

Guacho 3D  
V1.2

Generated by Doxygen 1.8.9.1

Thu Mar 10 2016 12:30:11



# Contents

<b>1</b>	<b>GUACHO-3D Documentation</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	release.notes . . . . .	1
1.3	requirements . . . . .	1
<b>2</b>	<b>Modules Index</b>	<b>3</b>
2.1	Modules List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Module Documentation</b>	<b>7</b>
4.1	boundaries Module Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.1.2	Function/Subroutine Documentation . . . . .	7
4.1.2.1	boundaryi . . . . .	7
4.1.2.2	boundaryii . . . . .	7
4.2	chemistry Module Reference . . . . .	7
4.2.1	Detailed Description . . . . .	8
4.2.2	Function/Subroutine Documentation . . . . .	8
4.2.2.1	chemstep . . . . .	8
4.2.2.2	update_chem . . . . .	8
4.3	coldens_utilities Module Reference . . . . .	9
4.3.1	Detailed Description . . . . .	9
4.3.2	Function/Subroutine Documentation . . . . .	9
4.3.2.1	fill_map . . . . .	9
4.3.2.2	getxyz . . . . .	10
4.3.2.3	init_coldens . . . . .	10
4.3.2.4	read_data . . . . .	10
4.3.2.5	rotation_x . . . . .	11
4.3.2.6	rotation_y . . . . .	11
4.3.2.7	rotation_z . . . . .	11

4.3.2.8	write_header	12
4.3.2.9	write_map	12
4.4	constants Module Reference	12
4.5	cooling_chi Module Reference	13
4.5.1	Detailed Description	13
4.5.2	Function/Subroutine Documentation	14
4.5.2.1	coolchi	14
4.5.2.2	coolingchi	15
4.5.2.3	read_table	15
4.6	cooling_dmc Module Reference	15
4.6.1	Detailed Description	16
4.6.2	Function/Subroutine Documentation	16
4.6.2.1	cooldmc	16
4.6.2.2	coolingdmc	16
4.6.2.3	read_table	16
4.7	cooling_h Module Reference	16
4.7.1	Detailed Description	17
4.7.2	Function/Subroutine Documentation	17
4.7.2.1	aloss	17
4.7.2.2	alpha	17
4.7.2.3	alpha1	18
4.7.2.4	atomic	18
4.7.2.5	betah	18
4.7.2.6	colf	19
4.7.2.7	coolingh	19
4.8	difrad Module Reference	19
4.8.1	Detailed Description	20
4.8.2	Function/Subroutine Documentation	20
4.8.2.1	diffuse_rad	20
4.8.2.2	emdiff	21
4.8.2.3	init_rand	21
4.8.2.4	photons	21
4.8.2.5	progress	22
4.8.2.6	radbounds	22
4.8.2.7	random_versor	22
4.8.2.8	starsource	22
4.9	globals Module Reference	23
4.9.1	Detailed Description	24
4.10	h_alpha_utilities Module Reference	24
4.10.1	Detailed Description	24

4.10.2	Function/Subroutine Documentation	25
4.10.2.1	fill_map	25
4.10.2.2	getxyz	26
4.10.2.3	init_ha	27
4.10.2.4	read_data	27
4.10.2.5	rotation_x	27
4.10.2.6	rotation_y	27
4.10.2.7	rotation_z	28
4.10.2.8	write_ha	28
4.10.2.9	write_rg	28
4.11	hll Module Reference	28
4.11.1	Detailed Description	29
4.11.2	Function/Subroutine Documentation	29
4.11.2.1	hllfluxes	29
4.11.2.2	prim2fhll	29
4.12	hllc Module Reference	30
4.12.1	Detailed Description	30
4.12.2	Function/Subroutine Documentation	30
4.12.2.1	hllcfluxes	30
4.12.2.2	prim2fhllc	31
4.13	hlld Module Reference	31
4.13.1	Detailed Description	32
4.13.2	Function/Subroutine Documentation	32
4.13.2.1	hlldfluxes	32
4.13.2.2	prim2fhlld	32
4.14	hlle Module Reference	33
4.14.1	Detailed Description	33
4.14.2	Function/Subroutine Documentation	33
4.14.2.1	hllefluxes	33
4.14.2.2	prim2fhlle	34
4.15	hydro_core Module Reference	34
4.15.1	Detailed Description	35
4.15.2	Function/Subroutine Documentation	35
4.15.2.1	calcpri	35
4.15.2.2	cfast	36
4.15.2.3	cfastx	36
4.15.2.4	csound	36
4.15.2.5	get_timestep	36
4.15.2.6	limiter	37
4.15.2.7	prim2f	37

4.15.2.8	prim2u	38
4.15.2.9	swapy	39
4.15.2.10	swapz	39
4.15.2.11	u2prim	39
4.16	hydro_solver Module Reference	39
4.16.1	Detailed Description	40
4.16.2	Function/Subroutine Documentation	40
4.16.2.1	step	40
4.16.2.2	tstep	40
4.16.2.3	viscosity	41
4.17	init Module Reference	42
4.17.1	Detailed Description	42
4.17.2	Function/Subroutine Documentation	42
4.17.2.1	initflow	42
4.17.2.2	initmain	42
4.18	linear_system Module Reference	43
4.18.1	Detailed Description	43
4.18.2	Function/Subroutine Documentation	43
4.18.2.1	linsys	43
4.18.2.2	lubksb	44
4.18.2.3	ludcmp	44
4.19	lyman_alpha_utilities Module Reference	44
4.19.1	Detailed Description	45
4.19.2	Function/Subroutine Documentation	45
4.19.2.1	fill_map	45
4.19.2.2	getxyz	46
4.19.2.3	init_la	46
4.19.2.4	phigauss	47
4.19.2.5	read_data	47
4.19.2.6	rotation_x	47
4.19.2.7	rotation_y	47
4.19.2.8	rotation_z	47
4.19.2.9	write_la	48
4.20	out_bin_module Module Reference	48
4.20.1	Detailed Description	48
4.20.2	Function/Subroutine Documentation	48
4.20.2.1	write_bin	48
4.20.2.2	write_header	49
4.21	out_silo_module Module Reference	49
4.21.1	Detailed Description	49

4.21.2	Function/Subroutine Documentation	49
4.21.2.1	outputsilo	49
4.21.2.2	writeblocks	50
4.21.2.3	writemaster	50
4.22	out_vtk_module Module Reference	50
4.22.1	Detailed Description	50
4.22.2	Function/Subroutine Documentation	51
4.22.2.1	write_vtk	51
4.23	output Module Reference	52
4.23.1	Detailed Description	52
4.23.2	Function/Subroutine Documentation	52
4.23.2.1	write_output	52
4.24	sources Module Reference	53
4.24.1	Detailed Description	53
4.24.2	Function/Subroutine Documentation	53
4.24.2.1	divbcorr_source	53
4.24.2.2	divergence_b	54
4.24.2.3	getpos	54
4.24.2.4	grav_source	54
4.24.2.5	radpress_source	55
4.24.2.6	source	55
4.25	thermal_cond Module Reference	56
4.25.1	Detailed Description	57
4.25.2	Function/Subroutine Documentation	57
4.25.2.1	get_dt_cond	57
4.25.2.2	heatfluxes	57
4.25.2.3	init_thermal_cond	58
4.25.2.4	ksp	58
4.25.2.5	ksp_parl	58
4.25.2.6	ksp_perp	58
4.25.2.7	mhd_heatfluxes	58
4.25.2.8	progress	59
4.25.2.9	st_steps	59
4.25.2.10	substep	59
4.25.2.11	superstep	60
4.25.2.12	thermal_bounds	60
4.25.2.13	thermal_conduction	60
<b>5</b>	<b>File Documentation</b>	<b>61</b>
5.1	doc/mainpage.h File Reference	61

5.2	src/boundaries.f90 File Reference	61
5.2.1	Detailed Description	61
5.3	src/chemistry.f90 File Reference	61
5.3.1	Detailed Description	62
5.4	src/coldens.f90 File Reference	62
5.4.1	Detailed Description	63
5.4.2	Function/Subroutine Documentation	63
5.4.2.1	coldens	63
5.5	src/constants.f90 File Reference	63
5.5.1	Detailed Description	64
5.6	src/cooling_chi.f90 File Reference	65
5.6.1	Detailed Description	65
5.7	src/cooling_dmc.f90 File Reference	65
5.7.1	Detailed Description	66
5.8	src/cooling_h.f90 File Reference	66
5.8.1	Detailed Description	66
5.9	src/difrad.f90 File Reference	67
5.9.1	Detailed Description	68
5.10	src/globals.f90 File Reference	68
5.10.1	Detailed Description	69
5.11	src/h_alpha_proj.f90 File Reference	69
5.11.1	Detailed Description	70
5.11.2	Function/Subroutine Documentation	70
5.11.2.1	h_alpha_proj	70
5.12	src/hll.f90 File Reference	70
5.12.1	Detailed Description	71
5.13	src/hllc.f90 File Reference	71
5.13.1	Detailed Description	71
5.14	src/hlld.f90 File Reference	71
5.14.1	Detailed Description	72
5.15	src/hlle.f90 File Reference	72
5.15.1	Detailed Description	72
5.16	src/hydro_core.f90 File Reference	72
5.16.1	Detailed Description	73
5.17	src/hydro_solver.f90 File Reference	73
5.17.1	Detailed Description	74
5.18	src/init.f90 File Reference	74
5.18.1	Detailed Description	74
5.19	src/linear_system.f90 File Reference	74
5.19.1	Detailed Description	75



5.20	src/lyman_alpha_tau.f90 File Reference	75
5.20.1	Detailed Description	76
5.20.2	Function/Subroutine Documentation	76
5.20.2.1	lyman_alpha_tau	76
5.21	src/main.f90 File Reference	76
5.21.1	Detailed Description	77
5.22	src/Out_BIN_Module.f90 File Reference	77
5.22.1	Detailed Description	77
5.23	src/Out_Silo_Module.f90 File Reference	78
5.23.1	Detailed Description	78
5.24	src/Out_VTK_Module.f90 File Reference	78
5.24.1	Detailed Description	79
5.25	src/output.f90 File Reference	79
5.25.1	Detailed Description	79
5.26	src/sources.f90 File Reference	79
5.26.1	Detailed Description	80
5.27	src/thermal_cond.f90 File Reference	80
5.27.1	Detailed Description	81
<b>Index</b>		<b>83</b>



# Chapter 1

## GUACHO-3D Documentation

### Authors

Alejandro Esquivel et al.

### 1.1 Introduction

Documentation of the Guacho code

### 1.2 release.notes

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/gpl.html>

### 1.3 requirements

Fortran 90/95 compiler with C preprocessor, Message Passing Interface (optional), gmake.



## Chapter 2

# Modules Index

### 2.1 Modules List

Here is a list of all documented modules with brief descriptions:

<a href="#">boundaries</a>		
	Boundary conditions . . . . .	7
<a href="#">chemistry</a>		
	Chemistry module . . . . .	7
<a href="#">coldens_utilities</a>		
	Column density projection . . . . .	9
<a href="#">constants</a>		
	Module containing physical and asronomical constants . . . . .	12
<a href="#">cooling_chi</a>		
	Cooling module with CHIANTI generated cooling curves . . . . .	13
<a href="#">cooling_dmc</a>		
	Cooling module with Dalgarno McCray coronal cooling curve . . . . .	15
<a href="#">cooling_h</a>		
	Cooling with parametrized cooling and H rate equation . . . . .	16
<a href="#">difrad</a>		
	Ray tracing Radiative Trasnport . . . . .	19
<a href="#">globals</a>		
	Module containing global variables . . . . .	23
<a href="#">h_alpha_utilities</a>		
	H alpha projection . . . . .	24
<a href="#">hll</a>		
	HLL approximate Riemann solver module . . . . .	28
<a href="#">hllc</a>		
	HLLC approximate Riemann solver module . . . . .	30
<a href="#">hlld</a>		
	HLLD approximate Riemann solver module . . . . .	31
<a href="#">hlle</a>		
	HLL E approximate Riemann solver module . . . . .	33
<a href="#">hydro_core</a>		
	Basic hydro (and MHD) subroutines utilities . . . . .	34
<a href="#">hydro_solver</a>		
	Advances the simulation one timestep . . . . .	39
<a href="#">init</a>		
	Guacho-3D initialization . . . . .	42
<a href="#">linear_system</a>		
	Linear system inversion module . . . . .	43
<a href="#">lyman_alpha_utilities</a>		
	Lyman_alpha_utilities . . . . .	44

<a href="#">out_bin_module</a>	
Output in BIN format . . . . .	48
<a href="#">out_silo_module</a>	
Output in Silo (+HDF5) Format . . . . .	49
<a href="#">out_vtk_module</a>	
Output in VTK format . . . . .	50
<a href="#">output</a>	
Writes output . . . . .	52
<a href="#">sources</a>	
Adds source terms . . . . .	53
<a href="#">thermal_cond</a>	
Adds thermal conduction . . . . .	56

## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

doc/ <a href="#">mainpage.h</a>	
Webpage frontend . . . . .	61
src/ <a href="#">boundaries.f90</a>	
Boundary conditions . . . . .	61
src/ <a href="#">chemistry.f90</a>	
Chemistry module . . . . .	61
src/ <a href="#">coldens.f90</a>	
Column density projection . . . . .	62
src/ <a href="#">constants.f90</a>	
Constants module . . . . .	63
src/ <a href="#">cooling_chi.f90</a>	
Cooling module with CHIANTI generated cooling curves . . . . .	65
src/ <a href="#">cooling_dmc.f90</a>	
Cooling module with Dlgarno Mac Cray coronal cooling curve . . . . .	65
src/ <a href="#">cooling_h.f90</a>	
Cooling with hydrogen rate parametrized cooling . . . . .	66
src/ <a href="#">difrad.f90</a>	
Diffuse radiation module . . . . .	67
src/ <a href="#">globals.f90</a>	
Global variables . . . . .	68
src/ <a href="#">h_alpha_proj.f90</a>	
H alpha projection . . . . .	69
src/ <a href="#">hll.f90</a>	
HLL approximate Riemann solver module . . . . .	70
src/ <a href="#">hllc.f90</a>	
HLLC approximate Riemann solver module . . . . .	71
src/ <a href="#">hlld.f90</a>	
HLLD approximate Riemann solver module . . . . .	71
src/ <a href="#">hlle.f90</a>	
HLL E approximate Riemann solver module . . . . .	72
src/ <a href="#">hydro_core.f90</a>	
Hydrodynamical and Magnetohydrodynamocal basic module . . . . .	72
src/ <a href="#">hydro_solver.f90</a>	
Hydrodynamical and Magnetohydrodynamocal solver module . . . . .	73
src/ <a href="#">init.f90</a>	
Guacho-3D initialization module . . . . .	74
src/ <a href="#">linear_system.f90</a>	
Linear system inversion module . . . . .	74

src/lyman_alpha_tau.f90	
Lyman_alpha_utilities . . . . .	75
src/main.f90	
Guacho-3D main program . . . . .	76
src/Out_BIN_Module.f90	
Output in BIN Format . . . . .	77
src/Out_Silo_Module.f90	
Output in Silo Format . . . . .	78
src/Out_VTK_Module.f90	
Output in VTK Format . . . . .	78
src/output.f90	
Writes Output . . . . .	79
src/sources.f90	
Adds source terms . . . . .	79
src/thermal_cond.f90	
Thermal conduction module . . . . .	80



## Chapter 4

# Module Documentation

### 4.1 boundaries Module Reference

Boundary conditions.

#### Functions/Subroutines

- subroutine `boundaryi` ()  
*Boundary conditions for 1st order half timestep.*
- subroutine `boundaryii` ()  
*Boundary conditions for 2nd order half timestep.*

#### 4.1.1 Detailed Description

Sets boundary conditions, the type of boundaries is set in the Makefile

#### 4.1.2 Function/Subroutine Documentation

##### 4.1.2.1 subroutine `boundaries::boundaryi` ( )

Boundary conditions for 1st order half timestep

The conditions only are imposed at the innermost ghost cell, on the u (unstepped) variables

Definition at line 48 of file `boundaries.f90`.

##### 4.1.2.2 subroutine `boundaries::boundaryii` ( )

Boundary conditions for 2nd order half timestep

The conditions only are imposed in two ghost cells on the up (stepped) variables

Definition at line 259 of file `boundaries.f90`.

### 4.2 chemistry Module Reference

chemistry module

## Functions/Subroutines

- subroutine `update_chem` ()  
*Advances the chemistry network.*
- subroutine `chemstep` (y, y0, T, deltt)  
*Advances the chemistry network in one cell.*

### 4.2.1 Detailed Description

module to solve the chemical/ionic network.

### 4.2.2 Function/Subroutine Documentation

4.2.2.1 subroutine `chemistry::chemstep` ( real (kind=8), dimension(n\_spec), intent(inout) y, real (kind=8), dimension(n\_elem), intent(in) y0, real (kind=8), intent(in) T, real (kind=8), intent(in) deltt )

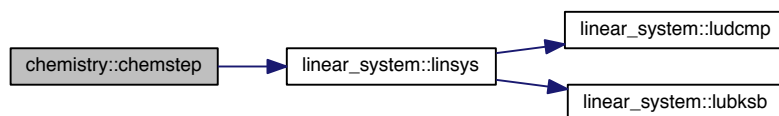
Advances the chemistry network on the in one cell

#### Parameters

<i>real</i>	[inout] y(n_spec) : number densities of the species to be updated by the chemistry
<i>real</i>	[in] y[n_elem] : total number density of each of the elements involved in the reactions
<i>real</i>	[in] T : Temperature [K]
<i>real</i>	[in] deltt : time interval (from the hydro, in seconds)

Definition at line 91 of file `chemistry.f90`.

Here is the call graph for this function:

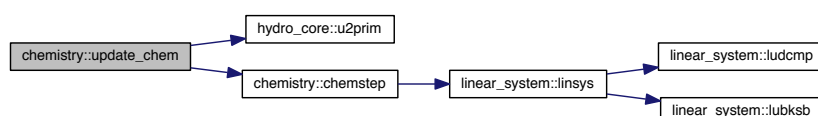


4.2.2.2 subroutine `chemistry::update_chem` ( )

Advances the chemistry network on the entire domain (except ghost cells), updates primitives and conserved variables in globals

Definition at line 43 of file `chemistry.f90`.

Here is the call graph for this function:



## 4.3 coldens\_utilities Module Reference

Column density projection.

### Functions/Subroutines

- subroutine `init_coldens` ()  
*Initializes data.*
- subroutine `read_data` (u, itprint, filepath)  
*reads data from file*
- subroutine `getxyz` (i, j, k, x, y, z)  
*gets position of a cell*
- subroutine `rotation_x` (theta, x, y, z, xn, yn, zn)  
*Rotation around the X axis.*
- subroutine `rotation_y` (theta, x, y, z, xn, yn, zn)  
*Rotation around the Y axis.*
- subroutine `rotation_z` (theta, x, y, z, xn, yn, zn)  
*Rotation around the Z axis.*
- subroutine `fill_map` (nxmap, nymap, u, map, dxT, dyT, theta\_x, theta\_y, theta\_z)  
*Fill target map.*
- subroutine `write_header` (unit, nx, ny)  
*Writes header.*
- subroutine `write_map` (fileout, nxmap, nymap, map)  
*Writes projection to file.*

### 4.3.1 Detailed Description

Utilities to compute a column density map

### 4.3.2 Function/Subroutine Documentation

- 4.3.2.1 subroutine `coldens_utilities::fill_map` ( integer, intent(in) *nxmap*, integer, intent(in) *nymap*, real, dimension(neq,nxmin:nxmax,nymmin:nymax,nzmin:nzmax), intent(in) *u*, real, dimension(nxmap,nymap), intent(out) *map*, real, intent(in) *dxT*, real, intent(in) *dyT*, real, intent(in) *theta\_x*, real, intent(in) *theta\_y*, real, intent(in) *theta\_z* )

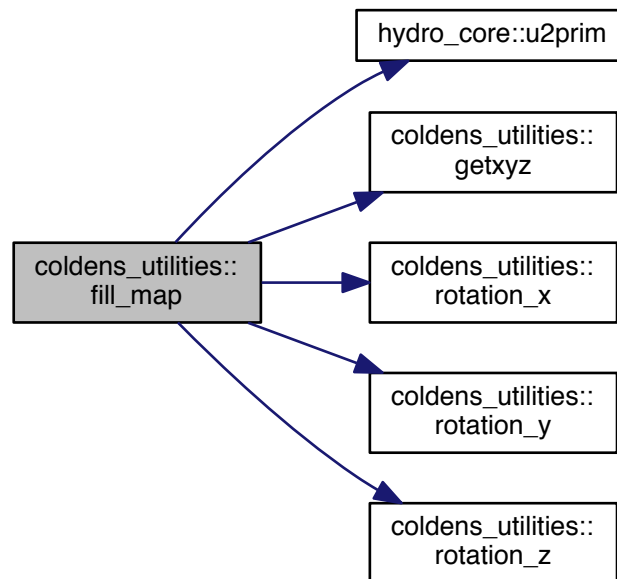
Fills the target map of one MPI block

#### Parameters

<i>integer</i>	[in] <i>nxmap</i> : Number of X cells in target
<i>integer</i>	[in] <i>nymap</i> : Number of Y cells in target
<i>real</i>	[in] <i>u</i> (neq,nxmin:nxmax,nymmin:nymax,nzmin:nzmax) : conserved variables
<i>real</i>	[out] <i>map</i> (nxmap,nymap) : Target map
<i>real</i>	[in] <i>dxT</i> : target pixel width
<i>real</i>	[in] <i>dyT</i> : target pixel height
<i>real</i>	[in] <i>thetax</i> : Rotation around X
<i>real</i>	[in] <i>thetay</i> : Rotation around Y
<i>real</i>	[in] <i>thetaz</i> : Rotation around Z

Definition at line 307 of file `coldens.f90`.

Here is the call graph for this function:



**4.3.2.2** subroutine `coldens_utilities::getxyz` ( *integer*, intent(in) *i*, *integer*, intent(in) *j*, *integer*, intent(in) *k*, *real*, intent(out) *x*, *real*, intent(out) *y*, *real*, intent(out) *z* )

Returns the position and spherical radius calculated with respect to the center of the grid

#### Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the x direction
<i>integer</i>	[in] <i>j</i> : cell index in the y direction
<i>integer</i>	[in] <i>k</i> : cell index in the z direction
<i>real</i>	[in] <i>x</i> : x position in the grid
<i>real</i>	[in] <i>y</i> : y position in the grid
<i>real</i>	[in] <i>z</i> : z position in the grid

Definition at line 209 of file `coldens.f90`.

**4.3.2.3** subroutine `coldens_utilities::init_coldens` ( )

Initializes data, MPI and other stuff

Definition at line 36 of file `coldens.f90`.

**4.3.2.4** subroutine `coldens_utilities::read_data` ( *real*, dimension(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax), intent(out) *u*, *integer*, intent(in) *itprint*, *character* (len=128), intent(in) *filepath* )

reads data from file

## Parameters

<i>real</i>	[out] u(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax) : conserved variables
<i>integer</i>	[in] itprint : number of output
<i>string</i>	[in] filepath : path where the output is

Definition at line 135 of file coldens.f90.

**4.3.2.5** subroutine coldens\_utilities::rotation\_x ( *real*, intent(in) *theta*, *real*, intent(in) *x*, *real*, intent(in) *y*, *real*, intent(in) *z*, *real*, intent(out) *xn*, *real*, intent(out) *yn*, *real*, intent(out) *zn* )

Does a rotation around the x axis

## Parameters

<i>real</i>	[in], <i>theta</i> : Angle of rotation (in radians)
<i>real</i>	[in], <i>x</i> : original x position in the grid
<i>real</i>	[in], <i>y</i> : original y position in the grid
<i>real</i>	[in], <i>x</i> : original z position in the grid
<i>real</i>	[out], <i>x</i> : final x position in the grid
<i>real</i>	[out], <i>y</i> : final y position in the grid
<i>real</i>	[out], <i>x</i> : final z position in the grid

Definition at line 235 of file coldens.f90.

**4.3.2.6** subroutine coldens\_utilities::rotation\_y ( *real*, intent(in) *theta*, *real*, intent(in) *x*, *real*, intent(in) *y*, *real*, intent(in) *z*, *real*, intent(out) *xn*, *real*, intent(out) *yn*, *real*, intent(out) *zn* )

Does a rotation around the x axis

## Parameters

<i>real</i>	[in], <i>theta</i> : Angle of rotation (in radians)
<i>real</i>	[in], <i>x</i> : original x position in the grid
<i>real</i>	[in], <i>y</i> : original y position in the grid
<i>real</i>	[in], <i>x</i> : original z position in the grid
<i>real</i>	[out], <i>x</i> : final x position in the grid
<i>real</i>	[out], <i>y</i> : final y position in the grid
<i>real</i>	[out], <i>x</i> : final z position in the grid

Definition at line 259 of file coldens.f90.

**4.3.2.7** subroutine coldens\_utilities::rotation\_z ( *real*, intent(in) *theta*, *real*, intent(in) *x*, *real*, intent(in) *y*, *real*, intent(in) *z*, *real*, intent(out) *xn*, *real*, intent(out) *yn*, *real*, intent(out) *zn* )

Does a rotation around the x axis

## Parameters

<i>real</i>	[in], <i>theta</i> : Angle of rotation (in radians)
<i>real</i>	[in], <i>x</i> : original x position in the grid
<i>real</i>	[in], <i>y</i> : original y position in the grid
<i>real</i>	[in], <i>x</i> : original z position in the grid
<i>real</i>	[out], <i>x</i> : final x position in the grid
<i>real</i>	[out], <i>y</i> : final y position in the grid

<i>real</i>	[out], x : final z position in the grid
-------------	---

Definition at line 281 of file coldens.f90.

4.3.2.8 subroutine coldens\_utilities::write\_header ( integer, intent(in) *unit*, integer, intent(in) *nx*, integer, intent(in) *ny* )

Writes header for binary input

Parameters

<i>integer</i>	[in] <i>unit</i> : number of logical unit
----------------	---

Definition at line 359 of file coldens.f90.

4.3.2.9 subroutine coldens\_utilities::write\_map ( character (len=128), intent(in) *fileout*, integer, intent(in) *nxmap*, integer, intent(in) *nymap*, real, dimension(nxmap,nymap), intent(in) *map* )

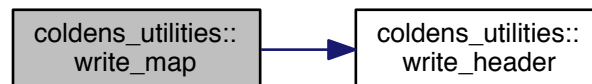
Writes projection to file

Parameters

<i>integer</i>	[in] <i>itprint</i> : number of output
<i>string</i>	[in] <i>fileout</i> : file where to write
<i>integer</i>	[in] <i>nxmap</i> : Number of X cells in target
<i>integer</i>	[in] <i>nymap</i> : Number of Y cells in target
<i>real</i>	[in] <i>map</i> (nxmap,nymap) : Target map

Definition at line 433 of file coldens.f90.

Here is the call graph for this function:



## 4.4 constants Module Reference

Module containing physical and asronomical constants.

### Variables

- real, parameter *pi* =acos(-1.)  
 $\pi$
- real, parameter *amh* =1.66e-24  
*hydrogen mass*
- real, parameter *kb* =1.38e-16  
*Boltzmann constant (cgs)*
- real, parameter *rg* =8.3145e7  
*Gas constant (cgs)*

- real, parameter `ggrav` =6.67259e-8  
*Gravitational constant (cgs)*
- real, parameter `clight` =2.99E10  
*speed of light in vacuum (cgs)*
- real, parameter `msun` =1.99E33  
*solar radius (cgs)*
- real, parameter `rsun` =6.955e10  
*solar mass (cgs)*
- real, parameter `mjup` =1.898E30  
*Jupiter mass (cgs)*
- real, parameter `rjup` =7.1492E9  
*Jupiter radius (cgs)*
- real, parameter `au` =1.496e13  
*1AU in cm*
- real, parameter `pc` =3.0857E18  
*1pc in cm*
- real, parameter `kpc` =3.0857E21  
*1Kpc in cm*
- real, parameter `hr` =3600.  
*1hr in seconds*
- real, parameter `day` =86400.  
*1day in seconds*
- real, parameter `yr` =3.1536E7  
*1yr in seconds*
- real, parameter `myr` =3.1536E13  
*1Myr in seconds*

## 4.5 cooling\_chi Module Reference

Cooling module with CHIANTI generated cooling curves.

### Functions/Subroutines

- subroutine `read_table` ()  
*Reads the cooling curve table.*
- real(kind=8) function `coolchi` (T)  
*Returns the cooling coefficient interpolating the table.*
- subroutine `coolingchi` ()  
*High level wrapper to apply cooling with CHIANTI tables.*

### Variables

- real(kind=8), dimension(2, 41) `cooltab`

#### 4.5.1 Detailed Description

Cooling module with CHIANTI generated cooling curves

The location of the tables is assumed to be in `src/CHIANTIlib/coolingCHIANTI.tab`

## 4.5.2 Function/Subroutine Documentation

4.5.2.1 `real (kind=8) function cooling_chi::coolchi ( real, intent(in)  $T$  )`



## Parameters

<i>real</i>	[in] T : Temperature K
-------------	------------------------

Definition at line 75 of file cooling\_chi.f90.

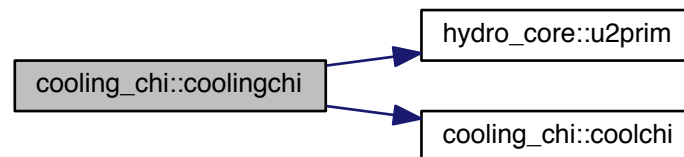
## 4.5.2.2 subroutine cooling\_chi::coolingchi ( )

High level wrapper to apply cooling with CHIANTI tables

cooling is applied in the entire domain and updates both the conserved and primitive variables

Definition at line 102 of file cooling\_chi.f90.

Here is the call graph for this function:



## 4.5.2.3 subroutine cooling\_chi::read\_table ( )

Reads the cooling curve table generated by CHUANTI, the location is assumed in /src/CHIANTIlib/coolingCHIANTI.tab

Definition at line 44 of file cooling\_chi.f90.

## 4.6 cooling\_dmc Module Reference

Cooling module with Dalgarno McCray coronal cooling curve.

## Functions/Subroutines

- subroutine `read_table` ()  
*Reads the cooling curve table.*
- real(kind=8) function `cooldmc` (T)  
*Returns the cooling coefficient interpolating the table.*
- subroutine `coolingdmc` ()  
*High level wrapper to apply cooling with DMC table.*

## Variables

- real(kind=8), dimension(2, 41) `cooltab`

### 4.6.1 Detailed Description

Cooling module with Dalgarno McCray coronal cooling curve

The location of the tables is assumed to be in src/DMClib/coolingDMC.tab, it is read by init subroutine

### 4.6.2 Function/Subroutine Documentation

#### 4.6.2.1 `real (kind=8) function cooling_dmc::cooldmc ( real, intent(in) T )`

Parameters

<i>real</i>	[in] T : Temperature K
-------------	------------------------

Definition at line 77 of file cooling\_dmc.f90.

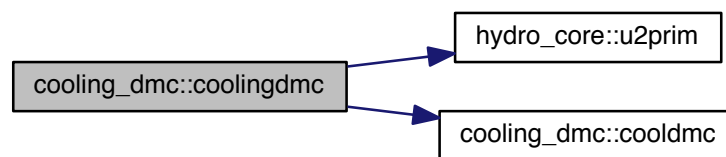
#### 4.6.2.2 `subroutine cooling_dmc::coolingdmc ( )`

High level wrapper to apply cooling with DMC table

cooling is applied in the entire domain and updates both the conserved and primitive variables

Definition at line 103 of file cooling\_dmc.f90.

Here is the call graph for this function:



#### 4.6.2.3 `subroutine cooling_dmc::read_table ( )`

Reads the Dalgarno McCray cooling curve the location is assumed in src/DMClib/coolingDMC.tab, it is read by init subroutine

Definition at line 45 of file cooling\_dmc.f90.

## 4.7 cooling\_h Module Reference

Cooling with parametrized cooling and H rate equation.

### Functions/Subroutines

- subroutine `coolingh` ()  
*High level wrapper to apply cooling.*
- `real(kind=8) function alpha` (T)  
*calculates the recombination rate (case B)*

- real(kind=8) function [alpha1](#) (T)  
*calculates the recombination rate to level 1*
- real(kind=8) function [colf](#) (T)  
*calculates the collisional ionization rate*
- real(kind=8) function [betah](#) (T)  
*betaH(T)*
- real(kind=8) function [aloss](#) (X1, X2, DT, DEN, DH0, TE0)  
*Non equilibrium cooling.*
- subroutine [atomic](#) (dt, uu, tau, radphi)  
*Updates the ionization fraction and applies cooling.*

#### 4.7.1 Detailed Description

Cooling with parametrized cooling and H rate equation

#### 4.7.2 Function/Subroutine Documentation

**4.7.2.1** real(kind=8) function `cooling_h::aloss` ( real(kind=8), intent(in) X1, real(kind=8), intent(in) X2, real, intent(in) DT, real(kind=8), intent(in) DEN, real(kind=8), intent(in) DH0, real(kind=8), intent(in) TE0 )

Non-equilibrium energy loss for low temperatures considering the collisional excitation of [O I] and [O II] lines and radiative recombination of H. This cooling rate is multiplied by a factor of 7.033 so that it has the same value as the "coronal equilibrium" cooling rate at a temperature of 44770 K (at temperatures higher than this value, the equilibrium cooling rate is used). The collisional ionization of H and excitation of Lyman-alpha are computed separately, and added to the cooling rate.

##### Parameters

<i>real8</i>	[in] x1 : initial H ionization fraction
<i>real8</i>	[in] x2 : final H ionization fraction
<i>real</i>	[in] dt : timestep
<i>real8</i>	[in] den : total density of hydrogen
<i>real8</i>	[in] dh0 : density of neutral hydrogen
<i>real8</i>	[in] Te0 : Temperature

Definition at line 164 of file `cooling_h.f90`.

Here is the call graph for this function:



**4.7.2.2** real(kind=8) function `cooling_h::alpha` ( real(kind=8), intent(in) T )

calculates the recombination rate (case B)

## Parameters

<i>real8</i>	[in] T : Temperature K
--------------	------------------------

Definition at line 80 of file cooling\_h.f90.

#### 4.7.2.3 `real (kind=8) function cooling_h::alpha1 ( real (kind=8), intent(in) T )`

calculates the recombination rate to level 1

## Parameters

<i>real8</i>	[in] T : Temperature K
--------------	------------------------

Definition at line 97 of file cooling\_h.f90.

#### 4.7.2.4 `subroutine cooling_h::atomic ( real, intent(in) dt, real, dimension(neq), intent(out) uu, real, intent(in) tau, real, intent(in) radphi )`

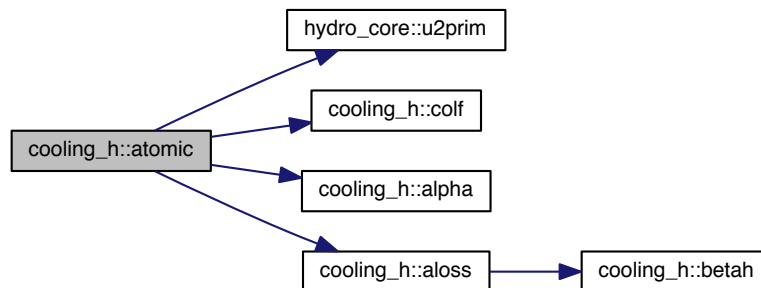
Calculates the new ionization state and energy density using a time dependent ionization calculation and an approximate time dependent cooling calculation

## Parameters

<i>real</i>	[in] dt : timestep (seconds)
<i>real</i>	[in] uu(neq) : conserved variables in one cell
<i>real</i>	[in] tau : optical depth (not in use)
<i>real</i>	[in] radphi : photoionizing rate

Definition at line 264 of file cooling\_h.f90.

Here is the call graph for this function:



#### 4.7.2.5 `real (kind=8) function cooling_h::betah ( real (kind=8), intent(in) T )`

$\beta_H(T)$

## Parameters

---

<i>real</i>	8[in] T : Temperature K
-------------	-------------------------

Definition at line 130 of file cooling\_h.f90.

#### 4.7.2.6 `real (kind=8) function cooling_h::colf ( real (kind=8), intent(in) T )`

calculates the collisional ionization rate

Parameters

<i>real8[in]</i>	T : Temperature K
------------------	-------------------

Definition at line 113 of file cooling\_h.f90.

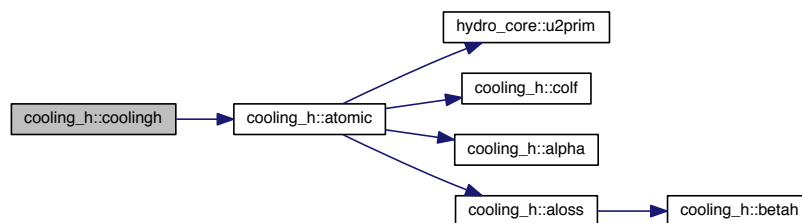
#### 4.7.2.7 `subroutine cooling_h::coolingh ( )`

High level wrapper to apply cooling

parametrized cooling curve, uses the ionization state of hydrogen and ties the O I and II to it

Definition at line 42 of file cooling\_h.f90.

Here is the call graph for this function:



## 4.8 difrad Module Reference

Ray tracing Radiative Trasnport.

### Functions/Subroutines

- subroutine `init_rand` ()  
*initializes random number generation*
- subroutine `emdiff` (emax)  
*calculates the diffuse fotoionization emissivity*
- subroutine `random_versor` (xd, yd, zd)  
*returns the 3 components of a random versor*
- subroutine `starsource` (srad, x0, y0, z0, x, y, z, xd, yd, zd)  
*Place photon packets at a "star" surface.*
- subroutine `photons` (xl0, yl0, zl0, xd, yd, zd, f)  
*Photon trajectories.*
- subroutine `radbounds` ()  
*follows the rays across MPI boundaries*
- subroutine `progress` (j, tot)

*Progress bar.*

- subroutine `diffuse_rad` ( )

*Diffuse radiation driver.*

## Variables

- real, parameter `a0` =6.3e-18

*Photoionization cross section.*

- integer, parameter `nrays` =1000000

*Number of rays.*

- real, dimension(:,:), allocatable `ph`

*Photoionizing rate.*

- real, dimension(:,:), allocatable `em`

*Photoionizing emissivity.*

- real, dimension(:,:), allocatable `photl`

*Auxiliary buffer for MPI.*

- real, dimension(:,:), allocatable `photr`

*Auxiliary buffer for MPI.*

- real, dimension(:,:), allocatable `photb`

*Auxiliary buffer for MPI.*

- real, dimension(:,:), allocatable `phott`

*Auxiliary buffer for MPI.*

- real, dimension(:,:), allocatable `photo`

*Auxiliary buffer for MPI.*

- real, dimension(:,:), allocatable `photi`

*Auxiliary buffer for MPI.*

- integer, dimension(6) `buffersize`

*Auxiliary buffer for MPI.*

### 4.8.1 Detailed Description

Ray tracing Radiative Transport

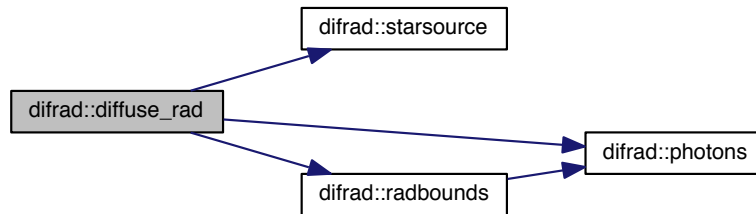
### 4.8.2 Function/Subroutine Documentation

#### 4.8.2.1 subroutine `difrad::diffuse_rad` ( )

Upper level wrapper to compute the diffuse photoionization rate

Definition at line 657 of file `difrad.f90`.

Here is the call graph for this function:



#### 4.8.2.2 subroutine `difrad::emdiff` ( *real*, intent(out) *emax* )

calculates the diffuse fotoionization emissivity in the entire domain

Parameters

<i>real</i>	[out] <i>emax</i> : maximum emissivity in the entire grid
-------------	---

Definition at line 98 of file `difrad.f90`.

Here is the call graph for this function:



#### 4.8.2.3 subroutine `difrad::init_rand` ( )

initializes random number generation

Definition at line 56 of file `difrad.f90`.

#### 4.8.2.4 subroutine `difrad::photons` ( *real*, intent(in) *xI0*, *real*, intent(in) *yI0*, *real*, intent(in) *zI0*, *real*, intent(in) *xd*, *real*, intent(in) *yd*, *real*, intent(in) *zd*, *real*, intent(inout) *f* )

Launches a photon from cell (*xc*,*yc*,*zc*) in the (*xd*,*yd*,*zd*) direction, with *f* and ionizing photons, and updates the photoionizing rate

Parameters

<i>real</i>	[in] xI0 : Initial X position
<i>real</i>	[in] yI0 : Initial Y position
<i>real</i>	[in] zI0 : Initial Z position
<i>real</i>	[in] xd : Direction in X
<i>real</i>	[in] yd : Direction in Y
<i>real</i>	[in] zd : Direction in Z
<i>real</i>	[in] f : NUmber of photoionizong photons

Definition at line 252 of file difrad.f90.

#### 4.8.2.5 subroutine difrad::progress ( integer(kind=4) j, integer(kind=4), intent(in) tot )

Progress bar (only tested with Fortran conmpiler) takes a number between 1 and tot

Parameters

<i>integer</i>	[in] j : current iteration
<i>integer</i>	[in] tot : total number of iterartions

Definition at line 635 of file difrad.f90.

#### 4.8.2.6 subroutine difrad::radbounds ( )

follows the rays across MPI boundaries

Definition at line 455 of file difrad.f90.

Here is the call graph for this function:



#### 4.8.2.7 subroutine difrad::random\_versor ( real, intent(out) xd, real, intent(out) yd, real, intent(out) zd )

returns the 3 components of a random versor (unit magnitude)

Parameters

<i>real</i>	[out] xd : x component
<i>real</i>	[out] yd : y component
<i>real</i>	[out] zd : z component

Definition at line 149 of file difrad.f90.

#### 4.8.2.8 subroutine difrad::starsource ( real, intent(in) srad, real, intent(in) x0, real, intent(in) y0, real, intent(in) z0, real, intent(out) x, real, intent(out) y, real, intent(out) z, real, intent(out) xd, real, intent(out) yd, real, intent(out) zd )

returns the random location and direction at a star surface, if the direction goes into the star, the direction is inverted



## Parameters

<i>real</i>	[in] Srad : radius of the "star"
<i>real</i>	[in] x0 : X position of the center of the star
<i>real</i>	[in] y0 : Y position of the center of the star
<i>real</i>	[in] z0 : Z position of the center of the star
<i>real</i>	[out] x : random X position at the star surface
<i>real</i>	[out] y : random Y position at the star surface
<i>real</i>	[out] z : random Z position at the star surface
<i>real</i>	[out] xd : random X direction
<i>real</i>	[out] yd : random Y direction
<i>real</i>	[out] zd : random Z direction

Definition at line 187 of file difrad.f90.

## 4.9 globals Module Reference

Module containing global variables.

## Variables

- *real*, dimension(:, :, :), allocatable **u**  
*conserved variables*
- *real*, dimension(:, :, :), allocatable **up**  
*conserved variables after 1/2 timestep*
- *real*, dimension(:, :, :), allocatable **primit**  
*primitive variables*
- *real*, dimension(:, :, :), allocatable **f**  
*X fluxes.*
- *real*, dimension(:, :, :), allocatable **g**  
*Y fluxes.*
- *real*, dimension(:, :, :), allocatable **h**  
*Z fluxes.*
- *real* **dx**  
*grid spacing in X*
- *real* **dy**  
*grid spacing in Y*
- *real* **dz**  
*grid spacing in Z*
- *integer*, dimension(0:2) **coords**  
*position of neighboring MPI blocks*
- *integer* **left**  
*MPI neighbor in the -x direction.*
- *integer* **right**  
*MPI neighbor in the +x direction.*
- *integer* **top**  
*MPI neighbor in the -y direction.*
- *integer* **bottom**  
*MPI neighbor in the +y direction.*
- *integer* **out**  
*MPI neighbor in the -z direction.*
- *integer* **in**

- *MPI neighbor in the +z direction.*
- integer `rank`  
*MPI rank.*
- integer `comm3d`  
*Cartesian MPI communicator.*
- real `time`  
*Current time.*
- real `dt_cfl`  
*Current CFL \$ t\$.*
- integer `currentiteration`  
*Current iteration.*
- real, dimension(:,:), allocatable `temp`  
*Temperature array [K].*

#### 4.9.1 Detailed Description

This module contains variables that are treated as global in the code

### 4.10 h\_alpha\_utilities Module Reference

H alpha projection.

#### Functions/Subroutines

- subroutine `init_ha` ()  
*Initializes data.*
- subroutine `read_data` (u, itprint, filepath)  
*reads data from file*
- subroutine `getxyz` (i, j, k, x, y, z)  
*gets position of a cell*
- subroutine `rotation_x` (theta, x, y, z, xn, yn, zn)  
*Rotation around the X axis.*
- subroutine `rotation_y` (theta, x, y, z, xn, yn, zn)  
*Rotation around the Y axis.*
- subroutine `rotation_z` (theta, x, y, z, xn, yn, zn)  
*Rotation around the Z axis.*
- subroutine `fill_map` (nxmap, nymap, u, map, dxT, dyT, theta\_x, theta\_y, theta\_z)  
*Fill target map.*
- subroutine `write_ha` (fileout, nxmap, nymap, map)  
*Writes projection to file.*
- subroutine `write_rg` (fileout, nxmap, nymap, map)  
*Writes projection to file in rg format.*

#### 4.10.1 Detailed Description

Utilities to compute an H alpha map

## 4.10.2 Function/Subroutine Documentation

4.10.2.1 subroutine h\_alpha\_utilities::fill\_map ( integer, intent(in) *nxmap*, integer, intent(in) *nymap*, real, dimension(nq,nxmin:nxmax,nymin:nymax,nzmin:nzmax), intent(in) *u*, real, dimension(nxmap,nymap), intent(out) *map*, real, intent(in) *dxT*, real, intent(in) *dyT*, real, intent(in) *theta\_x*, real, intent(in) *theta\_y*, real, intent(in) *theta\_z* )

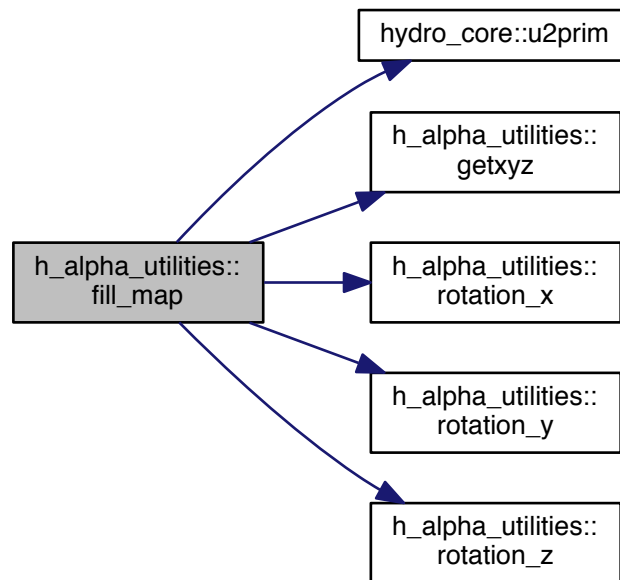
Fills the target map of one MPI block

## Parameters

<i>integer</i>	[in] nxmap : Number of X cells in target
<i>integer</i>	[in] nymap : Number of Y cells in target
<i>real</i>	[in] u(neq,nxmin:nxmax,nymin:nymax, nzmin:nzmax) : conserved variables
<i>real</i>	[out] map(nxmap,mymap) : Target map
<i>real</i>	[in] dxT : target pixel width
<i>real</i>	[in] dyT : target pixel height
<i>real</i>	[in] thetax : Rotation around X
<i>real</i>	[in] thetay : Rotation around Y
<i>real</i>	[in] thetaz : Rotation around Z

Definition at line 285 of file h\_alpha\_proj.f90.

Here is the call graph for this function:



4.10.2.2 subroutine `h_alpha_utilities::getxyz` ( *integer*, intent(in) *i*, *integer*, intent(in) *j*, *integer*, intent(in) *k*, *real*, intent(out) *x*, *real*, intent(out) *y*, *real*, intent(out) *z* )

Returns the position and spherical radius calculated with respect to the center of the grid

## Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the x direction
<i>integer</i>	[in] <i>j</i> : cell index in the y direction
<i>integer</i>	[in] <i>k</i> : cell index in the z direction
<i>real</i>	[in] <i>x</i> : x position in the grid

<i>real</i>	[in] y : y position in the grid
<i>real</i>	[in] z : z position in the grid

Definition at line 187 of file h\_alpha\_proj.f90.

#### 4.10.2.3 subroutine h\_alpha\_utilities::init\_ha ( )

Initializes data, MPI and other stuff

Definition at line 35 of file h\_alpha\_proj.f90.

#### 4.10.2.4 subroutine h\_alpha\_utilities::read\_data ( *real*, dimension(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax), intent(out) *u*, integer, intent(in) *itprint*, character (len=128), intent(in) *filepath* )

reads data from file

Parameters

<i>real</i>	[out] u(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax) : conserved variables
<i>integer</i>	[in] itprint : number of output
<i>string</i>	[in] filepath : path where the output is

Definition at line 134 of file h\_alpha\_proj.f90.

#### 4.10.2.5 subroutine h\_alpha\_utilities::rotation\_x ( *real*, intent(in) *theta*, *real*, intent(in) *x*, *real*, intent(in) *y*, *real*, intent(in) *z*, *real*, intent(out) *xn*, *real*, intent(out) *yn*, *real*, intent(out) *zn* )

Does a rotation around the x axis

Parameters

<i>real</i>	[in], theta : Angle of rotation (in radians)
<i>real</i>	[in], x : original x position in the grid
<i>real</i>	[in], y : original y position in the grid
<i>real</i>	[in], x : original z position in the grid
<i>real</i>	[out], x : final x position in the grid
<i>real</i>	[out], y : final y position in the grid
<i>real</i>	[out], x : final z position in the grid

Definition at line 213 of file h\_alpha\_proj.f90.

#### 4.10.2.6 subroutine h\_alpha\_utilities::rotation\_y ( *real*, intent(in) *theta*, *real*, intent(in) *x*, *real*, intent(in) *y*, *real*, intent(in) *z*, *real*, intent(out) *xn*, *real*, intent(out) *yn*, *real*, intent(out) *zn* )

Does a rotation around the x axis

Parameters

<i>real</i>	[in], theta : Angle of rotation (in radians)
<i>real</i>	[in], x : original x position in the grid
<i>real</i>	[in], y : original y position in the grid
<i>real</i>	[in], x : original z position in the grid
<i>real</i>	[out], x : final x position in the grid
<i>real</i>	[out], y : final y position in the grid

<i>real</i>	[out], x : final z position in the grid
-------------	---

Definition at line 237 of file h\_alpha\_proj.f90.

4.10.2.7 subroutine h\_alpha\_utilities::rotation\_z ( *real*, intent(in) *theta*, *real*, intent(in) *x*, *real*, intent(in) *y*, *real*, intent(in) *z*, *real*, intent(out) *xn*, *real*, intent(out) *yn*, *real*, intent(out) *zn* )

Does a rotation around the x axis

Parameters

<i>real</i>	[in], <i>theta</i> : Angle of rotation (in radians)
<i>real</i>	[in], <i>x</i> : original x position in the grid
<i>real</i>	[in], <i>y</i> : original y position in the grid
<i>real</i>	[in], <i>x</i> : original z position in the grid
<i>real</i>	[out], <i>x</i> : final x position in the grid
<i>real</i>	[out], <i>y</i> : final y position in the grid
<i>real</i>	[out], <i>x</i> : final z position in the grid

Definition at line 259 of file h\_alpha\_proj.f90.

4.10.2.8 subroutine h\_alpha\_utilities::write\_ha ( *character* (len=128), intent(in) *fileout*, *integer*, intent(in) *nxmap*, *integer*, intent(in) *nymap*, *real*, dimension(nxmap,nymap), intent(in) *map* )

Writes projection to file

Parameters

<i>integer</i>	[in] <i>itprint</i> : number of output
<i>string</i>	[in] <i>fileout</i> : file where to write
<i>integer</i>	[in] <i>nxmap</i> : Number of X cells in target
<i>integer</i>	[in] <i>nymap</i> : Number of Y cells in target
<i>real</i>	[in] <i>map</i> (nxmap,mymap) : Target map

Definition at line 362 of file h\_alpha\_proj.f90.

4.10.2.9 subroutine h\_alpha\_utilities::write\_rg ( *character* (len=128), intent(in) *fileout*, *integer*, intent(in) *nxmap*, *integer*, intent(in) *nymap*, *real*, dimension(nxmap,nymap), intent(in) *map* )

Writes projection to file

Parameters

<i>integer</i>	[in] <i>itprint</i> : number of output
<i>string</i>	[in] <i>fileout</i> : file where to write
<i>integer</i>	[in] <i>nxmap</i> : Number of X cells in target
<i>integer</i>	[in] <i>nymap</i> : Number of Y cells in target
<i>real</i>	[in] <i>map</i> (nxmap,mymap) : Target map

Definition at line 391 of file h\_alpha\_proj.f90.

## 4.11 hll Module Reference

HLL approximate Riemann solver module.

## Functions/Subroutines

- subroutine [prim2fhll](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLL solver.*
- subroutine [hllfluxes](#) (choice)  
*Calculates HLL fluxes from the primitive variables on all the domain.*

### 4.11.1 Detailed Description

The module contains the routines needed to Solve the Riemann problem in the entire domain and return the physical fluxes in x,y,z with the HLL solver

### 4.11.2 Function/Subroutine Documentation

#### 4.11.2.1 subroutine hll::hllfluxes ( integer, intent(in) choice )

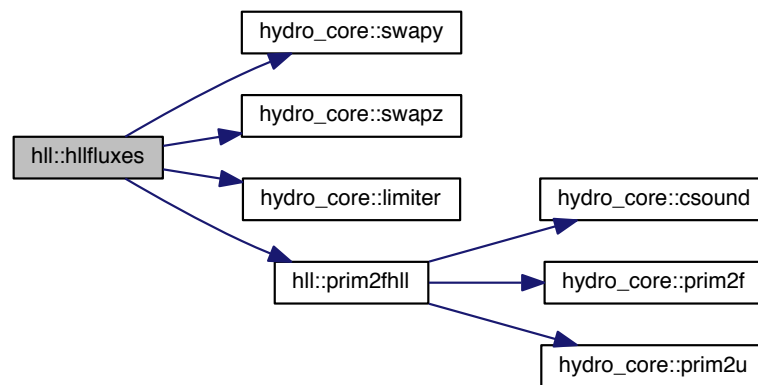
Calculates HLL fluxes from the primitive variables on all the domain

##### Parameters

<i>integer</i>	[in] choice : 1, uses primit for the 1st half of timestep (first order) 2 uses primit for second order timestep
----------------	--

Definition at line 93 of file hll.f90.

Here is the call graph for this function:



#### 4.11.2.2 subroutine hll::prim2fhll ( real, dimension(neq), intent(in) priml, real, dimension(neq), intent(in) primr, real, dimension(neq), intent(inout) ff )

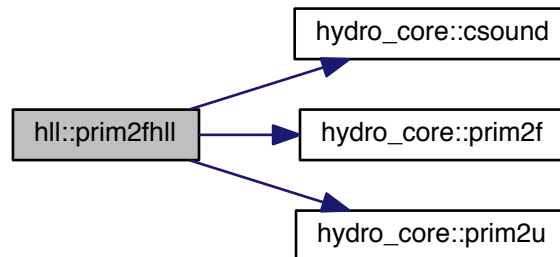
Solves the Riemann problem at the interface between PL and PR using the HLL solver  
The fluxes are computed in the X direction, to obtain the y and z directions a swap is performed

## Parameters

<i>real</i>	[in] primL : primitives at the Left state
<i>real</i>	[in] primR : primitives at the Right state
<i>real</i>	[out] ff : fluxes at the interface ( $F_{i+1/2}^i$ )

Definition at line 48 of file hll.f90.

Here is the call graph for this function:



## 4.12 hllc Module Reference

HLLC approximate Riemann solver module.

### Functions/Subroutines

- subroutine [prim2fhllc](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLLC solver.*
- subroutine [hllcfluxes](#) (choice)  
*Calculates HLLC fluxes from the primitive variables on all the domain.*

#### 4.12.1 Detailed Description

The module contains the routines needed to Solve the Riemann problem in the entire domain and return the physical fluxes in x,y,z with the HLLC solver

#### 4.12.2 Function/Subroutine Documentation

##### 4.12.2.1 subroutine hllc::hllcfluxes ( integer, intent(in) choice )

Calculates HLLC fluxes from the primitive variables on all the domain

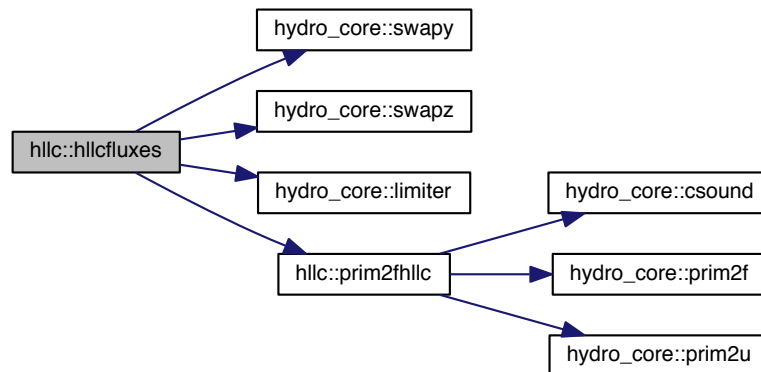
## Parameters

<i>integer</i>	[in] choice : 1, uses primit for the 1st half of timestep (first order) 2 uses primit for second order timestep
----------------	--

Definition at line 146 of file hllc.f90.



Here is the call graph for this function:



4.12.2.2 subroutine hllc::prim2fhllc ( real, dimension(neq), intent(in) *priml*, real, dimension(neq), intent(in) *primr*, real, dimension(neq), intent(inout) *ff* )

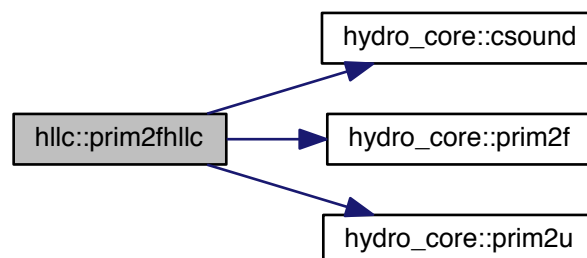
Solves the Riemann problem at the interface between PL and PR using the HLLC solver  
The fluxes are computed in the X direction, to obtain the y and z directions a swap is performed

Parameters

<i>real</i>	[in] primL : primitives at the Left state
<i>real</i>	[in] primR : primitives at the Right state
<i>real</i>	[out] ff : fluxes at the interface ( $F_{i+1/2}$ )

Definition at line 47 of file hllc.f90.

Here is the call graph for this function:



## 4.13 hlld Module Reference

HLLD approximate Riemann solver module.

## Functions/Subroutines

- subroutine `prim2fhlld` (`priml`, `primr`, `ff`)  
*Solves the Riemann problem at the interface PL,PR using the HLLD solver.*
- subroutine `hlldfluxes` (`choice`)  
*Calculates HLLD fluxes from the primitive variables on all the domain.*

### 4.13.1 Detailed Description

The module contains the routines needed to Solve the Riemann problem in the entire domain and return the physical fluxes in x,y,z with the HLLD solver

### 4.13.2 Function/Subroutine Documentation

#### 4.13.2.1 subroutine `hlld::hlldfluxes` ( `integer`, `intent(in)` *choice* )

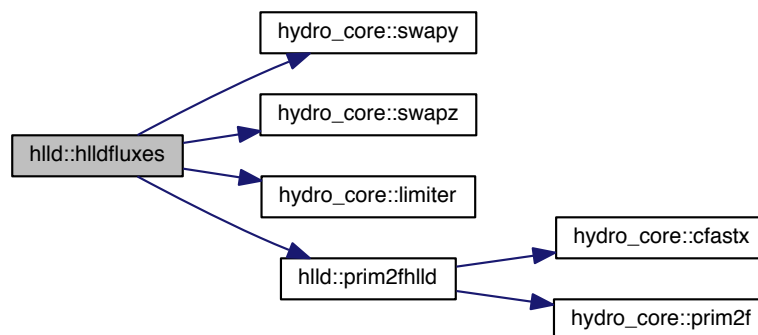
Calculates HLLD fluxes from the primitive variables on all the domain

##### Parameters

<i>integer</i>	[in] <i>choice</i> : 1, uses primit for the 1st half of timestep (first order) 2 uses primit for second order timestep
----------------	---

Definition at line 328 of file `hlld.f90`.

Here is the call graph for this function:



#### 4.13.2.2 subroutine `hlld::prim2fhlld` ( `real`, `dimension(neq)`, `intent(in)` *priml*, `real`, `dimension(neq)`, `intent(in)` *primr*, `real`, `dimension(neq)`, `intent(inout)` *ff* )

Solves the Riemann problem at the interface between PL and PR using the HLLD solver

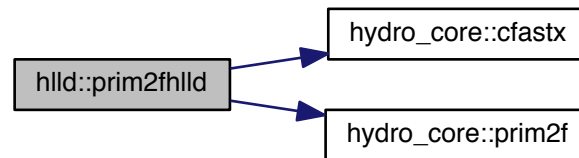
The fluxes are computed in the X direction, to obtain the y and z directions a swap is performed

##### Parameters

<i>real</i>	[in] primL : primitives at the Left state
<i>real</i>	[in] primR : primitives at the Right state
<i>real</i>	[out] ff : fluxes at the interface ( $F_{i+1/2}^i$ )

Definition at line 49 of file hlld.f90.

Here is the call graph for this function:



## 4.14 hlle Module Reference

HLLE approximate Riemann solver module.

### Functions/Subroutines

- subroutine [prim2fhlle](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLLE solver.*
- subroutine [hllefluxes](#) (choice)  
*Calculates HLLE fluxes from the primitive variables on all the domain.*

### 4.14.1 Detailed Description

The module contains the routines needed to Solve the Riemann problem in the entire domain and return the physical fluxes in x,y,z with the HLLE solver

### 4.14.2 Function/Subroutine Documentation

#### 4.14.2.1 subroutine hlle::hllefluxes ( integer, intent(in) choice )

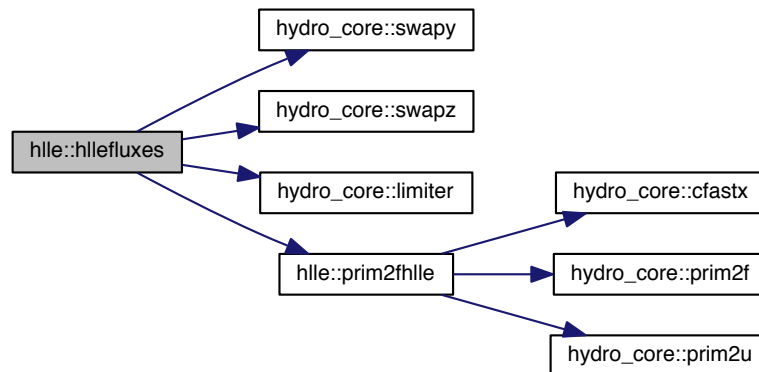
Calculates HLLE fluxes from the primitive variables on all the domain

##### Parameters

<i>integer</i>	[in] choice : 1, uses primit for the 1st half of timestep (first order) 2 uses primit for second order timestep
----------------	--

Definition at line 94 of file hlle.f90.

Here is the call graph for this function:



4.14.2.2 subroutine `hlle::prim2fhle` ( *real*, dimension(neq), intent(in) *priml*, *real*, dimension(neq), intent(in) *primr*, *real*, dimension(neq), intent(inout) *ff* )

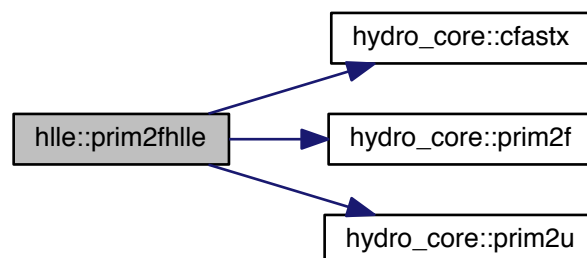
Solves the Riemann problem at the interface between PL and PR using the HLL solver  
The fluxes are computed in the X direction, to obtain the y and z directions a swap is performed

Parameters

<i>real</i>	[in] <i>primL</i> : primitives at the Left state
<i>real</i>	[in] <i>primR</i> : primitives at the Right state
<i>real</i>	[out] <i>ff</i> : fluxes at the interface ( $F_{i+1/2}$ )

Definition at line 49 of file `hlle.f90`.

Here is the call graph for this function:



## 4.15 hydro\_core Module Reference

Basic hydro (and MHD) subroutines utilities.

## Functions/Subroutines

- subroutine [u2prim](#) (uu, prim, T)  
*Computes the primitive variables and temperature from conserved variables on a single cell.*
- subroutine [calcprim](#) (u, primit, only\_ghost)  
*Updated the primitives, using the conserved variables in the entire domain.*
- subroutine [prim2u](#) (prim, uu)  
*Computes the conserved conserved variables from the primitives in a single cell.*
- subroutine [prim2f](#) (prim, ff)  
*Computes the Euler Fluxes in one cell.*
- subroutine [swapy](#) (var, neq)  
*Swaps the x and y components in a cell.*
- subroutine [swapz](#) (var, neq)  
*Swaps the x and z components in a cell.*
- subroutine [csound](#) (p, d, cs)  
*Computes the sound speed.*
- subroutine [cfast](#) (p, d, bx, by, bz, cfx, cfy, cfz)  
*Computes the fast magnetosonic speeds in the 3 coordinates.*
- subroutine [cfastx](#) (prim, cfX)  
*Computes the fast magnetosonic speed in the x direction.*
- subroutine [get\\_timestep](#) (current\_iter, n\_iter, current\_time, tprint, dt, dump\_flag)  
*Obtains the timestep allowed by the CFL condition in the entire.*
- subroutine [limiter](#) (PLL, PL, PR, PRR, neq)  
*Performs a linear reconstruction of the primitive variables.*

### 4.15.1 Detailed Description

This module contains subroutines and utilities that are the core of the hydro (and MHD) that are common to most implementations and will be used for the different specific solvers

### 4.15.2 Function/Subroutine Documentation

- 4.15.2.1 subroutine `hydro_core::calcprim` ( `real`, `dimension(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax)`, `intent(in)` `u`, `real`, `dimension(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax)`, `intent(out)` `primit`, `logical`, `intent(in)`, optional `only_ghost` )

Updated the primitives, using the conserved variables in the entire domain

#### Parameters

<i>real</i>	[in] <code>u(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax)</code> : conserved variables
<i>real</i>	[out] <code>prim(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax)</code> : primitive variables
<i>logical</i>	[in] <code>only_ghost</code> : if set to true then updates the primitives only on the ghost cells, it defaults to false (the entire domain is updated)

Definition at line 134 of file `hydro_core.f90`.

Here is the call graph for this function:



**4.15.2.2** subroutine `hydro_core::cfast` ( *real*, intent(in) *p*, *real*, intent(in) *d*, *real*, intent(in) *bx*, *real*, intent(in) *by*, *real*, intent(in) *bz*, *real*, intent(out) *cfx*, *real*, intent(out) *cfy*, *real*, intent(out) *cfz* )

Computes the fast magnetosonic speeds in the 3 coordinates

#### Parameters

<i>real</i>	[in] <i>p</i> : value of pressure
<i>real</i>	[in] <i>d</i> : value of density
<i>real</i>	[in] <i>Bx</i> : value of the x component of the magnetic field
<i>real</i>	[in] <i>By</i> : value of the y component of the magnetic field
<i>real</i>	[in] <i>Bz</i> : value of the z component of the magnetic field
<i>real</i>	[out] <i>csx</i> : fast magnetisonic speed in x
<i>real</i>	[out] <i>csy</i> : fast magnetisonic speed in y
<i>real</i>	[out] <i>csz</i> : fast magnetisonic speed in z

Definition at line 398 of file `hydro_core.f90`.

**4.15.2.3** subroutine `hydro_core::cfastx` ( *real*, dimension(neq), intent(in) *prim*, *real*, intent(out) *cfX* )

Computes the fast magnetosonic speed in the x direction

#### Parameters

<i>real</i>	[in] <i>prim</i> (neq) : vector with the primitives in one cell
-------------	---

Definition at line 423 of file `hydro_core.f90`.

**4.15.2.4** subroutine `hydro_core::csound` ( *real*, intent(in) *p*, *real*, intent(in) *d*, *real*, intent(out) *cs* )

Computes the sound speed

#### Parameters

<i>real</i>	[in] <i>p</i> : value of pressure
<i>real</i>	[in] <i>d</i> : value of density
<i>real</i>	[out] <i>cs</i> : sound speed

Definition at line 372 of file `hydro_core.f90`.

**4.15.2.5** subroutine `hydro_core::get_timestep` ( *integer*, intent(in) *current\_iter*, *integer*, intent(in) *n\_iter*, *real*, intent(in) *current\_time*, *real*, intent(in) *tprint*, *real*, intent(out) *dt*, *logical*, intent(out) *dump\_flag* )

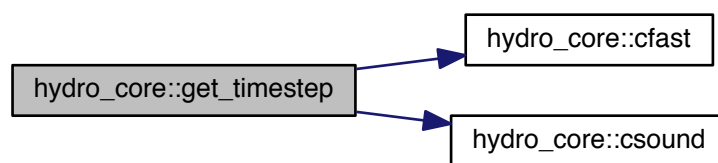
Obtains the timestep allowed by the CFL condition in the entire domain using the global primitives, and sets logical variable to dump output

## Parameters

<i>integer</i>	[in] <i>current_iter</i> : Current iteration, it starts with a small but increasing CFL in the first <i>N_trans</i> iterations
<i>integer</i>	[in] <i>n_iter</i> : Number of iterations to go from a small CFL to the final CFL (in parameters.f90)
<i>real</i>	[in] <i>current_time</i> : Current (global) simulation time
<i>real</i>	[in] <i>tprint</i> : time for the next programed disk dump
<i>real</i>	[out] : $\Delta t$ allowed by the CFL condition
<i>logical</i>	[out] <i>dump_flag</i> : Flag to write to disk

Definition at line 455 of file hydro\_core.f90.

Here is the call graph for this function:



**4.15.2.6** subroutine `hydro_core::limiter` ( *real*, dimension(*neq*), intent(in) *PLL*, *real*, dimension(*neq*), intent(inout) *PL*, *real*, dimension(*neq*), intent(inout) *PR*, *real*, dimension(*neq*), intent(in) *PRR*, *integer*, intent(in) *neq* )

returns a linear reconstruction of the variables at the interface between the primitives *PLL*, *PL*, *PR*, *PRR*  
 The reconstruction is made with a slope limiter chosen at compilation time (i.e. set on the Makefile)

## Parameters

<i>real</i>	[in] : primitives at the left of the left state
<i>real</i>	[inout] : primitives at the left state
<i>real</i>	[inout] : primitives at the right state
<i>real</i>	[in] : primitives at the right of the right state
<i>real</i>	[in] : number of equations

Definition at line 533 of file hydro\_core.f90.

**4.15.2.7** subroutine `hydro_core::prim2f` ( *real*, dimension(*neq*), intent(in) *prim*, *real*, dimension(*neq*), intent(out) *ff* )

Computes the Euler Fluxes in one cell, using the primitives

It returns the flux in the x direction (i.e. *F*), the y and z fluxes can be obtained swaping the respective entries (see `swapy` and `swapz` subroutines)

## Parameters

<i>real</i>	[in] <i>prim</i> ( <i>neq</i> ) : primitives in one cell
<i>real</i>	[out] <i>ff</i> ( <i>neq</i> ) : Euler Fluxes (x direction)

Definition at line 270 of file hydro\_core.f90.

4.15.2.8 subroutine hydro\_core::prim2u ( real, dimension(neq), intent(in) *prim*, real, dimension(neq), intent(out) *uu* )

Computes the conserved variables from the primitives in a single cell



## Parameters

<i>real</i>	[in] prim(neq) : primitives in one cell
<i>real</i>	[out] uu(neq) : conserved variables in one cell

Definition at line 229 of file hydro\_core.f90.

## 4.15.2.9 subroutine hydro\_core::swapy ( real, dimension(neq), intent(inout) var, integer, intent(in) neq )

Swaps the x and y components in a cell.

## Parameters

<i>real</i>	[inout] var(neq) : variable to be swapped
<i>real</i>	[in] neq : number of equations in the code

Definition at line 320 of file hydro\_core.f90.

## 4.15.2.10 subroutine hydro\_core::swapz ( real, dimension(neq), intent(inout) var, integer, intent(in) neq )

Swaps the x and z components in a cell.

## Parameters

<i>real</i>	[inout] var(neq) : variable to be swapped
<i>real</i>	[in] neq : number of equations in the code

Definition at line 346 of file hydro\_core.f90.

## 4.15.2.11 subroutine hydro\_core::u2prim ( real, dimension(neq), intent(in) uu, real, dimension(neq), intent(out) prim, real, intent(out) T )

Computes the primitive variables and temperature from conserved variables on a single cell

## Parameters

<i>real</i>	[in] uu(neq) : conserved variables in one cell
<i>real</i>	[out] prim(neq) : primitives in one cell
<i>real</i>	[out] T : Temperature [K]

Definition at line 45 of file hydro\_core.f90.

## 4.16 hydro\_solver Module Reference

Advances the simulation one timestep.

## Functions/Subroutines

- subroutine [viscosity](#) ()  
*Adds artificial viscosity to the conserved variables.*
- subroutine [step](#) (dt)  
*Upwind timestep.*
- subroutine [tstep](#) ()  
*High level wrapper to advance the simulation.*

### 4.16.1 Detailed Description

Advances the solution from  $t$  to  $t + \Delta t$

### 4.16.2 Function/Subroutine Documentation

#### 4.16.2.1 subroutine hydro\_solver::step ( real, intent(in) dt )

Performs the upwind timestep according to

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x} \left[ F_{i+1/2}^{n+1/2} - F_{i-1/2}^{n+1/2} \right]$$

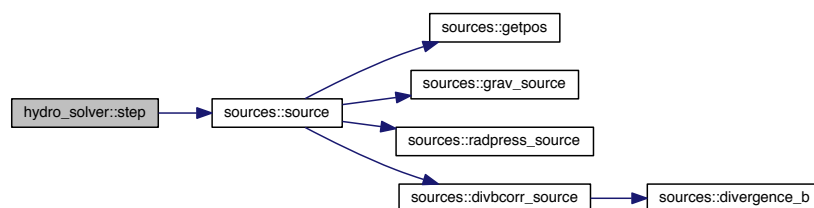
(in 3D), it takes  $U^{n+1}$ =up from the global variables and  $U^n$ =u

Parameters

<i>real</i>	[in] dt : timestep
-------------	--------------------

Definition at line 84 of file hydro\_solver.f90.

Here is the call graph for this function:



#### 4.16.2.2 subroutine hydro\_solver::tstep ( )

High level wrapper to advance the simulation  
The variables are taken from the globals module.

Definition at line 126 of file hydro\_solver.f90.

#### 4.16.2.3 subroutine hydro\_solver::viscosity ( )

Takes the variables from the `globals` module and it assumes that the `up` are the stepped variables, while `u` are unstepped

Generated on Thu Mar 10 2016 12:30:11 for Guacho 3D by Doxygen

## 4.17 init Module Reference

Guacho-3D initialization.

### Functions/Subroutines

- subroutine `initmain` (`tprint`, `itprint`)

*Main initialization routine.*

- subroutine `initflow` (`itprint`)

*Initializes the conserved variables, in the globals module.*

### 4.17.1 Detailed Description

This module contains the routines needed to initialize the code, it also initializes all the modules set by the user.

### 4.17.2 Function/Subroutine Documentation

#### 4.17.2.1 subroutine `init::initflow` ( `integer`, `intent(inout) itprint` )

Initializes the conserved variables, in the globals module

Parameters

<i>real</i>	[inout] <code>itprint</code> : number of current output
-------------	---

Definition at line 435 of file `init.f90`.

#### 4.17.2.2 subroutine `init::initmain` ( `real`, `intent(out) tprint`, `integer`, `intent(out) itprint` )

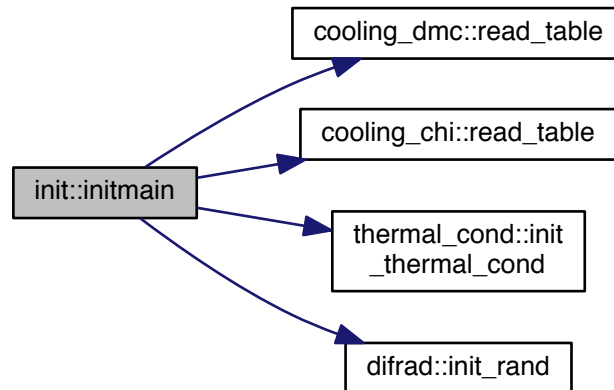
This subroutine initializes all the variables in the globals module, MPI, cooling and `user_mod` routines; and outputs to screen the main parameters used in the run

Parameters

<i>real</i>	[out] <code>tprint</code> : time of next output
<i>integer</i>	[out] <code>itprint</code> : number of next output

Definition at line 41 of file `init.f90`.

Here is the call graph for this function:



## 4.18 linear\_system Module Reference

linear system inversion module

### Functions/Subroutines

- subroutine [ludcmp](#) (a, n, indx, d)  
*LU decomposition.*
- subroutine [lubksb](#) (a, n, indx, b)  
*Solves a set of linear equations.*
- subroutine [linsys](#) (a, b, n)  
*Driver to solves a set of linear equations.*

### 4.18.1 Detailed Description

Inversion of a system of linear equations with an LU decomposition method (these routines are from Numerical Methods by Press et al.)

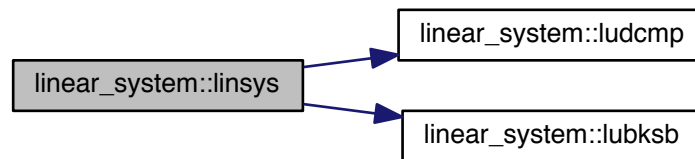
### 4.18.2 Function/Subroutine Documentation

4.18.2.1 subroutine `linear_system::linsys` ( `real (kind=8), dimension(n,n) a`, `real (kind=8), dimension(n) b`, `integer, intent(in) n` )

Solves a linear set of equations

Definition at line 178 of file `linear_system.f90`.

Here is the call graph for this function:



4.18.2.2 subroutine `linear_system::lubksb` ( `real` (kind=8), dimension(n,n), intent(in) *a*, `integer`, intent(in) *n*, `integer`, dimension(n), intent(in) *indx*, `real` (kind=8), dimension(n), intent(inout) *b* )

Solves a linear set of equations of the form

Definition at line 129 of file `linear_system.f90`.

4.18.2.3 subroutine `linear_system::ludcmp` ( `real` (kind=8), dimension(n,n), intent(inout) *a*, `integer`, intent(in) *n*, `integer`, dimension(n), intent(out) *indx*, `real` (kind=8), intent(inout) *d* )

LU decomposition of a row-wise permutation

Parameters

<i>real</i>	[inout] <i>a</i> (n,n) : matrix to be decomposed result is done in place
<i>integer</i>	[in] <i>n</i> : size of the matrix
<i>real</i>	[out] <i>index</i> (n) : vector that contains the row permutation affected by the partial pivoting
<i>integer</i>	[inout] <i>d</i> : +/- 1 depending if the row intergarches is even or odd

Definition at line 46 of file `linear_system.f90`.

## 4.19 lyman\_alpha\_utilities Module Reference

Lyman\_alpha\_utilities.

### Functions/Subroutines

- subroutine `init_la` ()  
*Initializes data.*
- subroutine `read_data` (*u*, *itprint*, *filepath*)  
*reads data from file*
- subroutine `getxyz` (*i*, *j*, *k*, *x*, *y*, *z*)  
*gets position of a cell*
- subroutine `rotation_x` (*theta*, *x*, *y*, *z*, *xn*, *yn*, *zn*)  
*Rotation around the X axis.*
- subroutine `rotation_y` (*theta*, *x*, *y*, *z*, *xn*, *yn*, *zn*)  
*Rotation around the Y axis.*
- subroutine `rotation_z` (*theta*, *x*, *y*, *z*, *xn*, *yn*, *zn*)

*Rotation around the Z axis.*

- subroutine [fill\\_map](#) (nxmap, nymap, nvmap, vmin, vmax, u, map, dxT, dyT, theta\_x, theta\_y, theta\_z)

*Fill target map.*

- subroutine [write\\_la](#) (itprint, filepath, nxmap, nymap, nvmap, map)

*Writes projection to file.*

- subroutine [phigauss](#) (T, vzn, vmin, vmax, nvmap, profile)

*This routine computes a gaussian line profile.*

### 4.19.1 Detailed Description

Utilities to compute the Lyman-

### 4.19.2 Function/Subroutine Documentation

- 4.19.2.1 subroutine `lyman_alpha_utilities::fill_map` ( integer, intent(in) *nxmap*, integer, intent(in) *nymap*, integer, intent(in) *nvmap*, real, intent(in) *vmin*, real, intent(in) *vmax*, real, dimension(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax), intent(in) *u*, real, dimension(nxmap,nymap,nvmap), intent(out) *map*, real, intent(in) *dxT*, real, intent(in) *dyT*, real, intent(in) *theta\_x*, real, intent(in) *theta\_y*, real, intent(in) *theta\_z* )

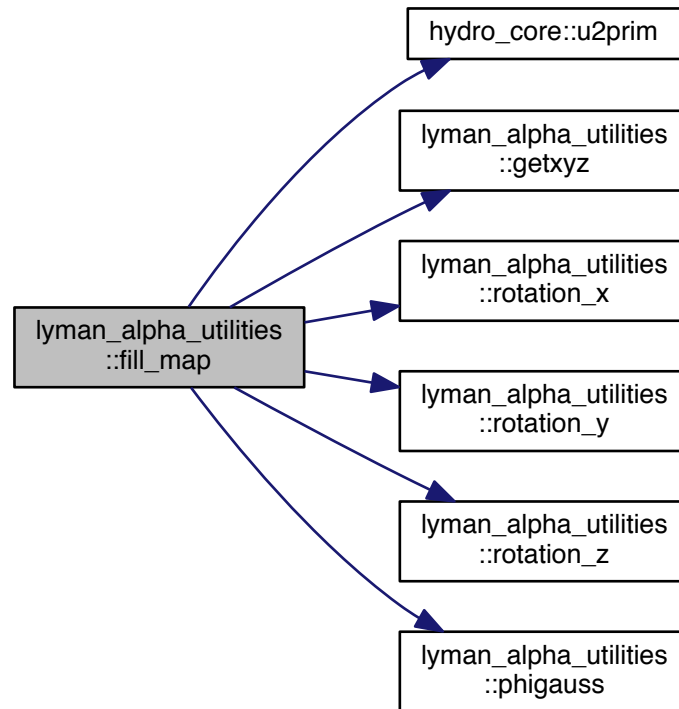
Fills the target map of one MPI block

Parameters

<i>integer</i>	[in] <i>nxmap</i> : Number of X cells in target
<i>integer</i>	[in] <i>nymap</i> : Number of Y cells in target
<i>real</i>	[in] <i>u</i> (neq,nxmin:nxmax,nymin:nymax, nzmin:nzmax) : conserved variables
<i>real</i>	[out] <i>map</i> (nxmap,nymap) : Target map
<i>real</i>	[in] <i>dxT</i> : target pixel width
<i>real</i>	[in] <i>dyT</i> : target pixel height
<i>real</i>	[in] <i>thetax</i> : Rotation around X
<i>real</i>	[in] <i>thetay</i> : Rotation around Y
<i>real</i>	[in] <i>thetaz</i> : Rotation around Z

Definition at line 285 of file `lyman_alpha_tau.f90`.

Here is the call graph for this function:



**4.19.2.2** subroutine `lyman_alpha_utilities::getxyz` ( *integer*, intent(in) *i*, *integer*, intent(in) *j*, *integer*, intent(in) *k*, *real*, intent(out) *x*, *real*, intent(out) *y*, *real*, intent(out) *z* )

Returns the position and spherical radius calculated with respect to the center of the grid

Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the x direction
<i>integer</i>	[in] <i>j</i> : cell index in the y direction
<i>integer</i>	[in] <i>k</i> : cell index in the z direction
<i>real</i>	[in] <i>x</i> : x position in the grid
<i>real</i>	[in] <i>y</i> : y position in the grid
<i>real</i>	[in] <i>z</i> : z position in the grid

Definition at line 186 of file `lyman_alpha_tau.f90`.

**4.19.2.3** subroutine `lyman_alpha_utilities::init_la` ( )

Initializes data, MPI and other stuff

Definition at line 36 of file `lyman_alpha_tau.f90`.



4.19.2.4 subroutine lyman\_alpha\_utilities::phigauss ( real, intent(in) *T*, real, intent(in) *vzn*, real, intent(in) *vmin*, real, intent(in) *vmax*, integer, intent(in) *nvmap*, real, dimension(nvmap), intent(out) *profile* )

This routine computes a gaussian line profile

Definition at line 386 of file lyman\_alpha\_tau.f90.

4.19.2.5 subroutine lyman\_alpha\_utilities::read\_data ( real, dimension(neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax), intent(out) *u*, integer, intent(in) *itprint*, character (len=128), intent(in) *filepath* )

reads data from file

Parameters

<i>real</i>	[out] <i>u</i> (neq,nxmin:nxmax,nymin:nymax,nzmin:nzmax) : conserved variables
<i>integer</i>	[in] <i>itprint</i> : number of output
<i>string</i>	[in] <i>filepath</i> : path where the output is

Definition at line 136 of file lyman\_alpha\_tau.f90.

4.19.2.6 subroutine lyman\_alpha\_utilities::rotation\_x ( real, intent(in) *theta*, real, intent(in) *x*, real, intent(in) *y*, real, intent(in) *z*, real, intent(out) *xn*, real, intent(out) *yn*, real, intent(out) *zn* )

Does a rotation around the x axis

Parameters

<i>real</i>	[in], <i>theta</i> : Angle of rotation (in radians)
<i>real</i>	[in], <i>x</i> : original x position in the grid
<i>real</i>	[in], <i>y</i> : original y position in the grid
<i>real</i>	[in], <i>z</i> : original z position in the grid
<i>real</i>	[out], <i>x</i> : final x position in the grid
<i>real</i>	[out], <i>y</i> : final y position in the grid
<i>real</i>	[out], <i>z</i> : final z position in the grid

Definition at line 212 of file lyman\_alpha\_tau.f90.

4.19.2.7 subroutine lyman\_alpha\_utilities::rotation\_y ( real, intent(in) *theta*, real, intent(in) *x*, real, intent(in) *y*, real, intent(in) *z*, real, intent(out) *xn*, real, intent(out) *yn*, real, intent(out) *zn* )

Does a rotation around the x axis

Parameters

<i>real</i>	[in], <i>theta</i> : Angle of rotation (in radians)
<i>real</i>	[in], <i>x</i> : original x position in the grid
<i>real</i>	[in], <i>y</i> : original y position in the grid
<i>real</i>	[in], <i>z</i> : original z position in the grid
<i>real</i>	[out], <i>x</i> : final x position in the grid
<i>real</i>	[out], <i>y</i> : final y position in the grid
<i>real</i>	[out], <i>z</i> : final z position in the grid

Definition at line 236 of file lyman\_alpha\_tau.f90.

4.19.2.8 subroutine lyman\_alpha\_utilities::rotation\_z ( real, intent(in) *theta*, real, intent(in) *x*, real, intent(in) *y*, real, intent(in) *z*, real, intent(out) *xn*, real, intent(out) *yn*, real, intent(out) *zn* )

Does a rotation around the x axis

## Parameters

<i>real</i>	[in], theta : Angle of rotation (in radians)
<i>real</i>	[in], x : original x position in the grid
<i>real</i>	[in], y : original y position in the grid
<i>real</i>	[in], x : original z position in the grid
<i>real</i>	[out], x : final x position in the grid
<i>real</i>	[out], y : final y position in the grid
<i>real</i>	[out], x : final z position in the grid

Definition at line 258 of file lyman\_alpha\_tau.f90.

4.19.2.9 subroutine lyman\_alpha\_utilities::write\_la ( integer, intent(in) *itprint*, character (len=128), intent(in) *filepath*, integer, intent(in) *nxmap*, integer, intent(in) *nymap*, integer, intent(in) *nvmap*, real, dimension(nxmap,nymap,nvmap), intent(in) *map* )

Writes projection to file

## Parameters

<i>integer</i>	[in] <i>itprint</i> : number of output
<i>string</i>	[in] <i>filepath</i> : path where to write
<i>integer</i>	[in] <i>nxmap</i> : Number of X cells in target
<i>integer</i>	[in] <i>nymap</i> : Number of Y cells in target
<i>integer</i>	[in] <i>nvmap</i> : Number of velocity channels
<i>real</i>	[in] <i>map</i> (nxmap,mymap) : Target map

Definition at line 361 of file lyman\_alpha\_tau.f90.

## 4.20 out\_bin\_module Module Reference

Output in BIN format.

### Functions/Subroutines

- subroutine [write\\_header](#) (unit, neq\_out, nghost\_out)  
*Writes header.*
- subroutine [write\\_bin](#) (itprint)  
*Writes Data, one file per processor.*

#### 4.20.1 Detailed Description

This module writes the ouput in BIN format

#### 4.20.2 Function/Subroutine Documentation

4.20.2.1 subroutine out\_bin\_module::write\_bin ( integer, intent(in) *itprint* )

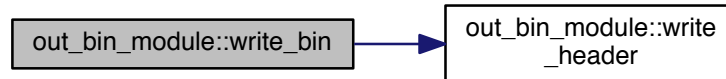
Writes Data in BIN format one file per processor

## Parameters

<i>integer</i>	[in] itprint : number of output
----------------	---------------------------------

Definition at line 112 of file Out\_BIN\_Module.f90.

Here is the call graph for this function:



4.20.2.2 subroutine out\_bin\_module::write\_header ( integer, intent(in) *unit*, integer, intent(in) *neq\_out*, integer, intent(in) *nghost\_out* )

Writes header for binary input

## Parameters

<i>integer</i>	[in] unit : number of logical unit
----------------	------------------------------------

Definition at line 44 of file Out\_BIN\_Module.f90.

## 4.21 out\_silo\_module Module Reference

Output in Silo (+HDF5) Format.

### Functions/Subroutines

- subroutine [writeblocks](#) (itprint)  
*Writes Data, one file per processor.*
- subroutine [writemaster](#) (itprint)  
*Writes the Master File.*
- subroutine [outputsilo](#) (itprint)  
*Upper level wrapper.*

#### 4.21.1 Detailed Description

This module writes the output in SILO (HDF5) format

#### 4.21.2 Function/Subroutine Documentation

4.21.2.1 subroutine out\_silo\_module::outputsilo ( integer, intent(in) *itprint* )

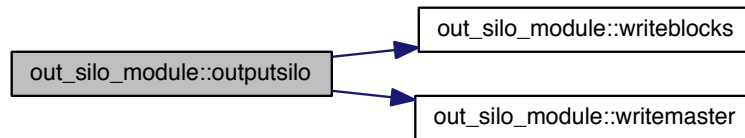
Upper level wrapper for the SILO output

**Parameters**

<i>integer</i>	[in] itprint : number of output
----------------	---------------------------------

Definition at line 347 of file Out\_Silo\_Module.f90.

Here is the call graph for this function:



#### 4.21.2.2 subroutine out\_silo\_module::writeblocks ( integer, intent(in) itprint )

Writes Data in silo format one file per processor

**Parameters**

<i>integer</i>	[in] itprint : number of output
----------------	---------------------------------

Definition at line 44 of file Out\_Silo\_Module.f90.

#### 4.21.2.3 subroutine out\_silo\_module::writemaster ( integer, intent(in) itprint )

Writes the master file with the metadata and multivars

**Parameters**

<i>integer</i>	[in] itprint : number of output
----------------	---------------------------------

Definition at line 198 of file Out\_Silo\_Module.f90.

## 4.22 out\_vtk\_module Module Reference

Output in VTK format.

**Functions/Subroutines**

- subroutine [write\\_vtk](#) (itprint)  
*Writes Data, one file per processor.*

### 4.22.1 Detailed Description

This module writes the output in VTK format

## 4.22.2 Function/Subroutine Documentation

### 4.22.2.1 subroutine out\_vtk\_module::write\_vtk ( integer, intent(in) *itprint* )

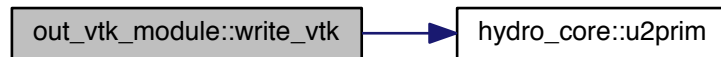
Writes Data in VTK format one file per processor

## Parameters

<i>integer</i>	[in] itprint : number of output
----------------	---------------------------------

Definition at line 43 of file Out\_VTK\_Module.f90.

Here is the call graph for this function:



## 4.23 output Module Reference

Writes output.

### Functions/Subroutines

- subroutine [write\\_output](#) (itprint)

*Writes output.*

#### 4.23.1 Detailed Description

This module writes the ouput in the formats specified in the makefile

#### 4.23.2 Function/Subroutine Documentation

##### 4.23.2.1 subroutine output::write\_output ( integer, intent(in) *itprint* )

Writes output, the format is chosen in makefile

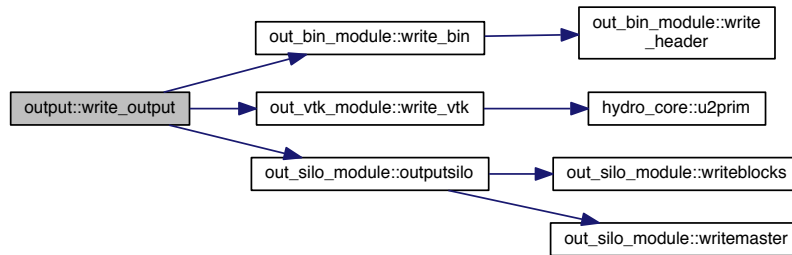
Supported formats are \*.bin and VTK (both BINARY), Silo (+hdf5)

## Parameters

<i>integer</i>	[in] itprint : number of output
----------------	---------------------------------

Definition at line 42 of file output.f90.

Here is the call graph for this function:



## 4.24 sources Module Reference

Adds source terms.

### Functions/Subroutines

- subroutine [getpos](#) (i, j, k, x, y, z, r)  
*Gets position in the grid.*
- subroutine [grav\\_source](#) (xc, yc, zc, pp, s)  
*Gravity due to point sources.*
- subroutine [radpress\\_source](#) (i, j, k, xc, yc, zc, rc, pp, s)  
*Radiation pressure force.*
- subroutine [divergence\\_b](#) (i, j, k, d)  
*Computes  $\text{div}(B)$*
- subroutine [divbcorr\\_source](#) (i, j, k, pp, s)  
*8 Wave source terms for  $\text{div}(B)$  correction*
- subroutine [source](#) (i, j, k, prim, s)  
*Upper level wrapper for sources.*

### 4.24.1 Detailed Description

This module adds the source terms from gravity, radiation pressure (not fully tested), and  $\text{div}(B)$  cleaning if the 8 wave scheme is used

### 4.24.2 Function/Subroutine Documentation

- 4.24.2.1 subroutine [sources::divbcorr\\_source](#) ( integer, intent(in) i, integer, intent(in) j, integer, intent(in) k, real, dimension(neq), intent(in) pp, real, dimension(neq), intent(inout) s )

Adds terms proportional to  $\text{div } B$  in Faraday's Law, momentum equation and energy equation as proposed in Powell et al. 1999

## Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the X direction
<i>integer</i>	[in] <i>j</i> : cell index in the Y direction
<i>integer</i>	[in] <i>k</i> : cell index in the Z direction
<i>real</i>	[in] <i>pp(neq)</i> : vector of primitive variables
<i>real</i>	[out] <i>s(neq)</i> : vector with source terms

Definition at line 201 of file sources.f90.

Here is the call graph for this function:



4.24.2.2 subroutine `sources::divergence_b` ( *integer*, intent(in) *i*, *integer*, intent(in) *j*, *integer*, intent(in) *k*, *real*, intent(out) *d* )

Computes div(B)

## Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the X direction
<i>integer</i>	[in] <i>j</i> : cell index in the Y direction
<i>integer</i>	[in] <i>k</i> : cell index in the Z direction
<i>real</i>	[out] <i>d</i> :: div(B)

Definition at line 178 of file sources.f90.

4.24.2.3 subroutine `sources::getpos` ( *integer*, intent(in) *i*, *integer*, intent(in) *j*, *integer*, intent(in) *k*, *real*, intent(out) *x*, *real*, intent(out) *y*, *real*, intent(out) *z*, *real*, intent(out) *r* )

Gets the position and spherical radius calculated with respect to the center of the grid

## Parameters

<i>integer</i>	[in] <i>i</i> : index in the X direction
<i>integer</i>	[in] <i>j</i> : index in the Y direction
<i>integer</i>	[in] <i>k</i> : index in the Z direction
<i>real</i>	[out] <i>x</i> : X position form the center of the grid (code units)
<i>real</i>	[out] <i>y</i> : Y position form the center of the grid (code units)
<i>real</i>	[out] <i>z</i> : Z position form the center of the grid (code units)
<i>real</i>	[out] <i>r</i> : Spherical radius form the center of the grid (code units)

Definition at line 55 of file sources.f90.

4.24.2.4 subroutine `sources::grav_source` ( *real*, intent(in) *xc*, *real*, intent(in) *yc*, *real*, intent(in) *zc*, *real*, dimension(neq), intent(in) *pp*, *real*, dimension(neq), intent(inout) *s* )

Adds the gravitational force due to point particles, at this moment is fixed to two point sources (exoplanet)



## Parameters

<i>real</i>	[in] xc : X position of the cell
<i>real</i>	[in] yc : Y position of the cell
<i>real</i>	[in] zc : Z position of the cell
<i>real</i>	[in] pp(neq) : vector of primitive variables
<i>real</i>	[out] s(neq) : vector with source terms

Definition at line 82 of file sources.f90.

4.24.2.5 subroutine sources::radpress\_source ( integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real, intent(in) *xc*, real, intent(in) *yc*, real, intent(in) *zc*, real, intent(in) *rc*, real, dimension(neq), intent(in) *pp*, real, dimension(neq), intent(inout) *s* )

Adds the radiation pressure force due to photo-ionization

## Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the X direction
<i>integer</i>	[in] <i>j</i> : cell index in the Y direction
<i>integer</i>	[in] <i>k</i> : cell index in the Z direction
<i>real</i>	[in] xc : X position of the cell
<i>real</i>	[in] yc : Y position of the cell
<i>real</i>	[in] zc : Z position of the cell
<i>real</i>	[in] rc : $\sqrt{x^2 + y^2 + z^2}$
<i>real</i>	[in] pp(neq) : vector of primitive variables
<i>real</i>	[out] s(neq) : vector with source terms

Definition at line 140 of file sources.f90.

4.24.2.6 subroutine sources::source ( integer, intent(in) *i*, integer, intent(in) *j*, integer, intent(in) *k*, real, dimension(neq), intent(in) *prim*, real, dimension(neq), intent(out) *s* )

Upper level wrapper for sources

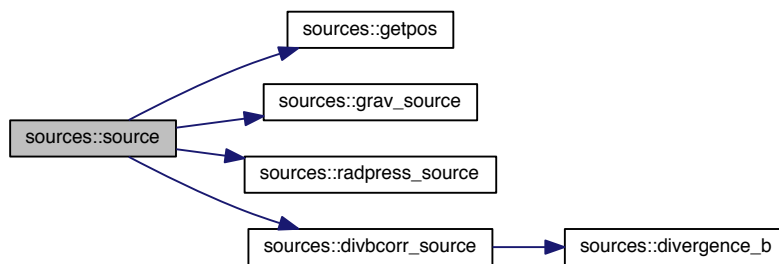
Main driver, this is called from the upwind stepping

## Parameters

<i>integer</i>	[in] <i>i</i> : cell index in the X direction
<i>integer</i>	[in] <i>j</i> : cell index in the Y direction
<i>integer</i>	[in] <i>k</i> : cell index in the Z direction
<i>real</i>	[in] prim(neq) : vector of primitive variables
<i>real</i>	[out] s(neq) : vector with source terms'

Definition at line 240 of file sources.f90.

Here is the call graph for this function:



## 4.25 thermal\_cond Module Reference

Adds thermal conduction.

### Functions/Subroutines

- subroutine `init_thermal_cond` ()  
*Initializes Temperature array.*
- subroutine `get_dt_cond` (dt)  
*computes conduction timescale*
- subroutine `progress` (j, tot)  
*Progress bar.*
- real function `ksp` (T)  
*Spitzer conductivity.*
- real function `ksp_parl` (xtemp)  
*Spitzer parallel conductivity.*
- real function `ksp_perp` (xtemp, xdens, B2)  
*Spitzer perpendicular conductivity.*
- subroutine `heatfluxes` ()  
*Returns Heat Fluxes.*
- subroutine `mhd_heatfluxes` ()  
*Returns Heat Fluxes with anisotropic thermal conduction.*
- subroutine `thermal_bounds` ()  
*Exchanges ghost cells for energy only.*
- real function `superstep` (N, snu)  
*Length of superstep.*
- real function `substep` (j, N, nu)  
*Size of substep j.*
- subroutine `st_steps` (fs, Ns, fstep)  
*Returns the number of Supersteps.*
- subroutine `thermal_conduction` ()  
*Upper level wrapper for thermal conduction.*

## Variables

- real, parameter `ph` =0.4  
*Parameter for the sturated regime in McKee.*
- real, parameter `nu` =0.01  
*Super-stepping daMPI\_NBg factor.*
- real, parameter `snu` =sqrt(`nu`)  
*Sqrt of damping factor.*
- integer, parameter `max_iter` = 100  
*Maximum number of iterations.*
- real, parameter `tstep_red_factor` =0.25  
*timestep reduction factor for the conduction*
- real `dt_cond`  
*conduction timestep*
- integer `tc_log`  
*loical unit to write TC log*

### 4.25.1 Detailed Description

Adds a thermal conduction term, affects both the primitive and conserved variables

### 4.25.2 Function/Subroutine Documentation

#### 4.25.2.1 subroutine thermal\_cond::get\_dt\_cond ( real, intent(out) dt )

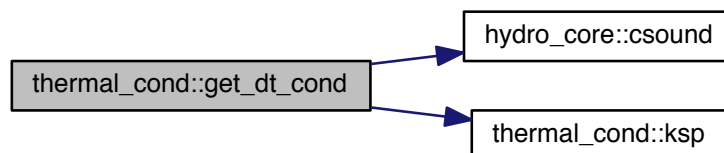
computes conduction timescale (in seconds)

##### Parameters

<i>real</i>	[out] dt :: conduction timescale
-------------	----------------------------------

Definition at line 83 of file thermal\_cond.f90.

Here is the call graph for this function:



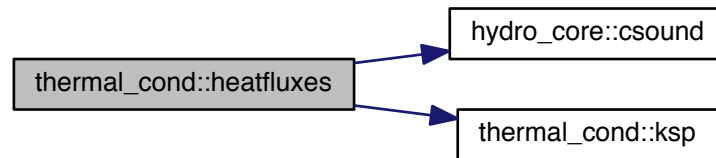
#### 4.25.2.2 subroutine thermal\_cond::heatfluxes ( )

Heat flux, if saturation enabled it takes minimum of the Spitzer and the saturated value

The result is stored in the 5th component of global the F,G,H fluxes (in cgs, conversion is done in dt product)

Definition at line 194 of file thermal\_cond.f90.

Here is the call graph for this function:



#### 4.25.2.3 subroutine thermal\_cond::init\_thermal\_cond ( )

Initializes Temperature array (to resolve dependencies it was moved to the globals module)

Definition at line 55 of file thermal\_cond.f90.

#### 4.25.2.4 real function thermal\_cond::ksp ( real, intent(in) T )

Computes the Spitzer conductivity

Parameters

<i>real</i>	[in] T : temperature [K]
-------------	--------------------------

Definition at line 147 of file thermal\_cond.f90.

#### 4.25.2.5 real function thermal\_cond::ksp\_parl ( real, intent(in) xtemp )

Computes the Spitzer conductivity parallel to B

Parameters

<i>real</i>	[in] T : temperature [K]
-------------	--------------------------

Definition at line 162 of file thermal\_cond.f90.

#### 4.25.2.6 real function thermal\_cond::ksp\_perp ( real, intent(in) xtemp, real, intent(in) xdens, real, intent(in) B2 )

Computes the Spitzer conductivity perpendicular to B

Parameters

<i>real</i>	[in] T : temperature [K]
-------------	--------------------------

Definition at line 177 of file thermal\_cond.f90.

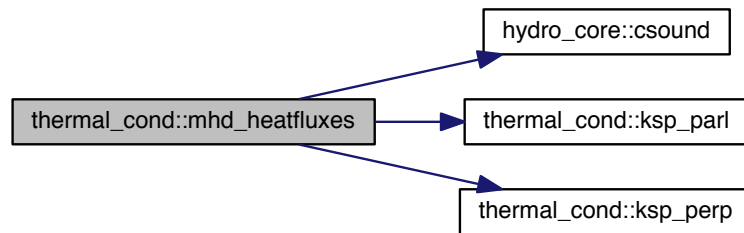
#### 4.25.2.7 subroutine thermal\_cond::mhd\_heatfluxes ( )

Heat flux, if sturation enabled takes minimum of the Spitzer and the saturated value

The result is stored in the 5th component of global the F,G,H fluxes (in cgs, conversion is done in dt product)

Definition at line 285 of file thermal\_cond.f90.

Here is the call graph for this function:



#### 4.25.2.8 subroutine `thermal_cond::progress` ( `integer(kind=4) j`, `integer(kind=4)`, `intent(in) tot` )

Progress bar (only tested with intel Fortran compiler) takes a number between 1 and tot

Parameters

<i>integer</i>	[in] <code>j</code> : current iteration
<i>integer</i>	[in] <code>tot</code> : total number of iterations

Definition at line 125 of file `thermal_cond.f90`.

#### 4.25.2.9 subroutine `thermal_cond::st_steps` ( `real`, `intent(in) fs`, `integer`, `intent(out) Ns`, `real`, `intent(out) fstep` )

Returns the number of Supersteps

Parameters

<i>real</i>	<code>fs</code> : ratio of <code>dtcond/dthydro</code>
<i>integer</i>	<code>Ns</code> : Number of Supersteps
<i>real</i>	<code>fstep</code> : Number of supersteps (float)

Definition at line 674 of file `thermal_cond.f90`.

Here is the call graph for this function:



#### 4.25.2.10 real function `thermal_cond::substep` ( `integer`, `intent(in) j`, `integer`, `intent(in) N`, `real`, `intent(in) nu` )

Returns the size of substep `j` of `N`

**Parameters**

<i>integer</i>	[in] <i>j</i> : index of current step
<i>integer</i>	[in] <i>N</i> : Total number of substeps
<i>real</i>	[in] <i>nu</i> : daMPI_NBg factor

Definition at line 656 of file thermal\_cond.f90.

#### 4.25.2.11 real function thermal\_cond::superstep ( integer *N*, real, intent(in) *snu* )

Returns the length of the superstep with *N* inner substeps

**Parameters**

<i>integer</i>	[in] <i>N</i> : Nunber of inner substeps
<i>real</i>	[in] <i>snu</i> : sqrt of daMPI_NBg factor

Definition at line 635 of file thermal\_cond.f90.

#### 4.25.2.12 subroutine thermal\_cond::thermal\_bounds ( )

Exchanges one layer of boundaries, only the equation that corresponds to the energy

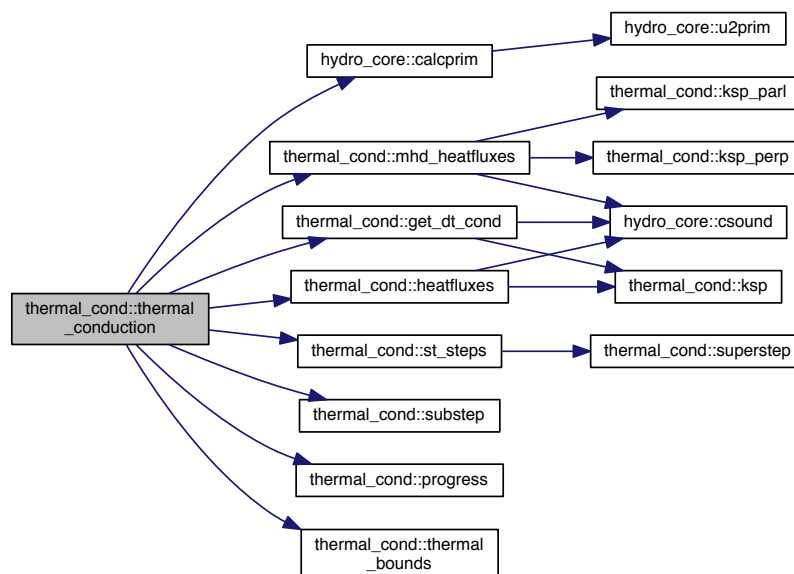
Definition at line 508 of file thermal\_cond.f90.

#### 4.25.2.13 subroutine thermal\_cond::thermal\_conduction ( )

This routine adds the heat conduction, receives the hydro timestep in seconds, and assumes the primitives and Temp(*i,j,k*) arrays are updated

Definition at line 700 of file thermal\_cond.f90.

Here is the call graph for this function:



## Chapter 5

# File Documentation

### 5.1 doc/mainpage.h File Reference

Webpage frontend.

### 5.2 src/boundaries.f90 File Reference

Boundary conditions.

#### Modules

- module [boundaries](#)  
*Boundary conditions.*

#### Functions/Subroutines

- subroutine [boundaries::boundaryi](#) ()  
*Boundary conditions for 1st order half timestep.*
- subroutine [boundaries::boundaryii](#) ()  
*Boundary conditions for 2nd order half timestep.*

#### 5.2.1 Detailed Description

##### Author

Alejandro Esquivel

##### Date

2/Nov/2014

### 5.3 src/chemistry.f90 File Reference

chemistry module

## Modules

- module [chemistry](#)  
*chemistry module*

## Functions/Subroutines

- subroutine [chemistry::update\\_chem](#) ()  
*Advances the chemistry network.*
- subroutine [chemistry::chemstep](#) (y, y0, T, deltt)  
*Advances the chemistry network in one cell.*

### 5.3.1 Detailed Description

#### Author

A. Castellanos, A. Rodriguez, A. Raga and A. Esquivel

#### Date

10/Mar/2016

## 5.4 [src/coldens.f90](#) File Reference

Column density projection.

## Modules

- module [coldens\\_utilities](#)  
*Column density projection.*

## Functions/Subroutines

- subroutine [coldens\\_utilities::init\\_coldens](#) ()  
*Initializes data.*
- subroutine [coldens\\_utilities::read\\_data](#) (u, itprint, filepath)  
*reads data from file*
- subroutine [coldens\\_utilities::getxyz](#) (i, j, k, x, y, z)  
*gets position of a cell*
- subroutine [coldens\\_utilities::rotation\\_x](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the X axis.*
- subroutine [coldens\\_utilities::rotation\\_y](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the Y axis.*
- subroutine [coldens\\_utilities::rotation\\_z](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the Z axis.*
- subroutine [coldens\\_utilities::fill\\_map](#) (nxmap, nymap, u, map, dxT, dyT, theta\_x, theta\_y, theta\_z)  
*Fill target map.*
- subroutine [coldens\\_utilities::write\\_header](#) (unit, nx, ny)  
*Writes header.*
- subroutine [coldens\\_utilities::write\\_map](#) (fileout, nxmap, nymap, map)



*Writes projection to file.*

- program `coldens`

*Computes the H-alpha emission.*

### 5.4.1 Detailed Description

Author

Alejandro Esquivel

Date

2/Nov/2014

### 5.4.2 Function/Subroutine Documentation

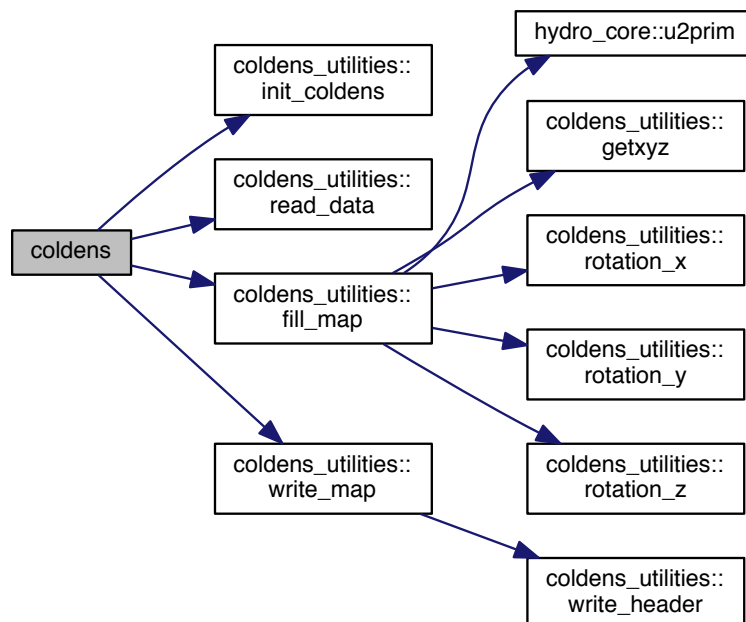
#### 5.4.2.1 program `coldens` ( )

Computes the H-alpha absorption

It rotates the data along each of the coordinates axis by an amount  $\theta_x, \theta_y, \theta_z$ , and projects the map along the the LOS, which is taken to be the Z axis

Definition at line 465 of file `coldens.f90`.

Here is the call graph for this function:



## 5.5 src/constants.f90 File Reference

Constants module.

## Modules

- module `constants`  
*Module containing physical and asronomical constants.*

## Variables

- real, parameter `constants::pi` =acos(-1.)  
 $\pi$
- real, parameter `constants::amh` =1.66e-24  
*hydrogen mass*
- real, parameter `constants::kb` =1.38e-16  
*Boltzmann constant (cgs)*
- real, parameter `constants::rg` =8.3145e7  
*Gas constant (cgs)*
- real, parameter `constants::ggrav` =6.67259e-8  
*Gravitational constant (cgs)*
- real, parameter `constants::clight` =2.99E10  
*speed of light in vacuum (cgs)*
- real, parameter `constants::msun` =1.99E33  
*solar radius (cgs)*
- real, parameter `constants::rsun` =6.955e10  
*solar mass (cgs)*
- real, parameter `constants::mjup` =1.898E30  
*Jupiter mass (cgs)*
- real, parameter `constants::rjup` =7.1492E9  
*Jupiter radius (cgs)*
- real, parameter `constants::au` =1.496e13  
*1AU in cm*
- real, parameter `constants::pc` =3.0857E18  
*1pc in cm*
- real, parameter `constants::kpc` =3.0857E21  
*1Kpc in cm*
- real, parameter `constants::hr` =3600.  
*1hr in seconds*
- real, parameter `constants::day` =86400.  
*1day in seconds*
- real, parameter `constants::yr` =3.1536E7  
*1yr in seconds*
- real, parameter `constants::myr` =3.1536E13  
*1Myr in seconds*

### 5.5.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.6 src/cooling\_chi.f90 File Reference

Cooling module with CHIANTI generated cooling curves.

### Modules

- module `cooling_chi`  
*Cooling module with CHIANTI generated cooling curves.*

### Functions/Subroutines

- subroutine `cooling_chi::read_table` ()  
*Reads the cooling curve table.*
- real(kind=8) function `cooling_chi::coolchi` (T)  
*Returns the cooling coefficient interpolating the table.*
- subroutine `cooling_chi::coolingchi` ()  
*High level wrapper to apply cooling with CHIANTI tables.*

### Variables

- real(kind=8), dimension(2, 41) `cooling_chi::cooltab`

#### 5.6.1 Detailed Description

##### Author

Alejandro Esquivel

##### Date

2/Nov/2014

## 5.7 src/cooling\_dmc.f90 File Reference

Cooling module with Dlgarno Mac Cray coronal cooling curve.

### Modules

- module `cooling_dmc`  
*Cooling module with Dalgarno McCray coronal cooling curve.*

### Functions/Subroutines

- subroutine `cooling_dmc::read_table` ()  
*Reads the cooling curve table.*
- real(kind=8) function `cooling_dmc::cooldmc` (T)  
*Returns the cooling coefficient interpolating the table.*
- subroutine `cooling_dmc::coolingdmc` ()  
*High level wrapper to apply cooling with DMC table.*

## Variables

- `real(kind=8), dimension(2, 41) cooling_dmc::cooltab`

### 5.7.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.8 src/cooling\_h.f90 File Reference

Cooling with hydrogen rate parametrized cooling.

## Modules

- module `cooling_h`  
*Cooling with parametrized cooling and H rate equation.*

## Functions/Subroutines

- subroutine `cooling_h::coolingh` ()  
*High level wrapper to apply cooling.*
- `real(kind=8)` function `cooling_h::alpha` (T)  
*calculates the recombination rate (case B)*
- `real(kind=8)` function `cooling_h::alpha1` (T)  
*calculates the recombination rate to level 1*
- `real(kind=8)` function `cooling_h::colf` (T)  
*calculates the collisional ionization rate*
- `real(kind=8)` function `cooling_h::betah` (T)  
*betaH(T)*
- `real(kind=8)` function `cooling_h::aloss` (X1, X2, DT, DEN, DH0, TE0)  
*Non equilibrium cooling.*
- subroutine `cooling_h::atomic` (dt, uu, tau, radphi)  
*Updates the ionization fraction and applies cooling.*

### 5.8.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.9 src/difrad.f90 File Reference

Diffuse radiation module.

### Modules

- module [difrad](#)  
*Ray tracing Radiative Trasnport.*

### Functions/Subroutines

- subroutine [difrad::init\\_rand](#) ()  
*initializes random number generation*
- subroutine [difrad::emdiff](#) (emax)  
*calculates the diffuse fotoionization emissivity*
- subroutine [difrad::random\\_versor](#) (xd, yd, zd)  
*returns the 3 components of a random versor*
- subroutine [difrad::starsource](#) (srad, x0, y0, z0, x, y, z, xd, yd, zd)  
*Place photon packets at a "star" surface.*
- subroutine [difrad::photons](#) (xl0, yl0, zl0, xd, yd, zd, f)  
*Photon trajectories.*
- subroutine [difrad::radbounds](#) ()  
*follows the rays across MPI boundaries*
- subroutine [difrad::progress](#) (j, tot)  
*Progress bar.*
- subroutine [difrad::diffuse\\_rad](#) ()  
*Diffuse radiation driver.*

### Variables

- real, parameter [difrad::a0](#) =6.3e-18  
*Fotoionization cross section.*
- integer, parameter [difrad::nrays](#) =1000000  
*Number of rays.*
- real, dimension(:, :, :), allocatable [difrad::ph](#)  
*Photoionizing rate.*
- real, dimension(:, :, :), allocatable [difrad::em](#)  
*Photoionizing emissivity.*
- real, dimension(:, :, :), allocatable [difrad::photl](#)  
*Auxiliary buffer for MPI.*
- real, dimension(:, :, :), allocatable [difrad::photr](#)  
*Auxiliary buffer for MPI.*
- real, dimension(:, :, :), allocatable [difrad::photb](#)  
*Auxiliary buffer for MPI.*
- real, dimension(:, :, :), allocatable [difrad::phott](#)  
*Auxiliary buffer for MPI.*
- real, dimension(:, :, :), allocatable [difrad::photo](#)  
*Auxiliary buffer for MPI.*
- real, dimension(:, :, :), allocatable [difrad::photi](#)  
*Auxiliary buffer for MPI.*
- integer, dimension(6) [difrad::buffersize](#)  
*Auxiliary buffer for MPI.*

### 5.9.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.10 src/globals.f90 File Reference

Global variables.

### Modules

- module [globals](#)  
*Module containing global variables.*

### Variables

- real, dimension(:, :, :), allocatable [globals::u](#)  
*conserved variables*
- real, dimension(:, :, :), allocatable [globals::up](#)  
*conserved variables after 1/2 timestep*
- real, dimension(:, :, :), allocatable [globals::primit](#)  
*primitive variables*
- real, dimension(:, :, :), allocatable [globals::f](#)  
*X fluxes.*
- real, dimension(:, :, :), allocatable [globals::g](#)  
*Y fluxes.*
- real, dimension(:, :, :), allocatable [globals::h](#)  
*Z fluxes.*
- real [globals::dx](#)  
*grid spacing in X*
- real [globals::dy](#)  
*grid spacing in Y*
- real [globals::dz](#)  
*grid spacing in Z*
- integer, dimension(0:2) [globals::coords](#)  
*position of neighboring MPI blocks*
- integer [globals::left](#)  
*MPI neighbor in the -x direction.*
- integer [globals::right](#)  
*MPI neighbor in the +x direction.*
- integer [globals::top](#)  
*MPI neighbor in the -y direction.*
- integer [globals::bottom](#)  
*MPI neighbor in the +y direction.*
- integer [globals::out](#)  
*MPI neighbor in the -z direction.*

- integer [globals::in](#)  
*MPI neighbor in the +z direction.*
- integer [globals::rank](#)  
*MPI rank.*
- integer [globals::comm3d](#)  
*Cartesian MPI communicator.*
- real [globals::time](#)  
*Current time.*
- real [globals::dt\\_cfl](#)  
*Current CFL \$ t\$.*
- integer [globals::currentiteration](#)  
*Current iteration.*
- real, dimension(:, :, :), allocatable [globals::temp](#)  
*Temperature array [K].*

### 5.10.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.11 src/h\_alpha\_proj.f90 File Reference

H alpha projection.

### Modules

- module [h\\_alpha\\_utilities](#)  
*H alpha projection.*

### Functions/Subroutines

- subroutine [h\\_alpha\\_utilities::init\\_ha](#) ()  
*Initializes data.*
- subroutine [h\\_alpha\\_utilities::read\\_data](#) (u, itprint, filepath)  
*reads data from file*
- subroutine [h\\_alpha\\_utilities::getxyz](#) (i, j, k, x, y, z)  
*gets position of a cell*
- subroutine [h\\_alpha\\_utilities::rotation\\_x](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the X axis.*
- subroutine [h\\_alpha\\_utilities::rotation\\_y](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the Y axis.*
- subroutine [h\\_alpha\\_utilities::rotation\\_z](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the Z axis.*
- subroutine [h\\_alpha\\_utilities::fill\\_map](#) (nxmap, nymap, u, map, dxT, dyT, theta\_x, theta\_y, theta\_z)  
*Fill target map.*

- subroutine `h_alpha_utilities::write_ha` (fileout, nxmap, nymap, map)  
*Writes projection to file.*
- subroutine `h_alpha_utilities::write_rg` (fileout, nxmap, nymap, map)  
*Writes projection to file in rg format.*
- program `h_alpha_proj`  
*Computes the H-alpha emission.*

### 5.11.1 Detailed Description

Author

Alejandro Esquivel

Date

2/Nov/2014

### 5.11.2 Function/Subroutine Documentation

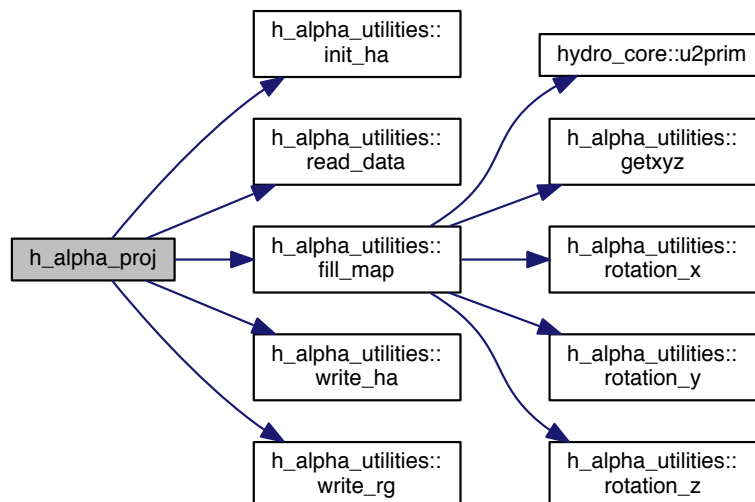
#### 5.11.2.1 program `h_alpha_proj` ( )

Computes the H-alpha absorption

It rotates the data along each of the coordinates axis by an amount  $\theta_x, \theta_y, \theta_z$ , and projects the map along the the LOS, which is taken to be the Z axis

Definition at line 428 of file `h_alpha_proj.f90`.

Here is the call graph for this function:



## 5.12 src/hll.f90 File Reference

HLL approximate Riemann solver module.



## Modules

- module [hll](#)  
*HLL approximate Riemann solver module.*

## Functions/Subroutines

- subroutine [hll::prim2fhll](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLL solver.*
- subroutine [hll::hllfluxes](#) (choice)  
*Calculates HLL fluxes from the primitive variables on all the domain.*

### 5.12.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.13 src/hllc.f90 File Reference

HLLC approximate Riemann solver module.

## Modules

- module [hllc](#)  
*HLLC approximate Riemann solver module.*

## Functions/Subroutines

- subroutine [hllc::prim2fhllc](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLLC solver.*
- subroutine [hllc::hllcfluxes](#) (choice)  
*Calculates HLLC fluxes from the primitive variables on all the domain.*

### 5.13.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.14 src/hlld.f90 File Reference

HLLD approximate Riemann solver module.

## Modules

- module [hlld](#)  
*HLLD approximate Riemann solver module.*

## Functions/Subroutines

- subroutine [hlld::prim2fhlld](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLLD solver.*
- subroutine [hlld::hlldfluxes](#) (choice)  
*Calculates HLLD fluxes from the primitive variables on all the domain.*

### 5.14.1 Detailed Description

#### Author

C. Villarreal D'Angelo, A. Esquivel, M. Schneiter

#### Date

2/Nov/2014

## 5.15 [src/hlle.f90](#) File Reference

HLLE approximate Riemann solver module.

## Modules

- module [hlle](#)  
*HLLE approximate Riemann solver module.*

## Functions/Subroutines

- subroutine [hlle::prim2fhlle](#) (priml, primr, ff)  
*Solves the Riemann problem at the interface PL,PR using the HLLE solver.*
- subroutine [hlle::hllefluxes](#) (choice)  
*Calculates HLLE fluxes from the primitive variables on all the domain.*

### 5.15.1 Detailed Description

#### Author

C. Villarreal D'Angelo, A. Esquivel, M. Schneiter

#### Date

2/Nov/2014

## 5.16 [src/hydro\\_core.f90](#) File Reference

Hydrodynamical and Magnetohydrodynamical basic module.

## Modules

- module [hydro\\_core](#)  
*Basic hydro (and MHD) subroutines utilities.*

## Functions/Subroutines

- subroutine [hydro\\_core::u2prim](#) (uu, prim, T)  
*Computes the primitive variables and temperature from conserved variables on a single cell.*
- subroutine [hydro\\_core::calcprim](#) (u, primit, only\_ghost)  
*Updated the primitives, using the conserved variables in the entire domain.*
- subroutine [hydro\\_core::prim2u](#) (prim, uu)  
*Computes the conserved conserved variables from the primitives in a single cell.*
- subroutine [hydro\\_core::prim2f](#) (prim, ff)  
*Computes the Euler Fluxes in one cell.*
- subroutine [hydro\\_core::swapy](#) (var, neq)  
*Swaps the x and y components in a cell.*
- subroutine [hydro\\_core::swapz](#) (var, neq)  
*Swaps the x and z components in a cell.*
- subroutine [hydro\\_core::csound](#) (p, d, cs)  
*Computes the sound speed.*
- subroutine [hydro\\_core::cfast](#) (p, d, bx, by, bz, cfx, cfy, cfz)  
*Computes the fast magnetosonic speeds in the 3 coordinates.*
- subroutine [hydro\\_core::cfastx](#) (prim, cfX)  
*Computes the fast magnetosonic speed in the x direction.*
- subroutine [hydro\\_core::get\\_timestep](#) (current\_iter, n\_iter, current\_time, tprint, dt, dump\_flag)  
*Obtains the timestep allowed by the CFL condition in the entire.*
- subroutine [hydro\\_core::limiter](#) (PLL, PL, PR, PRR, neq)  
*Performs a linear reconstruction of the primitive variables.*
- real function **average** (a, b)

### 5.16.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.17 src/hydro\_solver.f90 File Reference

Hydrodynamical and Magnetohydrodynamocal solver module.

## Modules

- module [hydro\\_solver](#)  
*Advances the simulation one timestep.*

## Functions/Subroutines

- subroutine `hydro_solver::viscosity` ()  
*Adds artificial viscosity to the conserved variables.*
- subroutine `hydro_solver::step` (dt)  
*Upwind timestep.*
- subroutine `hydro_solver::tstep` ()  
*High level wrapper to advance the simulation.*

### 5.17.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.18 `src/init.f90` File Reference

Guacho-3D initialization module.

## Modules

- module `init`  
*Guacho-3D initialization.*

## Functions/Subroutines

- subroutine `init::initmain` (tprint, itprint)  
*Main initialization routine.*
- subroutine `init::initflow` (itprint)  
*Initializes the conserved variables, in the globals module.*

### 5.18.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.19 `src/linear_system.f90` File Reference

linear system inversion module

## Modules

- module [linear\\_system](#)  
*linear system inversion module*

## Functions/Subroutines

- subroutine [linear\\_system::ludcmp](#) (a, n, indx, d)  
*LU decomposition.*
- subroutine [linear\\_system::lubksb](#) (a, n, indx, b)  
*Solves a set of linear equations.*
- subroutine [linear\\_system::linsys](#) (a, b, n)  
*Driver to solves a set of linear equations.*

### 5.19.1 Detailed Description

#### Author

A. Castellanos, A. Rodriguez, A. Raga and A. Esquivel

#### Date

10/Mar/2016

## 5.20 src/lyman\_alpha\_tau.f90 File Reference

Lyman\_alpha\_utilities.

## Modules

- module [lyman\\_alpha\\_utilities](#)  
*Lyman\_alpha\_utilities.*

## Functions/Subroutines

- subroutine [lyman\\_alpha\\_utilities::init\\_la](#) ()  
*Initializes data.*
- subroutine [lyman\\_alpha\\_utilities::read\\_data](#) (u, itprint, filepath)  
*reads data from file*
- subroutine [lyman\\_alpha\\_utilities::getxyz](#) (i, j, k, x, y, z)  
*gets position of a cell*
- subroutine [lyman\\_alpha\\_utilities::rotation\\_x](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the X axis.*
- subroutine [lyman\\_alpha\\_utilities::rotation\\_y](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the Y axis.*
- subroutine [lyman\\_alpha\\_utilities::rotation\\_z](#) (theta, x, y, z, xn, yn, zn)  
*Rotation around the Z axis.*
- subroutine [lyman\\_alpha\\_utilities::fill\\_map](#) (nxmap, nymap, nvmap, vmin, vmax, u, map, dxT, dyT, theta\_x, theta\_y, theta\_z)  
*Fill target map.*

- subroutine `lyman_alpha_utilities::write_la` (itprint, filepath, nxmap, nymap, nvmap, map)  
*Writes projection to file.*
- subroutine `lyman_alpha_utilities::phigauss` (T, vzn, vmin, vmax, nvmap, profile)  
*This routine computes a gaussian line profile.*
- program `lyman_alpha_tau`  
*Computes the Ly-alpha absorption.*

### 5.20.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

### 5.20.2 Function/Subroutine Documentation

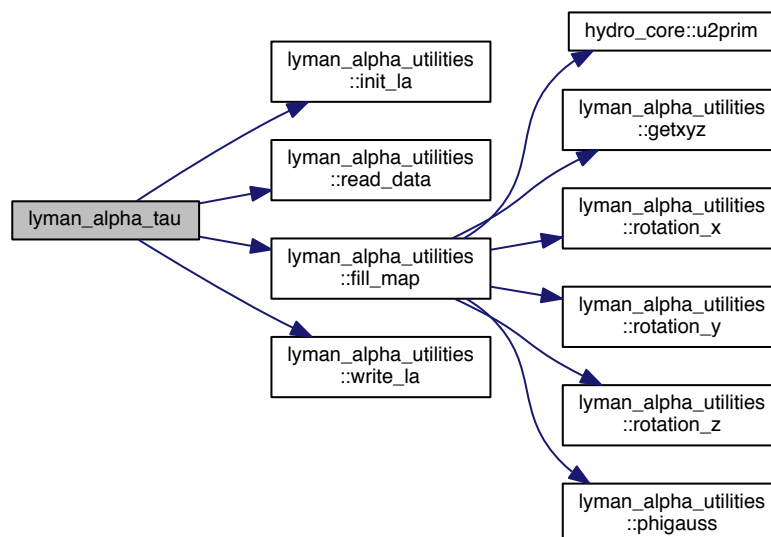
#### 5.20.2.1 program `lyman_alpha_tau` ( )

Computes the Ly-alpha absorption

It rotates the data along each of the coordinates axis by an amount  $\theta_x, \theta_y, \theta_z$ , and the LOS is along the Z axis

Definition at line 419 of file `lyman_alpha_tau.f90`.

Here is the call graph for this function:



### 5.21 src/main.f90 File Reference

Guacho-3D main program.

## Functions/Subroutines

- program [guacho](#)

*Guacho-3D Main Program This is the main program unit of the Guacho-3D code.*

*The code integrates Euler equations in three dimensions, the choice of the integration method is set in the makefile.*

*The flow (conserved) variables are taken to be:*

*ieq=*

*1 : rho (total)*

*2 : rho u*

*3 : rho v*

*4 : rho w*

*5 : Internal energy (thermal+kinetic)*

*6 : bx (optional, if MHD or PMHD)*

*7 : by (optional, if MHD or PMHD)*

*8 : bz (optional, if MHD or PMHD)*

*additional variables advected into the flow, e.g.:*

*9 (6): n\_HI*

*10 (7): n\_HII*

*11 (8): n\_HeI*

*12 (9): n\_HeII*

*13 (10): n\_HeIII*

*14 (11): rho\*zbar*

*15 (12): ne*

*This can be changed by the user according to cooling function for instance.*

### 5.21.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.22 src/Out\_BIN\_Module.f90 File Reference

Output in BIN Format.

## Modules

- module [out\\_bin\\_module](#)

*Output in BIN format.*

## Functions/Subroutines

- subroutine [out\\_bin\\_module::write\\_header](#) (unit, neq\_out, nghost\_out)

*Writes header.*

- subroutine [out\\_bin\\_module::write\\_bin](#) (itprint)

*Writes Data, one file per processor.*

### 5.22.1 Detailed Description

**Author**

Alejandro Esquivel

**Date**

2/Nov/2014

## 5.23 src/Out\_Silo\_Module.f90 File Reference

Output in Silo Format.

**Modules**

- module [out\\_silo\\_module](#)  
*Output in Silo (+HDF5) Format.*

**Functions/Subroutines**

- subroutine [out\\_silo\\_module::writeblocks](#) (itprint)  
*Writes Data, one file per processor.*
- subroutine [out\\_silo\\_module::writemaster](#) (itprint)  
*Writes the Master File.*
- subroutine [out\\_silo\\_module::outputsilo](#) (itprint)  
*Upper level wrapper.*

### 5.23.1 Detailed Description

**Author**

Alejandro Esquivel

**Date**

2/Nov/2014

## 5.24 src/Out\_VTK\_Module.f90 File Reference

Output in VTK Format.

**Modules**

- module [out\\_vtk\\_module](#)  
*Output in VTK format.*

**Functions/Subroutines**

- subroutine [out\\_vtk\\_module::write\\_vtk](#) (itprint)  
*Writes Data, one file per processor.*



### 5.24.1 Detailed Description

**Author**

Alejandro Esquivel

**Date**

2/Nov/2014

## 5.25 src/output.f90 File Reference

Writes Output.

**Modules**

- module [output](#)  
*Writes output.*

**Functions/Subroutines**

- subroutine [output::write\\_output](#) (itprint)  
*Writes output.*

### 5.25.1 Detailed Description

**Author**

Alejandro Esquivel

**Date**

2/Nov/2014

## 5.26 src/sources.f90 File Reference

Adds source terms.

**Modules**

- module [sources](#)  
*Adds source terms.*

**Functions/Subroutines**

- subroutine [sources::getpos](#) (i, j, k, x, y, z, r)  
*Gets position in the grid.*
- subroutine [sources::grav\\_source](#) (xc, yc, zc, pp, s)  
*Gravity due to point sources.*
- subroutine [sources::radpress\\_source](#) (i, j, k, xc, yc, zc, rc, pp, s)

- *Radiation pressure force.*
- subroutine [sources::divergence\\_b](#) (i, j, k, d)  
*Computes  $\text{div}(B)$*
- subroutine [sources::divbcorr\\_source](#) (i, j, k, pp, s)  
*8 Wave source terms for  $\text{div}(B)$  correction*
- subroutine [sources::source](#) (i, j, k, prim, s)  
*Upper level wrapper for sources.*

### 5.26.1 Detailed Description

#### Author

Alejandro Esquivel

#### Date

2/Nov/2014

## 5.27 src/thermal\_cond.f90 File Reference

Thermal conduction module.

### Modules

- module [thermal\\_cond](#)  
*Adds thermal conduction.*

### Functions/Subroutines

- subroutine [thermal\\_cond::init\\_thermal\\_cond](#) ()  
*Initializes Temperature array.*
- subroutine [thermal\\_cond::get\\_dt\\_cond](#) (dt)  
*computes conduction timescale*
- subroutine [thermal\\_cond::progress](#) (j, tot)  
*Progress bar.*
- real function [thermal\\_cond::ksp](#) (T)  
*Spitzer conductivity.*
- real function [thermal\\_cond::ksp\\_parl](#) (xtemp)  
*Spitzer parallel conductivity.*
- real function [thermal\\_cond::ksp\\_perp](#) (xtemp, xdens, B2)  
*Spitzer perpendicular conductivity.*
- subroutine [thermal\\_cond::heatfluxes](#) ()  
*Returns Heat Fluxes.*
- subroutine [thermal\\_cond::mhd\\_heatfluxes](#) ()  
*Returns Heat Fluxes with anisotropic thermal conduction.*
- subroutine [thermal\\_cond::thermal\\_bounds](#) ()  
*Exchanges ghost cells for energy only.*
- real function [thermal\\_cond::superstep](#) (N, snu)  
*Length of superstep.*
- real function [thermal\\_cond::substep](#) (j, N, nu)

*Size of substep  $j$ .*

- subroutine `thermal_cond::st_steps` (fs, Ns, fstep)

*Returns the number of Supersteps.*

- subroutine `thermal_cond::thermal_conduction` ()

*Upper level wrapper for thermal conduction.*

## Variables

- real, parameter `thermal_cond::ph` =0.4

*Parameter for the sturated regime in McKee.*

- real, parameter `thermal_cond::nu` =0.01

*Super-stepping daMPI\_NBg factor.*

- real, parameter `thermal_cond::snu` =sqrt(nu)

*Sqrt of damping factor.*

- integer, parameter `thermal_cond::max_iter` = 100

*Maximum number of iterations.*

- real, parameter `thermal_cond::tstep_red_factor` =0.25

*timestep reduction factor for the conduction*

- real `thermal_cond::dt_cond`

*conduction timestep*

- integer `thermal_cond::tc_log`

*loical unit to write TC log*

### 5.27.1 Detailed Description

#### Author

Alejandro Esquivel & Ernesto Zurbiggen

#### Date

07/Sep/2015



# Index

- aloss
  - cooling\_h, [17](#)
- alpha
  - cooling\_h, [17](#)
- alpha1
  - cooling\_h, [18](#)
- atomic
  - cooling\_h, [18](#)
- betah
  - cooling\_h, [18](#)
- boundaries, [7](#)
  - boundaryi, [7](#)
  - boundaryii, [7](#)
- boundaryi
  - boundaries, [7](#)
- boundaryii
  - boundaries, [7](#)
- calcprim
  - hydro\_core, [35](#)
- cfast
  - hydro\_core, [36](#)
- cfastx
  - hydro\_core, [36](#)
- chemistry, [7](#)
  - chemstep, [8](#)
  - update\_chem, [8](#)
- chemstep
  - chemistry, [8](#)
- coldens
  - coldens.f90, [63](#)
- coldens.f90
  - coldens, [63](#)
- coldens\_utilities, [9](#)
  - fill\_map, [9](#)
  - getxyz, [10](#)
  - init\_coldens, [10](#)
  - read\_data, [10](#)
  - rotation\_x, [11](#)
  - rotation\_y, [11](#)
  - rotation\_z, [11](#)
  - write\_header, [12](#)
  - write\_map, [12](#)
- colf
  - cooling\_h, [19](#)
- constants, [12](#)
- coolchi
  - cooling\_chi, [14](#)
- cooldmc
  - cooling\_dmc, [16](#)
- cooling\_chi, [13](#)
  - coolchi, [14](#)
  - coolingchi, [15](#)
  - read\_table, [15](#)
- cooling\_dmc, [15](#)
  - cooldmc, [16](#)
  - coolingdmc, [16](#)
  - read\_table, [16](#)
- cooling\_h, [16](#)
  - aloss, [17](#)
  - alpha, [17](#)
  - alpha1, [18](#)
  - atomic, [18](#)
  - betah, [18](#)
  - colf, [19](#)
  - coolingh, [19](#)
- coolingchi
  - cooling\_chi, [15](#)
- coolingdmc
  - cooling\_dmc, [16](#)
- coolingh
  - cooling\_h, [19](#)
- csound
  - hydro\_core, [36](#)
- diffuse\_rad
  - difrad, [20](#)
- difrad, [19](#)
  - diffuse\_rad, [20](#)
  - emdiff, [21](#)
  - init\_rand, [21](#)
  - photons, [21](#)
  - progress, [22](#)
  - radbounds, [22](#)
  - random\_versor, [22](#)
  - starsource, [22](#)
- divbcorr\_source
  - sources, [53](#)
- divergence\_b
  - sources, [54](#)
- doc/mainpage.h, [61](#)
- emdiff
  - difrad, [21](#)
- fill\_map
  - coldens\_utilities, [9](#)
  - h\_alpha\_utilities, [25](#)
  - lyman\_alpha\_utilities, [45](#)

- get\_dt\_cond
  - thermal\_cond, 57
- get\_timestep
  - hydro\_core, 36
- getpos
  - sources, 54
- getxyz
  - coldens\_utilities, 10
  - h\_alpha\_utilities, 26
  - lyman\_alpha\_utilities, 46
- globals, 23
- grav\_source
  - sources, 54
- h\_alpha\_proj
  - h\_alpha\_proj.f90, 70
- h\_alpha\_proj.f90
  - h\_alpha\_proj, 70
- h\_alpha\_utilities, 24
  - fill\_map, 25
  - getxyz, 26
  - init\_ha, 27
  - read\_data, 27
  - rotation\_x, 27
  - rotation\_y, 27
  - rotation\_z, 28
  - write\_ha, 28
  - write\_rg, 28
- heatfluxes
  - thermal\_cond, 57
- hll, 28
  - hllfluxes, 29
  - prim2fhll, 29
- hllc, 30
  - hllcfluxes, 30
  - prim2fhllc, 31
- hllcfluxes
  - hllc, 30
- hlld, 31
  - hlldfluxes, 32
  - prim2fhlld, 32
- hlldfluxes
  - hlld, 32
- hlle, 33
  - hllefluxes, 33
  - prim2fhlle, 34
- hllefluxes
  - hlle, 33
- hllfluxes
  - hll, 29
- hydro\_core, 34
  - calcprim, 35
  - cfast, 36
  - cfastx, 36
  - csound, 36
  - get\_timestep, 36
  - limiter, 37
  - prim2f, 37
  - prim2u, 37
  - swapy, 39
  - swapz, 39
  - u2prim, 39
- hydro\_solver, 39
  - step, 40
  - tstep, 40
  - viscosity, 41
- init, 42
  - initflow, 42
  - initmain, 42
- init\_coldens
  - coldens\_utilities, 10
- init\_ha
  - h\_alpha\_utilities, 27
- init\_la
  - lyman\_alpha\_utilities, 46
- init\_rand
  - difrad, 21
- init\_thermal\_cond
  - thermal\_cond, 58
- initflow
  - init, 42
- initmain
  - init, 42
- ksp
  - thermal\_cond, 58
- ksp\_parl
  - thermal\_cond, 58
- ksp\_perp
  - thermal\_cond, 58
- limiter
  - hydro\_core, 37
- linear\_system, 43
  - linsys, 43
  - lubksb, 44
  - ludcmp, 44
- linsys
  - linear\_system, 43
- lubksb
  - linear\_system, 44
- ludcmp
  - linear\_system, 44
- lyman\_alpha\_tau
  - lyman\_alpha\_tau.f90, 76
- lyman\_alpha\_tau.f90
  - lyman\_alpha\_tau, 76
- lyman\_alpha\_utilities, 44
  - fill\_map, 45
  - getxyz, 46
  - init\_la, 46
  - phigauss, 46
  - read\_data, 47
  - rotation\_x, 47
  - rotation\_y, 47
  - rotation\_z, 47
  - write\_la, 48

- mhd\_heatfluxes
  - thermal\_cond, 58
- out\_bin\_module, 48
  - write\_bin, 48
  - write\_header, 49
- out\_silo\_module, 49
  - outputsilo, 49
  - writeblocks, 50
  - writemaster, 50
- out\_vtk\_module, 50
  - write\_vtk, 51
- output, 52
  - write\_output, 52
- outputsilo
  - out\_silo\_module, 49
- phigauss
  - lyman\_alpha\_utilities, 46
- photons
  - difrad, 21
- prim2f
  - hydro\_core, 37
- prim2fhll
  - hll, 29
- prim2fhllc
  - hllc, 31
- prim2fhlld
  - hlld, 32
- prim2fhlle
  - hlle, 34
- prim2u
  - hydro\_core, 37
- progress
  - difrad, 22
  - thermal\_cond, 59
- radbounds
  - difrad, 22
- radpress\_source
  - sources, 55
- random\_versor
  - difrad, 22
- read\_data
  - coldens\_utilities, 10
  - h\_alpha\_utilities, 27
  - lyman\_alpha\_utilities, 47
- read\_table
  - cooling\_chi, 15
  - cooling\_dmc, 16
- rotation\_x
  - coldens\_utilities, 11
  - h\_alpha\_utilities, 27
  - lyman\_alpha\_utilities, 47
- rotation\_y
  - coldens\_utilities, 11
  - h\_alpha\_utilities, 27
  - lyman\_alpha\_utilities, 47
- rotation\_z
  - coldens\_utilities, 11
  - h\_alpha\_utilities, 28
  - lyman\_alpha\_utilities, 47
- source
  - sources, 55
- sources, 53
  - divbcorr\_source, 53
  - divergence\_b, 54
  - getpos, 54
  - grav\_source, 54
  - radpress\_source, 55
  - source, 55
- src/Out\_BIN\_Module.f90, 77
- src/Out\_Silo\_Module.f90, 78
- src/Out\_VTK\_Module.f90, 78
- src/boundaries.f90, 61
- src/chemistry.f90, 61
- src/coldens.f90, 62
- src/constants.f90, 63
- src/cooling\_chi.f90, 65
- src/cooling\_dmc.f90, 65
- src/cooling\_h.f90, 66
- src/difrad.f90, 67
- src/globals.f90, 68
- src/h\_alpha\_proj.f90, 69
- src/hll.f90, 70
- src/hllc.f90, 71
- src/hlld.f90, 71
- src/hlle.f90, 72
- src/hydro\_core.f90, 72
- src/hydro\_solver.f90, 73
- src/init.f90, 74
- src/linear\_system.f90, 74
- src/lyman\_alpha\_tau.f90, 75
- src/main.f90, 76
- src/output.f90, 79
- src/sources.f90, 79
- src/thermal\_cond.f90, 80
- st\_steps
  - thermal\_cond, 59
- starsource
  - difrad, 22
- step
  - hydro\_solver, 40
- substep
  - thermal\_cond, 59
- superstep
  - thermal\_cond, 60
- swapy
  - hydro\_core, 39
- swapz
  - hydro\_core, 39
- thermal\_bounds
  - thermal\_cond, 60
- thermal\_cond, 56
  - get\_dt\_cond, 57
  - heatfluxes, 57

- init\_thermal\_cond, 58
- ksp, 58
- ksp\_parl, 58
- ksp\_perp, 58
- mhd\_heatfluxes, 58
- progress, 59
- st\_steps, 59
- substep, 59
- superstep, 60
- thermal\_bounds, 60
- thermal\_conduction, 60
- thermal\_conduction
  - thermal\_cond, 60
- tstep
  - hydro\_solver, 40
- u2prim
  - hydro\_core, 39
- update\_chem
  - chemistry, 8
- viscosity
  - hydro\_solver, 41
- write\_bin
  - out\_bin\_module, 48
- write\_ha
  - h\_alpha\_utilities, 28
- write\_header
  - coldens\_utilities, 12
  - out\_bin\_module, 49
- write\_la
  - lyman\_alpha\_utilities, 48
- write\_map
  - coldens\_utilities, 12
- write\_output
  - output, 52
- write\_rg
  - h\_alpha\_utilities, 28
- write\_vtk
  - out\_vtk\_module, 51
- writeblocks
  - out\_silo\_module, 50
- writemaster
  - out\_silo\_module, 50