{T}^{(0)}, \mathcal{L}^{(0)})$ is not too far from the solution
the method will converge to the solution (maybe in 10 iterations ?)

Improvement: the Henyey scheme

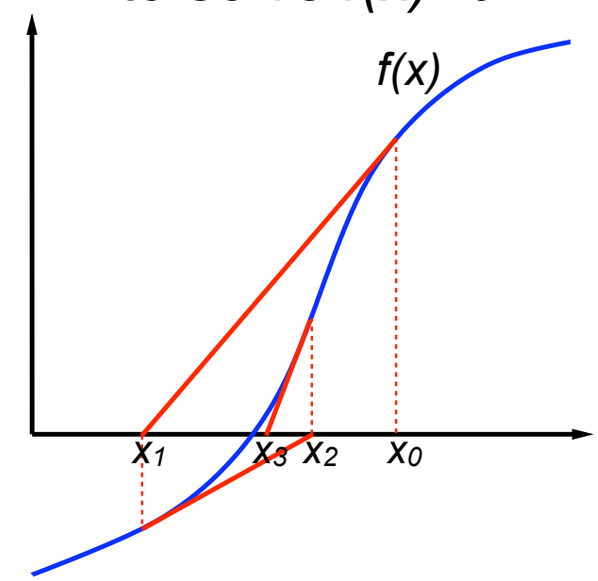
Instead of using brute force iterations, the Henyey scheme use the Newton-Raphson method for solving multi-dimensional equations.

Write the equations as:
$$\begin{cases} \mathcal{T} - \mathcal{T}^{\rm old} - dt \cdot F\left(\mathcal{T}, \frac{d\mathcal{L}}{da}\right) = 0 \\ \mathcal{L} - G\left(\mathcal{T}, \frac{d\mathcal{T}}{da}\right) = 0 \end{cases}$$

or, in N dimensional notation:

$$\Phi(X) = 0 \quad \text{with} \quad X = \begin{pmatrix} \mathcal{L}_0 \\ \mathcal{T}_1 \\ \mathcal{L}_2 \\ \mathcal{T}_3 \\ . \\ . \end{pmatrix} \quad \text{and} \quad \Phi(X) = \begin{pmatrix} \Phi_0(X) \\ \Phi_1(X) \\ \Phi_2(X) \\ \Phi_3(X) \\ . \\ . \end{pmatrix}$$

The Newton method to solve *f(x)=0*



$$f(x_{k+1}) = 0 \quad \leftrightarrow$$
$$f(x_k) + f'(x_k) \cdot (x_{k+1} - x_k) = 0$$
$$\Longrightarrow x_{k+1} = x_k - [f'(x_k)]^{-1} \cdot f(x_k)$$

and the Newton-Raphson iteration procedure is: $X^{(k+1)} = X^{(k)} - [D\Phi(X^{(k)})]^{-1} \cdot \Phi(X^{(k)})$

where $[D\Phi(X)]$ is the NxN derivative matrix of $\Phi(X)$ and $[D\Phi(X)]^{-1}$ the inverse matrix.

This involves calculating T derivatives of Q_v, Q_h, C_v, and λ and inverting a large matrix. Fortunately this matrix is tri-diagonal and its inversion is straightforward !

One still have to preform iterations but the convergence can be much faster than brute force.

Checking for iteration convergence and time step control

The Newton-Raphson iterations go as:

$$\mathcal{T}_i^{(k)} \rightarrow \mathcal{T}_i^{(k+1)} = \mathcal{T}_i^{(k)} + \delta\mathcal{T}_i^{(k)} \quad [i=1, 3, 5, ...]$$
$$\mathcal{L}_i^{(k)} \rightarrow \mathcal{L}_i^{(k+1)} = \mathcal{L}_i^{(k)} + \delta\mathcal{L}_i^{(k)} \quad [i=0, 2, 4, ...]$$

Convergence will be considered to have been achieved when

$$\mathrm{Max}_{i=1,3,5,\ldots}\left(\frac{\delta\mathcal{T}_i^{(k)}}{\mathcal{T}_i^{(k)}}\right) < \epsilon_T \quad \text{and} \quad \mathrm{Max}_{i=0,2,4,\ldots}\left(\frac{\delta\mathcal{L}_i^{(k)}}{\mathcal{L}_i^{(k)}}\right) < \epsilon_L$$

Values of ε_T and ε_L of the order of 10^{-10} can be reached in 4 - 6 iterations.
However, if $\mathcal{T}_i^{(0)}$ and/or $\mathcal{L}_i^{(0)}$ are too far away from the solution, iterations go on forever:
the loop is exited, the time step *dt* is shortened and the iteration procedure restarted.

(It is not unusual to see *dt* being cut many times, e.g., when a phase transition (superfluidity/superconductivity) occurs at some point in the star. Sometimes things go real bad (dt → almost zero): "`Ctrl-C`" is the only solution, and figure out what's happening.)

Time step control: at every new time step *dt* is increased: *dt* → *dt (1+α)* (*α ~ 0.2*) but:

- if Newton-Raphson converged in << 5 steps a larger α is chosen
- if Newton-Raphson needed > 10 steps to converge a smaller α is chosen
- if $\mathcal{T}$ and/or $\mathcal{L}$ changed too much (from $\mathcal{T}^{\mathrm{old}}$ and/or $\mathcal{L}^{\mathrm{old}}$) a smaller α is chosen,
  while if they changed ways too much, the time step is recalculated with a smaller *dt*.

The boundary conditions

Inner boundary condition: $L(r=0) = 0$ or $\mathcal{L}_{i=0} = 0$

This is easily implemented by initially starting with $\mathcal{L}_{i=0}^{(k=0)} = 0$ and imposing $\delta\mathcal{L}_{i=0}^{(k)} = 0$ at every iteration.

Outer boundary condition (see `NSCool_Guide_1_Introduction`):

It is (at $r = r_b$): $L(r_b) = 4\pi R^2 \sigma_{SB} [T_e(T_b)]^4$ with $T_b \equiv T(r_b)$

where (in present notations): $L(r_b) = e^{-2\Phi(i_{max}-1)}\, \mathcal{L}(i_{max}-1)$ *and* $T(r_b) = e^{-\Phi(i_{max})}\, \mathcal{T}(i_{max})$
and $T_e(T_b)$ is a function (a "T_e-T_b" relationship) obtained from some envelope model.

This is implemented as part of the inversion of the matrix $[D\Phi(X)]$

Add more details about this !

Flow diagram of NSCool

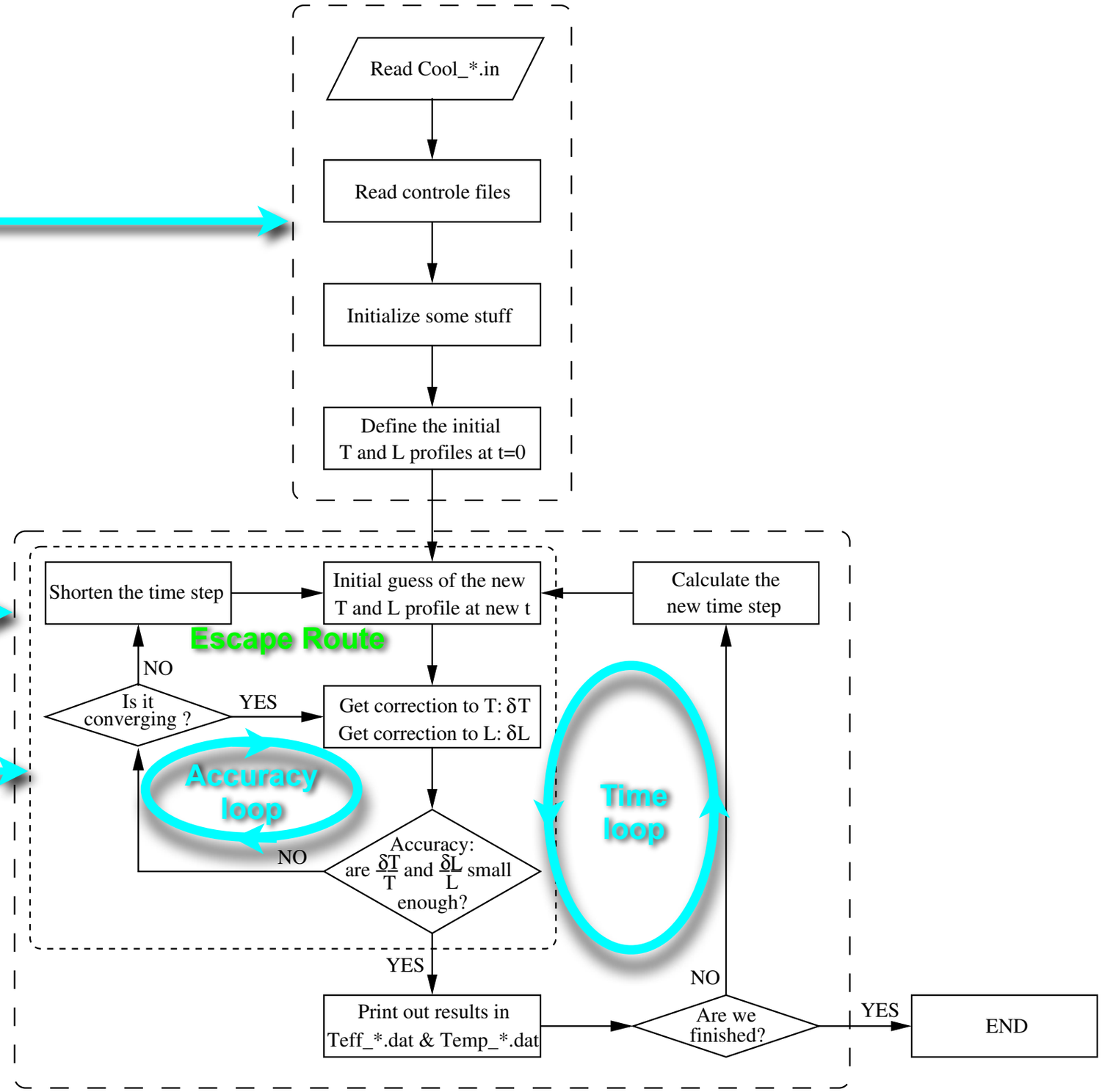
Initialization

Time integration

.

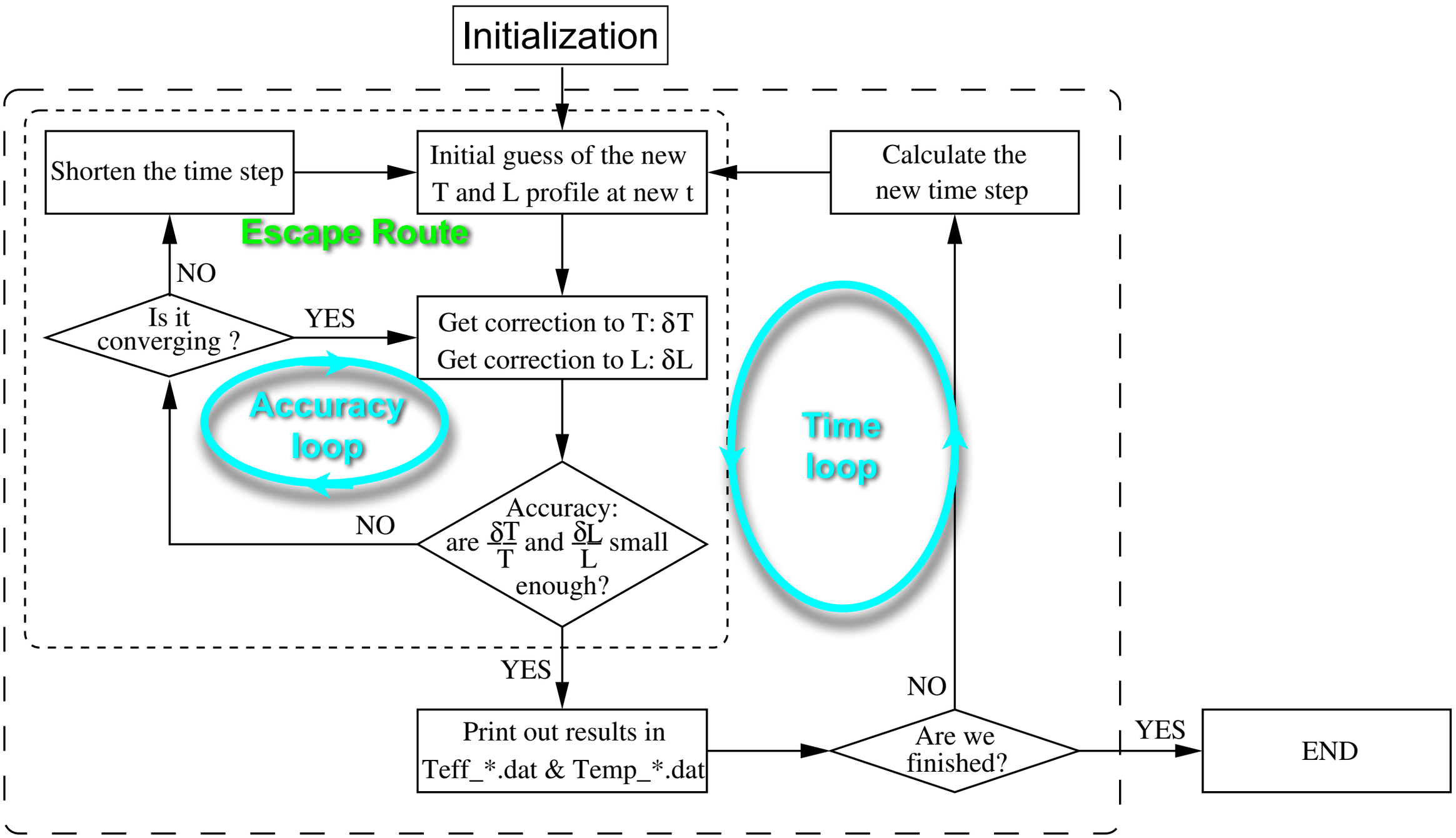
.

with iterations
to reach accuracy
in T & L profiles
at each time step



Notice: `NSCool` contains an extra “model loop” to run several cooling models from the same `Cool_*.in` file.

Flow diagram of NSCool (bigger)



Reading the NSCool.f file

The next slides describe the structure of NSCool.f :

***implementation* of the previous flow diagram**

There are many sections of just screen print out (unimportant for now). They are all marked the same way between two line of:

c ----------

.

.

c ----------

Lines like these would include commands for magnetic field evolutions (not used anymore).

NSCool.f

```
      end if
     end if
     call accretion_rate(time+dtime,dtime,m_dot)
     call accretion_velocity(m_dot)

c -----------------------------------------------------------------
     if (pscreen.ge.2) then
c      if (pscreen.eq.3) read(5,*)
      read(5,*)
      write(6,*)
      print '(1a28,1a50)','***************************',
     1 '**************************************************'
      print '(2a10,1i5,1a53)','**********','step#=',istep,
     1 '**************************************************'
      print '(1a28,1a50)','***************************',
     1 '**************************************************'
      print '(2(a10,1p1e10.3),a30,0p1f6.3)',
     1              'time =',(time+dtime)/year,
     2             'dtime =',dtime/year,
     3       'dtime/odtime =',dtime/odtime
      print *
      if (chtemp.eq.1.) then
       print '(a42,0p1f5.2,a9,1p1e9.2,a3,1p1e9.2)',
     1        'dtime limited by TEMP change, mdtemp =',mdtemp,
     2        'at rho=',rrho(icht),'T=',temp(icht)
      end if
      if (chstoke.eq.1.) then
       print '(a42,0p1f5.2,a9,1p1e9.2,a3,1p1e9.2)',
     1        'dtime limited by STOKE change, mdstoke =',mdstoke,
     2        'at rho=',rrho(ichs),'S=',stoke(ichs)
      end if
      if (chtrial.eq.1) then
       print '(a40)',
     1        '    dtime limited by ITRIAL'
      end if
     end if
c -----------------------------------------------------------------

c ***** Calculate ntemp & nlum for first guess ***********************

     if (debug.ge.1.) print *,'Guessing NLum & NTemp'
     coeff_int=0.8d0
-:** NSCool.f     35% (490,13)   (Fortran)
```

Line number !

Location in the file !

NSCool.f

```
c BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
c      INCLUDE 'Bfield/Bfield_3.inc.f'
c BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
-:-- NSCool.f     38% (476,22)   (Fortran)
```

Do not trust the line number too much: any small change in the code change them !

Essential variables in NSCool.f

Code variable	
time	t
dtime	dt
temp(i)	$\mathcal{T}_i^{old}$
ntemp(i)	$\mathcal{T}_i^{(k)}$
dtemp(i)	$d\mathcal{T}_i^{(k)}/da$
delt(i)	$\delta\mathcal{T}_i^{(k)}$
lum(i)	$\mathcal{L}_i^{old}$
nlum(i)	$\mathcal{L}_i^{(k)}$
dlum(i)	$d\mathcal{L}_i^{(k)}/da$
dell(i)	$\delta\mathcal{L}_i^{(k)}$

Code variable	
istep	time index
itrial	iteration index

Code variable	
rad(i)	r
rrho(i)	ρ
debar(i)	da

The new time is
t'=t+dt
but there is no variable for t'

The “model” loop

Beginning of the loop:

This runs through the various models listed in Cool_*.in , and will cool all of them, one after the other.

End of the loop:

Close the output files (`Teff_*.dat` & `Temp_*.dat`) and go to next model.

Now, it’s really finished. Close the master input file (`Cool_*.in`)

NSCool.f

```
c ***********************************************************************
c ***********************************************************************
c *********************     LET'S GO !     **************************
c ***********************************************************************
c ***********************************************************************

      i_model=0

c ***********************************************************************
c ***********************************************************************
c ************   BEGINNING OF A NEW MODEL CALCULATION   ***************
c ***********************************************************************
c ***********************************************************************

 1234 continue

      i_model=i_model+1

      idt=1
      htot=0.0
      contraction=0.d0

c ***********************************************************************
-:** NSCool.f     12% (171,0)   (Fortran)

c Close the output file for the present model: ************************
      close(unit=10,status='keep')
      close(unit=19,status='keep')

c Go back to the beginning: do the next model listed in the "Input file
      goto 1234

c ***********************************************************************
 9997 continue    ! This is the real end of it !
c ***********************************************************************

      close(unit=15,status='keep')

      print *,'Done !'

      end
c ***********************************************************************
c ***********************************************************************
c ***********************************************************************
c ***********************************************************************
c ***********************************************************************
-:** NSCool.f     98% (1275,15)  (Fortran)
```

The “Input File” (Cool_*.in)

If just beginning (i_model=1): ask for the master input file (Cool_*.in) and open it.

then:
read the cooling model files

NSCool.f

```
c ***********************************************************************
c ****************     GET INPUT MODEL FILES     **********************
c ***********************************************************************

      if (i_model.eq.1) then
c ***  Choose between two input: ************************************
C      Ask for the input file ***************************************
       write(6,*) 'Input master file ='
       read(5,*)filename
c ***  Can add here the directory where "Cool_*.in" is:
       filename='Model_1/'//filename
c ***  Or define it completely here: ********************************
c       filename='Model_1/Cool_Try.in'
c       write(6,*)'Using as input: ',filename
c***********************************************************************
       open(unit=15,file=filename,status='old')
      else
       read(15,*,end=9997,err=9997)
      end if
      read(15,*,end=9997,err=9997)version
      if (version.eq.'STR') then
       istrange=1
      else
       istrange=0
      end if
c *** BASIC MODEL FILES: *********************************************
      read(15,*,end=9997,err=9997)
      read(15,*,end=9997,err=9997)f_crustcc
      read(15,*,end=9997,err=9997)f_stareos
      read(15,*,end=9997,err=9997)f_profile
c *** OTHER MODEL FILES: *********************************************
      read(15,*,end=9997,err=9997)
      read(15,*,end=9997,err=9997)f_Structure
      read(15,*,end=9997,err=9997)f_Bound
      read(15,*,end=9997,err=9997)f_Pairing
      read(15,*,end=9997,err=9997)f_Neutrino
      read(15,*,end=9997,err=9997)f_Conduct
      read(15,*,end=9997,err=9997)f_Heat
      read(15,*,end=9997,err=9997)f_Bfield
      read(15,*,end=9997,err=9997)f_Accretion
      if (istrange.eq.1) then
       read(15,*,end=9997,err=9997)f_Strange
      end if
c *** OUTPUT FILES: **************************************************
      read(15,*)
      read(15,*,end=9997,err=9997)f_i
      read(15,*,end=9997,err=9997)f_Teff
      read(15,*,end=9997,err=9997)f_Temp
      read(15,*,end=9997,err=9997)f_Star
c***********************************************************************
```

-:** NSCool.f 14% (203,56) (Fortran)

Initialization (1)

Open and read the model files (the ones listed in `Cool_*.in`) [all this is in `NSCool_READ.inc.f`]

Call a bunch of subroutines [more on this later]

Calculate some pieces of physics (as, e.g., the e^{Φ}'s,...)

NSCool.f

```
c ***************************************************************
c ****************     READ THE ABOVE FILES     *********************
c ***************************************************************

      INCLUDE 'NSCool_READ.inc.f'

c ***************************************************************
c ****************        INITIALIZATION        *********************
c ***************************************************************
      if (debug.ge.1.) print *,'Initializing'

c ***************************************************************
c *** Get the time independent pieces of physics: ******************
c ***************************************************************
c     get_core_chemistry MUST be called BEFORE get_crust_chemistry
```

-:** NSCool.f 20% (259,0) (Fortran)

NSCool.f

```
c ***************************************************************
c ***** Calculate the T-independent coefficients *********************
c ***************************************************************
      if (debug.ge.1.) print *,'Calculating T-independent coeff.'
```

-:** NSCool.f 23% (287,31) (Fortran)

Initialization (2)

Call some more subroutines
[more on this later]

NSCool.f

```
c ***********************************************************************
c *** Initialize some more stuff: ***********************************
c ***********************************************************************

      if (i_heat_deep_crust.eq.1) then
       call initialize_heating_deep_crust(f_heat_deep_crust)
      end if
      if (i_heat_thermonuclear.eq.1) then
       call initialize_heating_thermonuclear(
```

-:** NSCool.f 25% (316,38) (Fortran)

Calculate the initial (*t=0*)
$\mathcal{T}$ & $\mathcal{L}$ profiles

NSCool.f

```
c ***********************************************************************
c ********* Calculate the initial Temp and Lum profiles **************
c ***********************************************************************

      INCLUDE 'NSCool_INIT_TPROF.inc.f'

c BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
c      INCLUDE 'Bfield/Bfield_2.inc.f'
c BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

c ***********************************************************************
c ***************** Open print out files ****************************
c ***********************************************************************

      INCLUDE 'NSCool_OPEN.inc.f'

c ***********************************************************************
c ***********************************************************************
```

-:** NSCool.f 28% (361,0) (Fortran)

Open the output files
[all this is in `NSCool_OPEN.inc.f`]

The “time loop”

Beginning of the loop:

The time stepping loop (after some little set-up, as resetting the time !)

NSCool.f

```
c ***********************************************************************
c ***********************************************************************
c ************************     COOLING     **************************
c ***********************************************************************
c ***********************************************************************

c ***********************************************************************
      time=time0        ! Initialize the time
      icycle=0          ! Initialize the counter for accretion cycles
c ***********************************************************************

c ***********************************************************************
c      itprint=0        ! To print out the initial T and L profiles
      itprint=1         ! To print out only at the required times
c ***********************************************************************

c ***********************************************************************
c     THIS IS THE MAIN TIME LOOP:
      do 9999 istep=1,istepmax
c ***********************************************************************
```

-:** NSCool.f 29% (386,0) (Fortran)

End of the loop:

NSCool.f

```
c ***********************************************************************

      if (time/year.ge.timemax) goto 9998
      if ((sign_l*teffective).lt.tempmin) goto 9998

c ***********************************************************************
9999  continue      ! This is the `end do' for the main time integration
c ***********************************************************************

c ***********************************************************************
9998  continue      ! Jump here if time > timemax
c ***********************************************************************

c Close the output file for the present model: ************************
```

-:** NSCool.f 97% (1272,0) (Fortran)

It stops when you run out of time steps or you run out of time

Prepare for iterations

Calculate the accretion rate (in case there is accretion)

NSCool.f

```
c *** Accretion rate: ************************************************
      icycle_old=icycle
      if ((i_acc.eq.1).or.(i_acc.eq.2)) then
       if (time+dtime.ge.t_acc0) then
        icycle=int((time+dtime-t_acc0)/t_acc1)+1
        t_burst= time+dtime -t_acc0 - float(icycle-1)*t_acc1
                 ! t_burst = time since beginning of burst
       else
        icycle=0
        t_burst=0.d0
       end if
      end if
      call accretion_rate(time+dtime,dtime,m_dot)
      call accretion_velocity(m_dot)
c ***********************************************************************
```

-:** NSCool.f 32% (418,0) (Fortran)

Calculate the first guess profiles: $\mathcal{T}_i^{(k=0)}$ and $\mathcal{L}_i^{(k=0)}$

NSCool.f

```
c ***********************************************************************
c ***** Calculate ntemp & nlum for first guess ***********************
c ***********************************************************************

      if (debug.ge.1.) print *,'Guessing NLum & NTemp'
      coeff_int=0.8d0
      do i=1,imax,2
       ntemp(i)=temp(i)+coeff_int*(temp(i)-otemp(i))*dtime/odtime
      end do
      dtemp(0)=0.d0
      do i=2,imax-1,2
       dtemp(i)=(ntemp(i+1)-ntemp(i-1))/(debar(i)+debar(i+1))
      end do

      nlum(0)=0.
      do i=2,imax-1,2
       nlum(i)=lum(i)+coeff_int*(lum(i)-olum(i))*dtime/odtime
      end do
      do i=1,imax-2,2
       dlum(i)=(nlum(i+1)-nlum(i-1))/(debar(i)+debar(i+1))
      end do
```

-:** NSCool.f 36% (464,0) (Fortran)

The Newton-Raphson loop

Reset the iteration loop counter:
(this is also a branch point in case of failure)

Beginning of the iteration loop:

Increment the iteration counter

Escape Route: First Exit

Too many iterations: it is not converging (start again with a smaller time step *dt*)

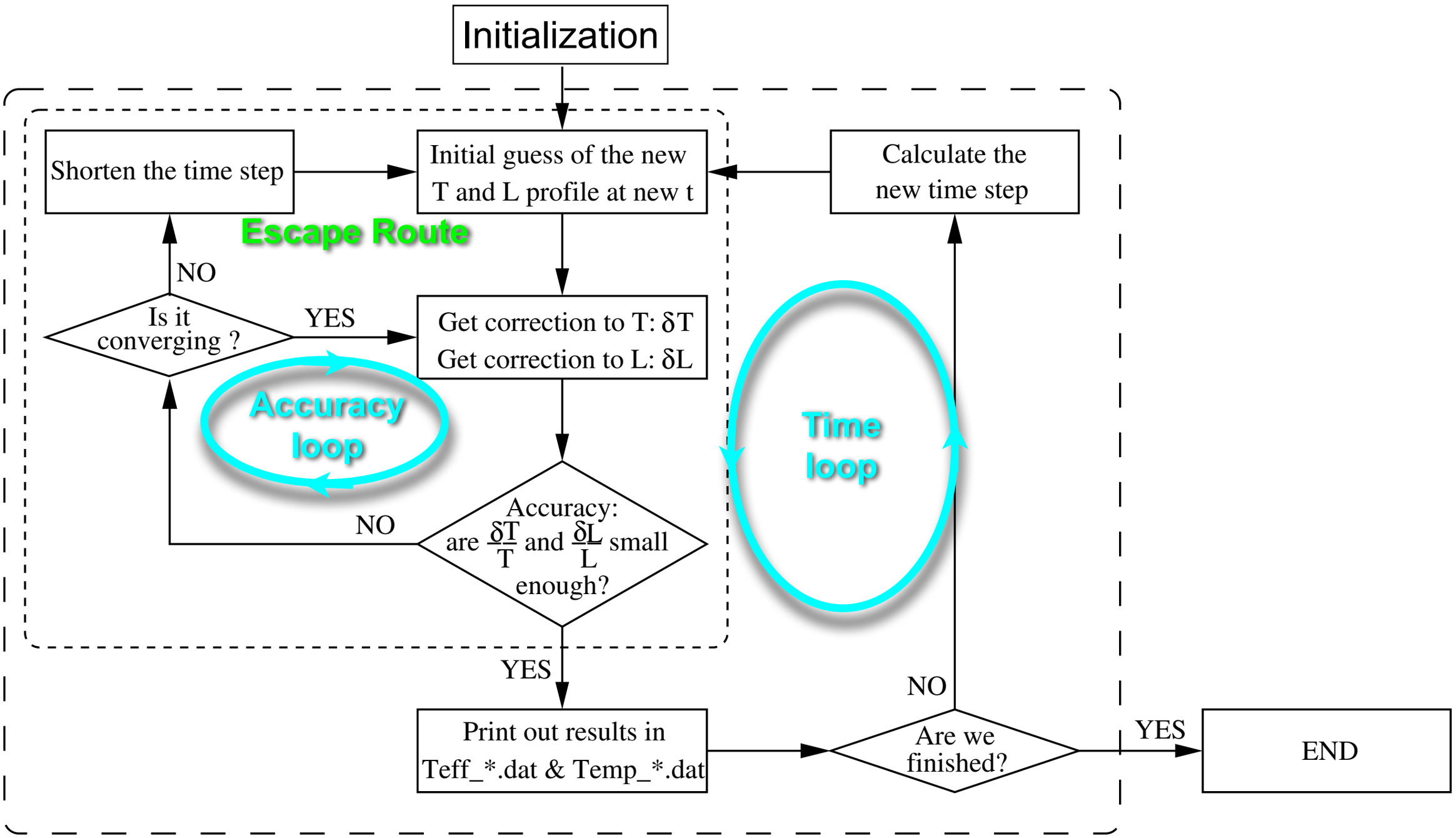
End of the iteration loop:

- Converged: go to next time step.
- Not converged: go to next iteration.

NSCool.f

```
c ***********************************************************************
c     THIS IS THE MAIN TIME LOOP:
      do 9999 istep=1,istepmax
c ***********************************************************************

        debug=0.
        if (istep.ge.istep_debug) debug=debug_keep
        if (debug.ge.1.) print *,'Going: istep=',istep

c ***********************************************************************
2345  itrial=0          ! Branch back here in case:
                        !  - Too many iteration in Newton-Raphson
                        !  - Envelope boundary condition cannot be solved
                        !  - Temp has changed too much
c ***********************************************************************
-:** NSCool.f     31% (397,0)    (Fortran)
c ***********************************************************************
c ***********************************************************************
c ***** Branch here if new trial ***************************************
c ***********************************************************************
c ***********************************************************************

c ***********************************************************************
2000  itrial=itrial+1                  ! This is the Newton-Raphson loop
c ***********************************************************************
       if (itrial.eq.itrial_max+1)then
        tcut=dsqrt(dt0)
        if (time.le.1.e5) tcut=dsqrt(dt1)
        dtime=dtime/tcut
        goto 2345
       end if
-:** NSCool.f     39% (498,0)    (Fortran)

c ***********************************************************************
c     Decide if converged or not:
      if ((ratiot.lt.mratt).and.(ratiol.lt.mratl).and.(ratios.lt.mrats))
     x then
        continue          ! Converged ! continue to next time step
       else
        goto 2000         ! Not converged ! Go back for another iteration
       end if
c ***********************************************************************
-:** NSCool.f     66% (855,0)    (Fortran)
```

Flow diagram of NSCool (bigger)



Prepare matrix $[D\Phi(X^{(k)})]$

Adjust density in outer part, if required.

Calculates physics (Q_v, Q_h, C_v, and λ) at $T_i^{(k)}$

$T_i^{(k)}$ is changed to $(1-\varepsilon)\cdot T_i^{(k)}$ to calculate the derivatives

Readjust density in outer part, if required.

Recalculates physics (Q_v, Q_h, C_v, and λ) at $(1-\varepsilon)\cdot T_i^{(k)}$

NSCool.f

```
c ***********************************************************************
c ***** Calculate the new density in inner envelope at ntemp **********
c ***********************************************************************

      if (debug.ge.1.) print *,'Calculating new density at NTemp'
      do i=imax-1,ienv+1,-2
-:** NSCool.f     41% (518,0)    (Fortran)

c ***********************************************************************
c ***** Calculate the physical parameters at ntemp ********************
c ***********************************************************************

      if (debug.ge.1.) print *,'Calculating physics at NTemp'
      do i=1,imax,2
       t=ntemp(i)/ephi(i)
       d=rrho(i)
       a=a_cell(i)
       a1=a_ion(i)
       z=z_ion(i)
       call neutrino(i,t,d,a,z,qnu(i),
-:** NSCool.f     43% (544,0)    (Fortran)

c ***********************************************************************
c ***** Calculate the new density at (1-tinc)*ntemp *******************
c ***********************************************************************

      if (debug.ge.1.) print *,'Calculating density at NTemp'''
      tinc=max(1.d-12,ratiot/1.d1)
      do i=1,imax,2
       ntemp1(i)=ntemp(i)*(1.d0-tinc)
      end do

      do i=imax-1,ienv+1,-2
-:** NSCool.f     46% (590,0)    (Fortran)

c ***********************************************************************
c ***** Calculate the physical parameters at (1-tinc)*ntemp ***********
c ***********************************************************************

      if (debug.ge.1.) print *,'Calculating physics at NTemp'''
      do i=1,imax,2
       t=ntemp1(i)/ephi(i)
       d=rrho1(i)
-:** NSCool.f     48% (623,0)    (Fortran)
```

Calculate $[D\Phi(X^{(k)})]$, invert it and get $X^{(k+1)}$

Calculate the matrix elements $\frac{d\Phi_i}{dX_j}$

Calculate the $\Phi_i(X^{(k)})$

Calculate $[D\Phi(X^{(k)})]^{-1}$

Calculate the new $X_i^{(k+1)}$

$$X^{(k+1)} = X^{(k)} - [D\Phi(X^{(k)})]^{-1} \cdot \Phi(X^{(k)})$$

NSCool.f

```
     2        *(dlog(rrho1(i))-dlog(orrho(i)))/dtime*contraction
      end do

c ***********************************************************************
c ***** Calculate the derivatives of fp,fq & fr *********************
c ***********************************************************************

      if (debug.ge.1.) print *,'Calculating derivatives of FP, FQ, FR'
       do i=1,imax,2
        t =ntemp(i)
-:** NSCool.f     51% (691,1)   (Fortran)
       end do

c ***********************************************************************
c ***** Calculate ff ************************************************
c ***********************************************************************

      if (debug.ge.1.) print *,'Calculating FF'
      ff(0)=0.d0
      do i=2,imax-1,2
       ff(i)=nlum(i)+.5d0*(fp(i-1)+fp(i+1))*a2ephin(i)*dtemp(i)
       ff(i-1)=fr(i-1)+fq(i-1)*dlum(i-1)+(ntemp(i-1)-temp(i-1))/dtime
-:** NSCool.f     52% (713,21)   (Fortran)

c ***********************************************************************
c ***** Newton-Raphson method ***************************************
c ***********************************************************************

      if (debug.ge.1.) print *,'Newton-Raphson'
      do i=2,imax-1,2
       fa(i)=.5d0*dfp(i+1)*a2ephin(i)*dtemp(i)+
     1        .5d0*(fp(i+1)+fp(i-1))*a2ephin(i)/(debar(i)+debar(i+1))
       fb(i)=.5d0*dfp(i-1)*a2ephin(i)*dtemp(i)-
-:** NSCool.f     53% (729,0)   (Fortran)
      end if

c ***********************************************************************
c ****** Get ntemp & nlum *******************************************
c ***********************************************************************

      if (debug.ge.1.) print *,'Getting NTemp & NLum'
      delt(imax)=ntp-ntemp(imax)
      do i=imax-2,1,-2
       dell(i+1)=fk(i+1)-fj(i+1)*delt(i+2)
       dell(i+1)=sign(min(2.d3*abs(nlum(i+1)),abs(dell(i+1))),
-:** NSCool.f     58% (792,0)   (Fortran)
```

The outer boundary condition

This solves the condition:

$$L(r_b) = 4\pi R^2 \sigma_{SB} [T_e(T_b)]^4 \quad \text{with} \quad T_b \equiv T(r_b)$$

using Newton's method.
The function fteff(...) is $T_e(T_b)$.

Escape Route: Second Exit

Newton fails to find the solution (start again with a smaller time step *dt*)

NSCool.f

```
c ***********************************************************************
c **************** Boundary condition ***************************
c ***********************************************************************

      if (debug.ge.1.) print *,'Boundary Condition'
      if (ifteff.ne.15) then
       epsilon=1.d-8
       precision=1.d-12
       coeff=4.d0*pi*radius**2*5.67d-5*e2phi(imax)/lsol
       lhs=nlum(imax-1)+fk(imax-1)+fj(imax-1)*ntemp(imax)
       ntp=ntemp(imax)
       tp0_keep=ntp
7654   tp0=ntp
       teff0=fteff(tp0/ephi(imax),ifteff,eta,bf_r(imax),istep,
     1             time,ts1,ts2,z_ion(imax),a_ion(imax),rrho(imax),
     2             debug)
       if(debug.eq.-50.) print *,'Tb0, Te0 =',tp0,teff0
       tp1=(1.d0+epsilon)*tp0
       teff1=fteff(tp1/ephi(imax),ifteff,eta,bf_r(imax),istep,
     1             time,ts1,ts2,z_ion(imax),a_ion(imax),rrho(imax),
     2             debug)
       if(debug.eq.-50.) print *,'Tb1, Te1 =',tp1,teff1
       derivative=coeff*(teff1**4-teff0**4)/(epsilon*tp0)
       derivative=-fj(imax-1)-derivative
       if(debug.eq.-50.) print *,'Derivative =',derivative
       function=lhs-fj(imax-1)*tp0-coeff*teff0**4
       if(debug.eq.-50.) print *,'Function =',function
       ntp=tp0-function/derivative
       if(debug.eq.-50.) print *,'Del(Tp)/Tp =',abs(tp0-ntp)/tp0
       if(debug.eq.-50.) print *,'------> New Tb =',ntp
       if ((ntp.le.0.).or.(ntp.gt.1.e12)) then ! In case the method diverges
                                  ! restart iterations with shorter time step
        tcut=dsqrt(scale_dt0)
        if (time.le.1.e5) tcut=dsqrt(scale_dt1)
        dtime=dtime/tcut
        goto 2345
       end if
       if(abs(tp0-ntp)/tp0.gt.precision)goto 7654
      else
       ntp=tb_acc0            ! Fixed T_b for accretion
      end if

c ***********************************************************************
-:-- NSCool.f      55% (712,0)    (Fortran)
```

Check accuracy

Calculate the:

$$\mathrm{Max}_{i=1,3,5,\ldots}\left(\frac{\delta\mathcal{T}_i^{(k)}}{\mathcal{T}_i^{(k)}}\right)$$

$$\mathrm{Max}_{i=2,4,6,\ldots}\left(\frac{\delta\mathcal{L}_i^{(k)}}{\mathcal{L}_i^{(k)}}\right)$$

to check accuracy and convergence

As seen previously, next comes the end of the iteration loop:

- Converged: go to next time step.
- Not converged: go to next iteration.

NSCool.f

```
c ***********************************************************************
c ***** Analyze the results to see if it has converged ****************
c ***********************************************************************

      if (debug.ge.1.) print *,'Analyzing Results'
      ratiot=0.d0
      ratiol=0.d0
      do i=1,imax-2,2
       ratl=abs(dell(i+1))/(abs(nlum(i+1))+1.d-12)
       if(ratl.gt.ratiol)then
        ratiol=ratl
        iratl=i+1
       end if
       ratt=abs(delt(i)/(ntemp(i)+1.d-30))
       if (ratt.gt.ratiot)then
        ratiot=ratt
        iratt=i
       end if
      end do
```

-:** NSCool.f 63% (822,0) (Fortran)

```
c ***********************************************************************
c     Decide if converged or not:
      if ((ratiot.lt.mratt).and.(ratiol.lt.mratl).and.(ratios.lt.mrats))
     x then
        continue        ! Converged ! continue to next time step
       else
        goto 2000       ! Not converged ! Go back for another iteration
       end if
c ***********************************************************************
```

-:** NSCool.f 66% (856,0) (Fortran)

Note: "ratios" was the same thing for the magnetic field Stoke's function: not here anymore (ratios is set to zero)

Getting the new dt

Now that iterations have converged, NSCool analyzes the process and prepares for the next time step.

NSCool tries to increase the time step dt (=dtime variable) as:
dtime → scale_dt∗dtime

Factors controlling scale_dt:

1) If $\mathcal{T}_i$ differs too much from $\mathcal{T}_i^{\rm old}$, scale_dt is shortened. This uses

$$\text{max\_dtemp} \equiv \mathrm{Max}_{i=1,3,5,\ldots}\left(\frac{|\mathcal{T}_i - \mathcal{T}_i^{\rm old}|}{\mathcal{T}_i^{\rm old}}\right)$$

2) If the resulting scale_dt is too small, i.e., $\mathcal{T}_i$ differs way too much from $\mathcal{T}_i^{\rm old}$, the time step is recalculated with a shorted dtime.

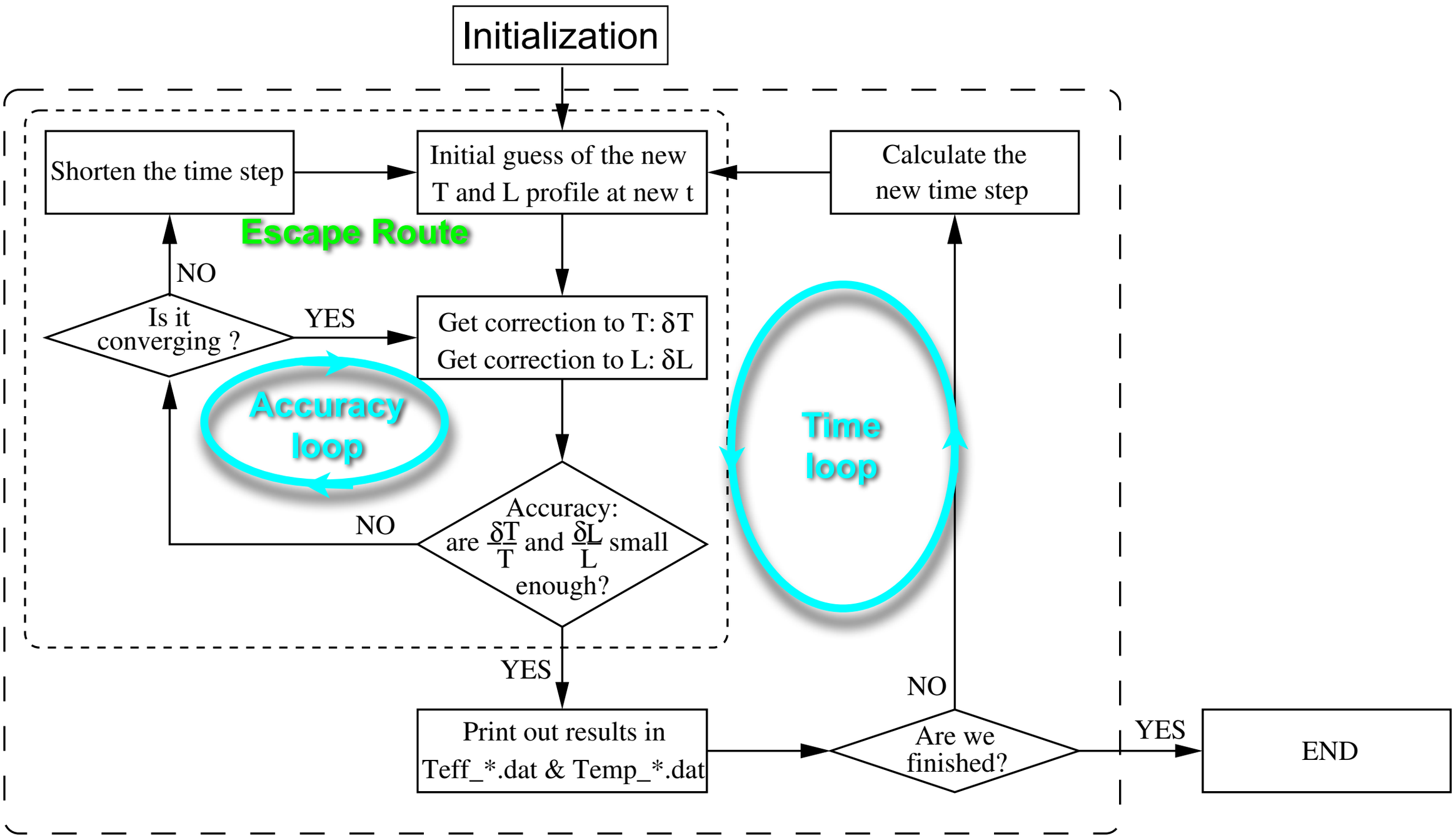
3) If finding the solutions required more than the desired number of iterations, scale_dt is also reduced.

NSCool.f

```
c ***********************************************************************
c ******* PREPARATION TO CALCULATE THE NEW TIME STEP  ****************
c ***********************************************************************
c
c This is a delicate part, based on experience and many trials and errors.
c It works pretty well, so avoid changing it !
c
c PHILOSOPHY OF TIME STEP CONTROL:
c
c (Time of step just finished is "time+dtime", not just time !)
c The new "dtime" will be "scale_dt*dtime" with "scale_dt" calculated below.
c Allows for 2 different "scale_dt": at early time, while relaxing from initial
c conditions, accuracy is not important and one can allow for larger timestep:
c "scale_dt0" and "scale_dt1" are read from the file
c NUM_PARAM.dat in NSCool_READ.inc.f
c and are the maximum allowed relative increase in "dtime"
c ***********************************************************************
-:** NSCool.f      67% (872,0)    (Fortran)
```

Escape Route: 3rd & Last Exit

Flow diagram of NSCool (bigger)



Up-date $\mathcal{T}$ & $\mathcal{L}$

Iterations have converged: ntemp(i) and nlum(i) are the solution $\mathcal{T}_i$ and $\mathcal{L}_i$. They are copied to temp(i) and lum(i) so that they become the $\mathcal{T}_i^{\text{old}}$ and $\mathcal{L}_i^{\text{old}}$ at next time step.

[The variables "osomething" are so defined that, at next time step ,they will refer to two time steps back: they will be used to guess the initial profiles $\mathcal{T}_i^{(k=0)}$ and $\mathcal{L}_i^{(k=0)}$ by extrapolating.]

NSCool.f

```
c ***********************************************************************
c ***** End of iterations
c ***********************************************************************

      do 171 i=1,imax,2
       otemp(i)=temp(i)
       temp(i)=ntemp(i)
       orrho(i)=rrho(i)
       orad(i)=rad(i)
       obar(i)=bar(i)
171   continue
       do 172 i=2,imax-1,2
       olum(i)=lum(i)
       lum(i)=nlum(i)
       orrho(i)=rrho(i)
       orad(i)=rad(i)
       obar(i)=bar(i)
172   continue
-:** NSCool.f     74% (963,0)   (Fortran)
```

NSCool.f

```
c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c
c Stuff below, till the next +++++ line is only informative and not
c used in the calculations.
-:** NSCool.f     77% (996,0)   (Fortran)

c ***********************************************************************
c ***** Calculate the neutrino luminosity and heating: ****************
c ***********************************************************************
-:** NSCool.f     79% (1038,0)   (Fortran)

c ***** CALCULATE THE INTEGRATED NEUTRINO LUMINOSITIES: ***************
c     Note: lnu_tot, calculated from qnu(i), is the garanteed total
c     neutrino luminosity. The other ones are only informative.
-:** NSCool.f     81% (1057,0)   (Fortran)

c ***** CALCULATE THE INTEGRATED SPECIFIC HEATS: **********************
c     cv_tot_all, calculated from cv(i), is the garanteed total
c     specific heat. The other ones are only informative.
-:** NSCool.f     82% (1078,0)   (Fortran)

c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-:** NSCool.f     85% (1117,0)   (Fortran)
```

The following sections calculate a bunch of things for information purpose.

That's all Folks !

Print out results in the files
"`Teff_*.dat`" and "`Temp_*.dat`"
[all done in file `NSCool_PRINT.inc.f`]

Update the `time` variable

NSCool.f

```
c ***********************************************************************
c ******************** print out results ****************************
c ***********************************************************************

      INCLUDE 'NSCool_PRINT.inc.f'
```

-:** NSCool.f 86% (1121,0) (Fortran)

```
c ***********************************************************************
c ****************  CALCULATE THE NEW TIME STEP  **********************
c ***********************************************************************
      time=time+dtime
      odtime=dtime
      dtime=min(scale_dt*dtime,dtlimit)
```

-:** NSCool.f 91% (1197,0) (Fortran)

Follow two sections to control `dtime` in case of accretion: more on this later !

NSCool.f

```
c ***********************************************************************

      if (time/year.ge.timemax) goto 9998
      if ((sign_l*teffective).lt.tempmin) goto 9998

c ***********************************************************************
9999  continue     ! This is the `end do' for the main time integration
c ***********************************************************************

c ***********************************************************************
9998  continue     ! Jump here if time > timemax
c ***********************************************************************
```

-:-- NSCool.f 97% (1286,0) (Fortran)