PB3D

[2.45]

Chapter 1

Welcome to PB3D

The main presentation of the PB3D project information can be found at the `PB3D home`*.*

Be sure to also consult the `html version` ***of this document.***

This page contains the technical documentation of the **PB3D** project. A pdf version can be downloaded `here`.

It contains a general overview of the code in General code structure, which is much more detailed than what can be found in [17].

To get started, for installation instructions, please see Installation instructions.

Also, in Tutorial, you can find a hands-on tutorial on running the code.

Furthermore, in Input variables, the multitude of input variables is discussed.

Additionally, in Code outputs, the same is done for the output that can be produced, including the output by POST.

For a list of frequently asked questions and frequently encountered problems, consult the Frequently Asked Questions.

Finally, the various modules of which the program consists can be explored in detail in the appendices pages, as well as the procedures and the variables.

—this documentation was created with `Doxygen 1.8.17`*—*

Chapter 2

Input variables

Footnotes are situated at the end of the chapter

This page describes the various input variables that PB3D and POST can take.

Inputs are generally done through the input file, but there are some inputs that are provided on runtime, using the command-line with `--[option_name]` format.

See also

See init_files().

For both PB3D and POST, both types are discussed.

Some general remarks:

- There is some fault detection built into the routines. These distinguish between minor faults, that trigger a warning but do not stop execution, and major faults, that trigger an error that stop execution.

  See also

  See writo().
- The data type `real` refers to double precision real, defined by num_vars.dp. For integers, standard precision is used.
- There are hard-coded `min_tol` and `max_tol` in read_input_opts(), which are used to limit the tolerances `tol_rich`, `tol_zero` and `tol_SLEPC`. This could be changed if needed.
- If `eq_style = 2` (i.e. HELENA), the total maximum memory available must be enough to perform the first Richardson level in a single equilibrium job (see General code structure).

2.1 PB3D Inputs

2.1.1 Input file

Table 2.1 Table 1. PB3D input file

input parameter	explanation	default value	data type	note
concerning solution				
`n_r_sol`	number of points in solution grid	100	`int`	
`min_r_sol`	minimum normalized flux of computational domain $(0 \dots 1)$	0.1	`real`	(1)
`max_r_sol`	maximum normalized flux of computational domain $(0 \dots 1)$	1.0	`real`	(1)
`tol_norm`	tolerance for normal range	0.05	`real`	(2)
`EV_style`	style for EV calculation: 1 — SLEPC solver	1	`int`	
concerning field line				
`alpha_style`	style for alpha: 1 — one field line, many turns 2 — many field lines, one turn	2 (VMEC), 1 (HELE↩ NA)	`int`	
`n_alpha`	number of field lines for `alpha_style` 2	10 (VMEC)	`int`	
`alpha`	field line label $[\pi]$ for `alpha_style` 1	0	`real`	
`min_alpha`	minimum field line label α $[\pi]$ for `alpha_`↩ `style` 2	0	`int`	
`max_alpha`	maximum field line label α $[\pi]$ for `alpha_style` 2	2	`int`	
concerning perturbation				
`min_n_par_X`	minimum number of parallel points for integration	20	`int`	(3)
`min_par_X`	minimum parallel angle $[\pi]$ for integration	-4	`int`	
`max_par_X`	maximum parallel angle $[\pi]$ for integration	4	`int`	

input parameter	explanation	default value	data type	note
prim_X	primary mode number in geodesic direction	20	int	(4)
min_sec_X	minimum secondary mode number in parallel direction		int	(4, 5)
max_sec_X	maximum secondary mode number in parallel direction		int	(4, 5)
n_mod_X	number of secondary modes in parallel direction	20	int	(4, 5)
use_pol_flux_F	on which flux to base the normal coordinate. .true. : rescaled poloidal flux $\frac{\Psi_{\text{pol}}}{2\pi}$ .false. : rescaled toroidal flux $\frac{\Psi_{\text{tor}}}{2\pi}$	.true.	log	(35)
concerning normalization				
rho_0	Normalization factor for density $\rho\ \frac{kg}{m^3}$	10^{-7}	real	
R_0	Normalization factor for length scale $R\,m$		real	(6)
pres_0	Normalization factor for pressure $p\,Pa$		real	(6)
psi_0	Normalization factor for flux $\Psi\,Tm^2$		real	(6)
B_0	Normalization factor for magnetic field $\vec{B}$ T		real	(6)
T_0	Normalization factor for time s		real	(6)
concerning input and output				
n_sol_requested	number of solutions requested	1	int	

input parameter	explanation	default value	data type	note
`retain_all_sol`	retain all solutions `.true.` un-physical eigenvalues are also retained `.false.` un-physical eigenvalues are left out	2	`log`	(7)
`plot_resonance`	plot the safety factor (poloidal flux) or rotational transform (toroidal flux) and the resonant values	`.false.`	`log`	(13)
`plot_magn_grid`	plot magnetic field lines in the flux surfaces in 3-D geometry as an animation	`.false.`	`log`	(14)
`plot_B`	plot magnetic field $\vec{B} = \nabla\alpha \times \nabla\psi\,[T]$	`.false.`	`log`	(15)
`plot_J`	plot current $\vec{J} = \mu_0^{-1}\nabla \times (\nabla\alpha \times \nabla\psi)\left[\frac{A}{m^2}\right]$	`.false.`	`log`	(16)
`plot_kappa`	plot curvature $\vec{\kappa} = \frac{\vec{B}}{B} \cdot \nabla\frac{\vec{B}}{B}\left[\frac{1}{m}\right]$ and its components $\kappa_n = \frac{\nabla\psi}{\lvert\nabla\psi\rvert^2} \cdot \vec{\kappa}\left[\frac{1}{Tm^2}\right]$ and $\kappa_g = \frac{\nabla\psi\times\vec{B}}{B^2} \cdot \vec{\kappa}\,[\,]$	`.false.`	`log`	(17)
`plot_flux_q`	plot flux quantities q, ι, $p\,[Pa]$, $\Psi_{\text{pol}}\left[Tm^2\right]$ and $\Psi_{\text{tor}}\left[Tm^2\right]$	`.false.`	`log`	(18)
`n_theta_plot`	number of points in plots in θ direction		`int`	(19)
`n_zeta_plot`	number of points in plots in ζ direction		`int`	(19)
`min_theta_plot`	minimum of θ range for plots	1	`int`	
`max_theta_plot`	minimum of θ range for plots	3	`int`	
`min_zeta_plot`	minimum of ζ range for plots		`int`	(20)
`max_zeta_plot`	minimum of ζ range for plots		`int`	(20)

input parameter	explanation	default value	data type	note
plot_size	size of 2-D plots in inch by inch	$[10, 5]$	int(2)	
ex_plot_style	style how external plots are made 1: GnuPlot 2: Bokeh for 2-D and Mayavi for 3-D	1	int	(21)
concerning Richardson extrapolation				
max_it_rich	maximum number of Richardson extrapolations	1	int	
tol_rich	tolerance for convergence of Richardson extrapolations	1^{-4}	real	
rich_restart_lvl	Richardson level at which to restart a previous simulation	1	int	(8)
concerning finding zeros				
max_it_zero	maximum number of iterations in num_ops.calc_zero_hh() and calc_zero_zhang()	100	int	(9)
tol_zero	tolerance for zero finding	1^{-10}	real	(9)
max_nr_backtracks_HH	maximum number of times the correction can be relaxed by half in a backtrack for the Householder methods	20	int	(9)
concerning runtime				
use_normalization	use normalization for the internal calculations	.true.	log	(6)
max_tot_mem	maximum total memory available to the simulation $[kB]$	6000	real	
sol_n_procs	Number of processes available for SLEPC solution $(1 \dots N)$ where N is the number of MPI processes	N	int	
norm_disc_prec_eq	normal discretization precision for equilibrium quantities	3	int	

input parameter	explanation	default value	data type	note
`norm_disc_prec_X`	normal discretization precision for perturbation quantities	3	`int`	
`norm_disc_prec_sol`	normal discretization precision for solution quantities	3	`int`	
`norm_disc_style_sol`	style how solution quantities are discretized 1: central finite differences 2: left finite differences	2	`int`	
`magn_int_style`	style how magnetic integrals are calculated 1: trapezoidal rule 2: Simpson 3/8 rule	1	`int`	
`X_grid_style`	style how perturbation grid is set up 1: identical to equilibrium grid 2: identical to solution grid 3: enriched version of equilibrium grid (see `max_njq_change`)	1 (`X_style` 1), 2 (`X_style` 2)	`int`	(5, 36)

input parameter	explanation	default value	data type	note
max_njq_change	maximum change of resonant mode numbers for X_grid_style 3	0.49	real	(36)
max_it_SLEPC	maximum number of iterations in SLEPC calculation to find eigenvalue	1000	int	
EV_BC	eigenvalue used in boundary condition of style BC_style = 1	1	real	(10)
EV_guess	target guess for eigenvalue	-0.3	real	
tol_SLEPC	tolerance used in SLEPC for different Richardson levels	$[1^{-5}, \ldots]$	real(max_it_rich)	
rho_style	style how the density ρ is calculated 1: constant rho_0	1	int	
U_style	style how to calculate geodesical perturbation $U = \vec{\xi} \cdot \frac{\nabla\psi \times \vec{B}}{\lvert\nabla\psi\rvert^2}$ is calculated 1: up to order $\sim \frac{1}{n}$ 2: up to order $\sim \left(\frac{1}{n}\right)^2$ 3: up to order $\sim \left(\frac{1}{n}\right)^3$	3	int	

input parameter	explanation	default value	data type	note
`K_style`	style how to calculate kinetic energy $K = \int \left\vec{\xi}\right\vert^2 d$V 1: normalization of full perpendicular component 2: normalization of only normal component	1	`int`	
`BC_style`	style how boundary conditions are applied for left and right boundary 1: set to zero 2: minimization of surface energy through asymmetric finite differences 3: minization through extension of grid 4: explicit introduction of the surface energy minimization	$[1, 2]$	`int(2)`	(10)

input parameter	explanation	default value	data type	note
`norm_style`	style how to calculate normalization 1: MIS↩ HKA normalization with magnetic field on axis 2: COB↩ RA normalization with pressure on axis	1	`int`	(11)
`solver_SLEPC_style`	style used in SL↩ EPC to solve the eigenvalue equation 1: Krylov-↩ Schur with shift-invert 2: generalized Davidson with preconditioner	1	`int`	
`matrix_SLEPC_style`	style used for S↩ LEPC matrices 1: sparse matrices 2: shell matrices	1	`int`	(12)

2.1.2 Command-line inputs

Table 2.2 Table 2. PB3D command-line options

input parameter	explanation	argument	note
`test`	test mode Note Debug version only		(28)
`no_plots`	do not produce any output plots		
`no_outputs`	do not produce text output		(29)
`do_execute_command_line`	run the external command line shell commands during execution		(30)
`mem_info`	produce memory profiles for each process Note Debug version only		(31)
`no_guess`	do not use a guess		
`jump_to_sol`	jump straight to the solution for the Richardson level at which the simulationes are (re)started, possibly for different solution grid parameters such as `n_r_sol`		(33)
`export_HEL`	create a VMEC input file from HELENA variables, possibly adding a toroidal ripple		(34)
`plot_VMEC_modes`	plot decay of VMEC modes		
`invert_top_bottom_H`	invert top and bottom of HELENA equilibrium Note Debug version only		

2.2 POST Inputs

2.2.1 Input file

Table 2.3 Table 3. POST input file

input parameter	explanation	default value	data type	note
concerning solution				
`min_r_plot`	minimum normalized flux of output domain $(0 \ldots 1)$		`real`	(24)
`max_r_plot`	maximum normalized flux of output domain $(0 \ldots 1)$		`real`	(24)
concerning input and output				
`n_sol_plotted`	number of solutions plotted	`n_sol_requested`	`int`	
`pert_mult_factor_POST`	factor by which the grid can be perturbed, with respect to the maximum perturbation of each solution independently	0	`real`	(25)
`plot_resonance`	plot the safety factor (poloidal flux) or rotational transform (toroidal flux) and the resonant values	`.false.`	`log`	(13)

input parameter	explanation	default value	data type	note
plot_magn_grid	plot magnetic field lines in the flux surfaces in 3-D geometry as an animation	.false.	log	(14)
plot_B	plot magnetic field $\vec{B} = \nabla\alpha \times \nabla\psi$ $[T]$	.false.	log	(15)
plot_J	plot current $\vec{J} = \mu_0^{-1}\nabla \times (\nabla\alpha \times \nabla\psi)$ $\left[\frac{A}{m^2}\right]$	.false.	log	(16)
plot_kappa	plot curvature $\vec{\kappa} = \frac{\vec{B}}{B} \cdot \nabla\frac{\vec{B}}{B}$ $\left[\frac{1}{m}\right]$ and its components $\kappa_n = \frac{\nabla\psi}{\lvert\nabla\psi\rvert^2} \cdot \vec{\kappa}$ $\left[\frac{1}{Tm^2}\right]$ and $\kappa_g = \frac{\nabla\psi\times\vec{B}}{B^2} \cdot \vec{\kappa}$ $[\,]$	.false.	log	(17)
plot_flux_q	plot flux quantities q, ι, p $[Pa]$, Ψ_{pol} $\left[Tm^2\right]$ and Ψ_{tor} $\left[Tm^2\right]$	.false.	log	(18)
plot_sol_xi	plot normal mode $\vec{\xi}$ $[m]$ and its components $X = \frac{\nabla\psi}{B^2} \cdot \vec{\xi}$ $\left[\frac{m^2}{T}\right]$ and $U = \frac{\nabla\psi\times\vec{B}}{\lvert\nabla\psi\rvert^2} \cdot \vec{\xi}$ $[\,]$	.false.	log	(26)
plot_sol_Q	plot magnetic field perturbation due to the normal mode $\vec{Q} = \nabla \times \left(\vec{\xi} \times \vec{B}\right)$ $[T]$ and its components $Q_n = \frac{\nabla\psi}{B^2} \cdot \vec{Q}$ $[m]$ and $Q_g = \frac{\nabla\psi\times\vec{B}}{\lvert\nabla\psi\rvert^2} \cdot \vec{Q}$ $\left[\frac{T}{m}\right]$	.false.	log	(26)
plot_E_rec	plot energy reconstruction	.false.	log	(27)
n_theta_plot	number of points in plots in θ direction		int	(19)
n_zeta_plot	number of points in plots in ζ direction		int	(19)
min_theta_plot	minimum of θ range for plots	1	int	
max_theta_plot	minimum of θ range for plots	3	int	
min_zeta_plot	minimum of ζ range for plots		int	(20)
max_zeta_plot	minimum of ζ range for plots		int	(20)
plot_size	size of 2-D plots in inch by inch	$[10, 5]$	int(2)	
ex_plot_style	style how external plots are made 1 – GnuPlot 2 – Bokeh for 2-D and Mayavi for 3-D	1	int	(21)
concerning Richardson extrapolation				
PB3D_rich_lvl	Richardson level at which to perform post-processing	int	(22)	
concerning finding zeros				
max_it_zero	maximum number of iterations in num_ops.calc_zero_hh() and calc_zero_zhang()	100	int	(9)
tol_zero	tolerance for zero finding	1^{-10}	real	(9)

input parameter	explanation	default value	data type	note
max_nr_backtracks_HH	maximum number of times the correction can be relaxed by half in a backtrack for the Householder methods	20	int	(9)
concerning runtime				
max_tot_mem	maximum total memory available to the simulation $[kB]$	6000	real	
POST_style	style how post-processing is done 1: extended rectangular grid in θ and ζ 2: field-aligned grid used internally by PB3D	1	int	(23)
plot_grid_style	style used for output plotting 0: 3-D plots 1: slab plots: θ and ζ are sides of a slab 2: slab plots with folding of fundamental interval $0 \ldots 2\pi$ 3: straight cylinder geometry: ζ is straightened	0	int	(23)

2.2.2 Command-line inputs

Table 2.4 Table 4. POST command-line options

input parameter	explanation	argument	note
test	test mode Note Debug version only		(28)
no_plots	do not produce any output plots		
no_outputs	do not produce text output		(29)
do_execute_command_line	run the external command line shell commands during execution		(30)
mem_info	produce memory profiles for each process Note Debug version only		(31)

input parameter	explanation	argument	note
swap_angles	swap θ and ζ of output grid in POST_style = 2		
compare_tor_pos	compare $\vec{B}$, $\vec{J}$ and $\vec{\kappa}$ from plot_B, plot_J and plot_kappa at different toroidal positions, as well as the difference in position	RZ_0(2)	(32)

Note

1. The normalized flux is either the poloidal (use_pol_flux_F = .true.) or toroidal (use_pol_flux_F = .false.), divided by its value at the boundary.
2. Is used in check_x_modes() whether there exists a range in which each of the modes resonates.
3. This is the number for the first Richardson level. It is doubled for every next level.
4. Should satisfy the high-n approximation, currently set to > 5 .
5. If min_sec_X and max_sec_X is set, the user prescribes the secondary mode number range (X_style = 1). If n_mod_X is set, on the other hand, the user prescribes the size of the range, which is automatically chosen (X_style = 2). They cannot be both chosen.
6. Normalization constants, with exception of rho_0 are set in the routine calc_normalization_const().
7. In store_results(), it is decided whether the eigenvalue is physical or not, by considering the ratio between real and imaginary part.
8. This needs to be between 1 and the maximum level that was obtained in a previous solution, plus 1. For technical HDF5 reasons, if Richardson restart is used, the same input fil parameters should be used for max_tot_mem and the same number of processes should be used. Furthermore, it should not exceed max_it_rich.
9. For the case of num_ops.calc_zero_hh(), at every iteration it is checked whether the correction approximation is better than the original one. If it is not, the correction is relaxed by a factor half. This is done max_nr_backtracks_HH times.
10. See set_bc().
11. See calc_normalization_const().
12. See solve_ev_system_slepc()
13. See resonance_plot().
14. See magn_grid_plot().
15. See b_plot(). The output is done in various coordinate system, as explained in calc_vec_comp().
16. See j_plot(). The output is done in various coordinate system, as explained in calc_vec_comp().
17. See kappa_plot(). The output is done in various coordinate system, as explained in calc_vec_comp().
18. See flux_q_plot().
19. For eq_style = 1, the equilibrium is axisymmetric, so by default n_theta_plot = 501 and n_zeta_plot = 1; for eq_style = 2, the equilibrium can be 3-D, so n_theta_plot = 201 and n_zeta_plot = 51
20. For eq_style = 1, the equilibrium is axisymmetric, so by default min_zeta_plot = max_zeta_plot 0; for eq_style = 2, the equilibrium can be 3-D, so min_zeta_plot = 0 and max_zeta_plot = 2.
21. This refers to the external plots, which are complimentary to the HDF5 plots. See draw_ex().
22. If output is done that requires a solution, this needs to be between 1 and the maximum level that was obtained in PB3D. See find_max_rich_lvl(). If no solution is found, some of the outputs that do not depend on this still work.Optionally, this can be forced by choosing PB3D_rich_lvl not positive.
23. The POST_style determines whether the post-processing is done on an extended, new grid, or on the PB3D internal grid. On top of this, through plot_grid_style the user can set the output grid for the post-processing. Note that some combinations of styles do not make sense.

24. By default, the plot normal range is chosen equal to the solution normal range from `min_r_sol` and `max_r_sol`, but it can be restricted. This is useful, for example, to reduce computation time.
25. The Cartesian components of the result of the division of `pert_mult_factor_POST` by the maximum value of the solution normal mode amplitude is added to the equilibrium Cartesian grid components to illustrate the perturbation in 3-D. This is done for each solution independently and sequentially. If it is not positive, it is ignored. Note that `plot_sol_xi = .true.` is necessary for this to be used.
26. These quantities are calculated in sol_utilities.calc_xuq(). This is done for a time sequene, currently hard-coded in plot_sol_vec(), depending on whether the solution is stable or unstable.
27. The energy reconstruction is performed in calc_e() if either one of `plot_sol_xi`, `plot_sol_Q` or `plot_E_rec` is true.
28. Equivalent use is through `-t`. If this is used, `do_execute_command_line` should be set to true as well. See generic_tests().
29. This is used internally as well, to suppress certain output. See run_driver_x() and run_driver_post().
30. This can fail on local systems, and generally works badly on computational clusters, where the computational nodes often don't have the external tools (see `ex_plot_style`) installed. By default, an output file is created for the shell commands, which can be executed afterwards. See Shell commands file.
31. This is done by calling get_mem_usage() and storing the result at every allocation and deallocation of an `eq_1_type`, `eq_2_type`, `X_1_type`, `X_2_type`, `vac_type` or `sol_type`, as well as for input quantities. Furthermore at every user output with writo(), this is also done, and the output is prepended to the output string. Also, the output is written to a file. See Memory usage file.
32. This is useful to calculate the ripple in these quantities. To use this, it is necessary to have an output grid of only 3 points, where the poloidal projection of the middle point should be the in the middle between the poloidal projections of the two others. Also, you have to provide through `RZ_0` the R and Z value of the origin of the geometrical poloidal angle that is used to calculate distances. See grid_utilities.calc_tor_diff().
33. Jumping straight to the solution is useful if a simulation already calculated all the necessary quantities to set up the solution matrices $\overline{\mathrm{A}}$ and $\overline{\mathrm{B}}$ in $\overline{\mathrm{A}}\vec{X} = \lambda\overline{\mathrm{B}}\vec{X}$ and the solution needs to be redone, for example because a calculation was attempted with a method that did not converge.
34. This is a rather complicated procedure with many possibilities that is intended for expert use. See create_vmec_input().
35. Using the toroidal flux is **experimental** and **untested**.
36. Have a look in calc_norm_range(). `X_grid_style` 1 is generally too coarse while `X_grid_style` 2 is too fine. `X_grid_style` 3 was designed to have the both of best world. It only makes sense for `X_style` 2 (fast), though, as in the enrichment process the fast secondary mode range is used to limit the maximum amount the quantity nq (when using poloidal flux) or $m\iota$ (when using toroidal flux) to `max_njq_change` between two normal grid points.

Chapter 3

Frequently Asked Questions

3.1 runtime errors

Table 3.1 Table 1. frequently encountered errors

error	solution
”need at least N points in grid”	grid_utilities.setup_deriv_data() needs a minimum number of points in a grid to setup data with which the derivatives can be calculated later in grid_utilities.apply_disc(). If the grid does not have enough points, and the order of the derivative is too high, the procedure gives this error message. The solution is to either increase the number of points, or to lower the order. It might be that the number of points in the grid is lower than you expect because the grid is divided. In this case, you need more memory (see `max_tot_mem` in Input variables).
”Aument number of modes or choose finer grid”	The range of secondary mode numbers at every flux surface is either user-specified, for `X_style` 1, or chosen automatically around the most resonant values for `X_style` 2 (see Input variables). This implies that, on the other hand, every secondary mode should have a normal range in which it is present, possibly multiple ranges for reversed-shear configurations. The code automatically checks whether this is indeed the case. It could, for example, be that a certain mode number is not present anywhere, when the normal grid in which it is tabulated is too coarse. A solution is therefore to choose a finer grid. Alternatively, for `X_style` 1, the range can be adapted, and for `X_style` 2 the number of modes `n_mod_X`. Note that the particular grid that this discussion is about, can be found out by the trace of the error message. If it results from the solution driver, it concerns the solution grid, and if it concerns the perturbation driver, it concerns the perturbation grid. The solution grid is specified by its range through `min_r_sol` and `max_r_sol` and the number of points `n_r_sol`, and the perturbation grid by the `X_grid_style`. See Input variables.

error	solution
”Zero pivot in LU factorization”	There exist cases in which the eigenvalue solver fails when `n_mod_X = 1`, for unkown reasons. Increase `n_mod_X` or play with other parameters, such as the `norm_disc_prec_sol`.
`jump_to_sol` does not work	Jumping over the equilibrium and perturbation drivers straight to the solution only works if solution parameters such as the boundary conditions are changed, *not* if the number of points in the solution, `n_r_sol`, is changed, as in this case the equilibrium and solution drivers need to be recalculated properly.
The solution suffers from numerical oscillations	This is a work in progress. Change the limits of the solution grid `min_r_sol` (and possibly `max_r_sol`). Better even is to use asymmetric finite differences `norm_disc_style_sol = 2`

3.2 common problems

Table 3.2 Table 2. common problems

problem	solution
code compilation stuck	Especially on intel systems, it has been found that sometimes the compiler seems to be stuck, especially for `eq_ops`. A solution is to use less optimization for the problematic files through `-O2`.
unphysical eigenvector	For coarse equilbrium grids and/or strongly varying safety factors, and when using the fast version (i.e. with `n_mod_X`), it is possible that the resonating mode number range varies too quickly, leading to inaccurate interpolation of tensorial perturbation quantities in the solution grid. Try using `X_grid_style` 3, where enrichment of the equilibrium grid is used to set up the perturbation grid. If this does not help, additionally lower `max_njq_change` from its default value.

problem	solution
low accuracy for energy reconstruction	Energy reconstruction tries to see whether by calculating the total volume integral of the perturbed potential energy and dividing it by the total volume integral of the kinetic energy of the perturbation, the eigenvalue can be recovered. This gives a lot of insight in the processes that drive the instabilities, by looking at the individual terms of these integrals, and is a good check on the internal consistency of the PB3D code. Note that for axisymmetric equilibria, the volume integral reduces to a surface integral in a poloidal cross-section. It is actually very hard to perform a good energy reconstruction because the techniques used differ from the ones in the PB3D code, where the normal derivatives of the eigenvector are performed using finite differences, while in the energy reconstruction direct volume integrals are used. To get the best possible results, therefore, it is important to achieve full Richardson convergence in the number of parallel points (by increasing `max_it_rich` and/or `min_n_par_X`), as well as convergence in the number of secondary mode harmonics (by increasing `n_mod_X`). Doing this, for axisymmetric cases, the code should retun quite accurate results, from version 2.26 upwards. These can furthermore be improved by choosing a finer post-processing grid (see `n_theta_plot` and `n_zeta_plot`). It is, however, possible that a small fraction, possibly of the order of a percent remains in the relative difference. This is probably due to the fact that in PB3D for `X_style` 2 (fast), the stencil is cut when the normal index falls out of its mode range (see insert_block_mat()).
Changing input does not work for Richardson extrapolation	You cannot change the inputs as Richardson extrapolation depends on having these constant (with the exception of the number of poins in the parallel integrals, but this is done internally in PB3D). The important exception to this fact, however, is that you can change some of the solution grid parameters, such as `n_r_sol` (see Input variables).
The code has not been tested satisfactorily for decreasing normal coordinate.	When the normal coordinate is decreasing (but still monotonous), the code should be able to handle this. However, a quick test gave some problems, which have to be debuged. An easy workaround is to just run the equilibrium solver again with inverted fields and currents.
The code has to be adapted for reversed-shear regions still.	When there are reversed-shear regions, the routines that calculate the resonant flux surfaces, such as `calc_res_surf()` need to be extended. If this is not done, there will be problems, mostly for `X_style` 2 (fast), and especially for `X_grid_style` 3 (enriched).

Chapter 4

Code outputs

Footnotes are situated at the end of the chapter

This page describes the various outputs that PB3D and POST can produce.

On the one hand, there are various text output files, which are discussed in section Output Files.

Furthermore, a multitude of plots can be produced. This is the topic of section Output Plots.

4.1 Output Files

Some of these files are opened in open_output(), but others are opened afterwards.

4.1.1 Log file

Both PB3D and POST produce a log file, in addition to what is produced on the screen.

These log files are called `PB3D_out.txt` and `POST_out.txt` and they contain the same information as the output on the screen, but without the special formatting provided by the foul library (see Introduction).

4.1.2 Shell commands file

Some output routines make use of use_execute_command_line() in order to execute shell commands. This can be, for example, the creation of a plot by calling GnuPlot, among other things.

However, this often does not work well by default (see 30 in Input variables). In PB3D and POST, therefore, by default, these commands are not executed, but written to an output file. After execution of PB3D and POST has finished, this file can be made executed.

This behavior can be changed using the `--do_execute_command_line` flag (see Input variables).

4.1.3 HDF5 output file

PB3D writes variables to an HDF5 output file during execution, as shown in table 4.1. Note that there is a difference between the equilibrium styles because of the way how the equilibrium codes provide output:

- VMEC (`eq_style = 1`) produces double Fourier series:
  - This can be evaluated at arbitrary angular points.
  - In PB3D, the equilibrium and perturbation variables are therefore already calculated on the field-aligned grid.
  - Parallel integration can then be done directly.
- HELENA (`eq_style = 2`) produces 2-D tables for the poloidal cross-section.
  - This needs to be interpolated along the magnetic field lines before parallel integration.
  - The equilibrium and perturbation variables are calculated first in the HELENA grid.

Table 4.1 Table 1. HDF5 output file

variable type	VMEC (`eq_style = 1`)	HELENA (`eq_style = 2`)
inputs	in preliminary phase	
grids of grid_vars.grid_type	**equilibrium**	
	field-aligned-grid at every Richardson level	
		also output HELENA grid at first Richardson level
		used later for interpolation to field-aligned grid
	perturbation	
	field-aligned-grid at every Richardson level	
		also output HELENA grid at first Richardson level
		used later for interpolation to field-aligned grid
	solution	
	only at first Richardson level	
flux equilibria of eq_vars.eq_1_type	only at first Richardson level	
metric equilibria of eq_vars.eq_2_type	only at first Richardson level	
		also output HELENA grid at first Richardson level
		used later for interpolation to field-aligned grid
vacuums of vac_vars.vac_type	at every Richardson level	only at first Richardson level
solutions of sol_vars.sol_type	at every Richardson level	

This data is used for Richardson restart (see variable `rich_restart_lvl` in Input variables).

In the post-processing phase, POST reads this data as well.

4.1.4 Eigenvalue summary file

In each Richardson extrapolation level, after the solution of the eigenvalue system of equations, eigenvalues are stored in PB3D.

- For SLEPC, the results are saved in the procedure store_results():
  - If normalization is used, the normalization time scale is also stored.
  - If an unphysical complex eigenvalue is detected, this is indicated. See `option` retain_all_sol in Input file.

4.1.5 Energy reconstruction file

If energy reconstruction is performed by POST (see `plot_E_rec` in Input variables), the individual terms that make up the perturbed potential energy as well as the kinetic energy of the perturbation are calculated and plot. (3) Furthermore, the integration of these quantities over the whole volume, is returned in an output file.

In table 4.2, an overview is given of the different terms. Here, ρ is the density, $\sigma = \frac{\vec{B}\cdot\vec{J}}{B^2}$ is the parallel current and $\kappa = \frac{\vec{B}}{B}\cdot\nabla\frac{\vec{B}}{B}$ is the magnetic curvature. See calc_e().

Table 4.2 Table 2. energy reconstruction

potential energy E_{pot}	kinetic energy E_{kin}
• normal line bending term $\frac{1}{\lvert\nabla\psi\rvert^2}\left(\frac{\nabla\psi}{\lvert\nabla\psi\rvert^2}\cdot\vec{Q}\right)^2$ • gedesic line bending term $\mathcal{J}^2\frac{\lvert\nabla\psi\rvert^2}{B^2}\left(\frac{\nabla\psi\times\vec{B}}{B^2}\cdot\vec{Q}\right)^2$ • normal ballooning term $-2p'\left(\nabla\psi\cdot\vec{\xi}\right)^2(\nabla\psi\cdot\vec{\kappa})$ • geodesic ballooning term $-2p'\left(\nabla\psi\cdot\vec{\xi}\right)\left(\frac{\nabla\psi\times\vec{B}}{\lvert\nabla\psi\rvert^2}\cdot\vec{\xi}\right)^*\left(\frac{\nabla\psi\times\vec{B}}{\lvert\nabla\psi\rvert^2}\cdot\vec{\kappa}\right)$ • normal kink term $-2\sigma\left(\nabla\psi\cdot\vec{\xi}\right)^*\left(\frac{\nabla\psi\times\vec{B}}{B^2}\cdot\vec{Q}\right)$ • geodesic kink term $2\sigma\left(\frac{\nabla\psi\times\vec{B}}{B^2}\cdot\vec{\xi}\right)^*\left(\nabla\psi\cdot\vec{Q}\right)$	• normal kinetic term $\frac{1}{\lvert\nabla\psi\rvert^2}\left(\frac{\nabla\psi}{\lvert\nabla\psi\rvert^2}\cdot\vec{\xi}\right)^2$ • gedesic kinetic term $\mathcal{J}^2\frac{\lvert\nabla\psi\rvert^2}{B^2}\left(\frac{\nabla\psi\times\vec{B}}{B^2}\cdot\vec{\xi}\right)^2$

4.1.6 Memory usage file

The memory usage of PB3D and POST can be monitored using the optional input argument `--mem_info` (see Command-line inputs).

This is a rather crude implementation and more advance profiling methods should be used, such as `Scalasca` on top of `Score-P`.

Note

Debug version only

4.2 Output Plots

A multitude of plots can be produced by PB3D and POST.

POST can produce all the plots that PB3D can, and some more.

The plots themselves fall in two categories:

- Plots using an external program.
  - Produced by output_ops.print_ex_2d() or output_ops.print_ex_3d() combined with draw_ex().
  - A list of external programs is given in draw_ex().
  - These plots require a shell command to be run, which is disabled by default and the commands are stored in a Shell commands file.
  - The produced plots can be 2-D or 3-D, and they can be interactive or not, depending on the ex↩ `_plot_style` (see Input variables).
- Data files that can be used to produce plots with visualization software:
  - The software used here was `VisIt`, based on personal preference, but `ParaView` or other V↩ TK-based solutions should work equally well.
  - Every plot creates two output files: one heavy-data HDF5 file and one `XDMF` file that contains specifics about the HDF5 file.
  - This is necessary, as HDF5 itself is too general a format for general visualization tools to interpret easily.
  - The XDMF file that accompanies each HDF5 file basically contains information about the saved variables, such as whether they are vectors or scalars, and on which grid type they are defined.
  - This is the file that should be opened by the visualzation software.
  - The dimensions of these plots can be 2-D or 3-D, depending on `eq_style`. See 19 and 20 in Input variables.

An overview of possible plots is now given in table 4.3. The name of the respective input parameter is provided as well. See Input variables.

Table 4.3 Table 3. possible plots

plot type	external plot	HDF5 plot
flux quantities (`plot_flux_q`)	• pressure p • poloidal flux Ψ_{pol} • toroidal flux Ψ_{tor}	• safety factor q • rotational transform ι • pressure p • poloidal flux Ψ_{pol} • toroidal flux Ψ_{tor}
resonance (`plot_resonance`)	safety factor q (poloidal flux) or rotational transform ι (toroidal flux) with the indication of the resonant flux surfaces (1)	surfaces of resonant values for safety factor q (poloidal flux) or rotational transform ι (toroidal flux) (1)
magnetic grid (`plot_magn_grid`)		magnetic grid lines in the magnetic flux surfaces, as a time collection

plot type	external plot	HDF5 plot
magnetic field $\vec{B}$ (`plot_B`)		co- and contravariant components of magnetic field, as well as magnitude and vector plot
current density $\vec{J}$ (`plot_J`)		co- and contravariant components of current density, as well as magnitude and vector plot
magnetic curvature $\vec{\kappa}$ (`plot_kappa`)		co- and contravariant components of magnetic curvature, as well as magnitude and vector plot
solution normal mode $\vec{\xi}$ (`plot_sol_xi`)		real and imaginary part and phase of perturbation of position, as well as vector plot
perturbated magnetic field due to solution normal mode $\vec{Q}$ (`plot_sol_Q`)		real and imaginary part and phase of perturbation of magnetic field, as well as vector plot
energy reconstruction (`plot_E_rec`)		different components of energy reconstruction, integrated as well as profiles (2)

Note

1. See `use_pol_flux_F` in Input variables.
2. The integrated energy reconstruction terms are also output to a file. See Energy reconstruction file.
3. Energy reconstruction only works well when `POST_style = 2` is used, i.e. when the same grid is used as for PB3D.

Chapter 5

General code structure

This page describes the general code structure of PB3D, as well as its accompanying post-process program POST. It is more complete than the documentation that can be found in [17].

5.1 PB3D flowchart

In figure 1, the top-level flowchart of PB3D is shown (adapted from [17]):

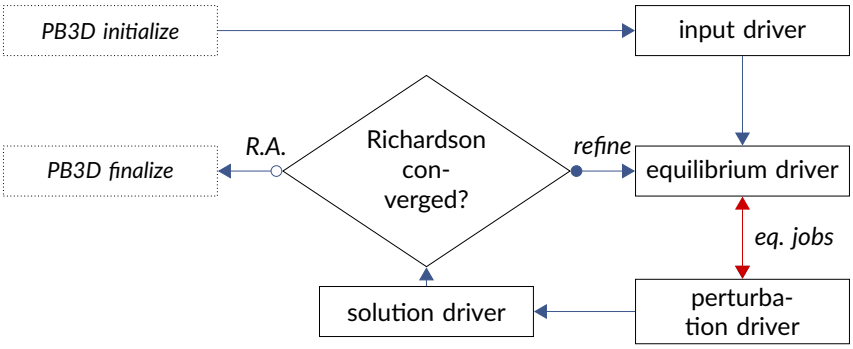


Figure 5.1 Figure 1 : basic PB3D flowchart

Roughly speaking, the main body of the code is taken up by four driver routines:

- input driver
- equilibrium driver
- perturbation driver
- solution driver

The **input driver** sets the other drivers up by reading input options as well as quantities from equilibrium codes, and converting them to the internal PB3D format.

The **equilibrium driver** uses these input variables to set up equilibrium quantities.

- On the one hand these consist of flux equilibrium quantities (stored in a eq_vars.eq_1_type) that only depend on the flux coordinates, such as the pressure p, the safety factor q, etc.
- On the other hand, they consist of metric equilibrium quantities (stored in a eq_vars.eq_2_type) that depend on the angular position as well, such as the metric quantities of the grid $g_{ij} = \vec{e}_i \cdot \vec{e}_j$ and $h_{ij} = \nabla u^i \cdot \nabla u^j$, as well as derived quantities such as the parallel current $\sigma = \frac{\vec{B} \cdot \vec{J}}{B^2}$, etc. [16]

The **perturbation driver** uses both types of equation variables to calculate perturbation quantites. Again, there are two types.

- The first type of variables are the vectorial perturbation quantities (stored in a x_vars.x_1_type), which contains the geodesic component of the perturbation $U = \frac{\nabla \times \vec{B}}{|\nabla \psi|^2} \cdot \vec{\xi}$ where $\vec{\xi}$ is the perturbation. Also the parallel derivatives of these quantities are calculated.
- The other type re the tensorial perturbation quantities (stored in a x_vars.x_2_type), and it combines the vectorial perturbation quantities with the equilibrium quantities to form the tensorial quantities $\widetilde{PV}^i_{k,m}$ and $\widetilde{KV}^i_{k,m}$ that represent the perturbed potential energy as well as the kinetic energy of the perturbation [16].

These are also complemented by $\delta^{\text{vac}}_{k,m}$, the vacuum contribution to the perturbed potential energy (1).

After integrating them along the magnetic field lines, these magnetic integrals form the building blocks of the discretized matrices $\overline{\text{A}}$ and $\overline{\text{B}}$ in the generalized system of eigenvalue equations $\overline{\text{A}}\vec{X} = \lambda \overline{\text{B}} \vec{X}$.

These matrices are set up in the **solution driver**, which currently works with the SLEPc library that is build on the PETSc library for sparse linear algebra. After solution, the eigenvalues and -vectors are stored appropriately.

Finally, it is checked whether another Richardson level should be attempted or not in **Richardson converged?**, and if so, the number of parallel grid points is doubled.

5.2 Equilibrium & Perturbation Jobs

The general code structure explained in PB3D flowchart basically comes down to setting up the matrices to be solved for the eigenvalues and eigenvectors. In general, the dimensions of these variables are too large (see the grids in Detailed PB3D Flowchart) for them to fit in memory. A structure of two nested loops is therefore employed in PB3D, called **jobs**, to keep the memory requirements under a user-defined maximum (see `max_tot_mem` in Input variables).

5.2.1 Perturbation jobs

perturbation jobs are used to divided the calculation of the tensorial perturbation quantities, which come in pairs of mode numbers. These combinations are blocked, where each block fits in memory. These blocks are the perturbation jobs.

They form the inner loop.

See also

See divide_x_jobs().

5.2.2 Equilibrium jobs

As a result the memory still needs to hold all the relevant equilibrium and vectorial perturbation quantities necessary for the individual perturbation jobs during the whole time these jobs are being calculated. **equilibrium jobs** are therefore employed in an outer loop to divide these equilibrium and vectorial perturbation quantities in manageable chunks.

To be more precise, the parallel range over which variables are to be integrated in the magnetic integrals (see Detailed PB3D Flowchart), is cut in pieces. The advantage here is that only the results from the sub-integrals have to be saved and updated, not the function values themselves.

See also

See divide_eq_jobs().

5.3 Detailed PB3D Flowchart

The different blocks if figure 1 are expanded in table 5.1. The cells that are typeset in light blue are only performed for the first equilibrium job of either the restart Richardson level, or the first Richardson level if this is not provided.

Table 5.1 Table 1. detailed PB3D flowchart

variable type	VMEC (`eq_style = 1`)	HELENA (`eq_style = 2`)
PB3D initialize	the PB3D structure is set up, such as the MPI communicator, etc.	
input driver	The arguments are parsed (see parse_args())	
	The input files are opened (see open_input() and Input variables)	
	The user inputs are processed (see read_input_opts())	
	The equilibrium code output is read and converted to the internal PB3D format (see read_inp	
	Normalization constants are calculated (see calc_normalization_const()) and the input is norm	
	Output files are opened (see open_output() and Code outputs)	
	Input variables are written to HDF5 output (see print_output_in())	
	Some other variables are broadcasted directly to all the processes using MPI (see broadcast_	
	normal part of equilibrium grid is set up, as for the angular part, the field-lines are necessary which are not yet calculated	the equilibrium grid is set up equal to the HELENA output grid
	flux equilibrium quantities (see eq_vars.eq_1_type) are calculated (see eq_ops.calc_eq())	
	flux equilibrium quantities are written to HDF5 output (see eq_ops.print_output_eq())	
	equilibrium grid is completed by tracing the field lines (see calc_ang_grid_eq_b())	an additional, field-aligned equilibrium grid is set up by tracing the field lines (see setup_grid_eq_b()), and it is written to HDF5 output (see print_output_grid())
	the equilibrium grid is written to output	the equilibrium grid is written to the output
	the metric equilibrium variables (see eq_vars.eq_2_type) are calculated (see eq_ops.calc_eq())	the metric equilibrium variables (see eq_vars.eq_2_type) are calculated (see eq_ops.calc_eq())

variable type	VMEC (`eq_style = 1`)	HELENA (`eq_style = 2`)
		the metric equilibrium variables are written to the output (see eq_ops.print_output_eq())
	store equilibrium quantities necessary for vacuum calculation (see store_vac())	store equilibrium quantities necessary for vacuum calculation (see store_vac())
	possible plot of magnetic field, current density and/or curvature in (field-aligned) equilibrium grid (see b_plot(), j_plot(), kappa_plot())	possible plot of magnetic field, current density and/or curvature in (HELENA) equilibrium grid (see b_plot(), j_plot(), kappa_plot())
	redistribute the equilibrium grid (see redistribute_output_grid())	redistribute the equilibrium grid (see redistribute_output_grid())
	redistribute the flux equilibrium quantities (see eq_ops.redistribute_output_eq())	
	redistribute the metric equilibrium quantities (see eq_ops.redistribute_output_eq())	redistribute the metric equilibrium quantities (see eq_ops.redistribute_output_eq())
		The field-aligned equilibrium grid is redistributed (see redistribute_output_grid())
	possible plot of magnetic field lines in flux surfaces (see magn_grid_plot()) if last equilibrium jo	
perturbation driver		the metric equilibrium quantities are inerpolated in the field-aligned equilibrium grid (see interp_hel_on_grid())
	determine perturbation grid (see setup_grid_x()) and write it to HD↩ F5 ouput (see print_output_grid())	determine perturbation grid (see setup_grid_x()) and write it to HD↩ F5 ouput (see print_output_grid())
		set up field-aligned perturbation grid and write to output
	set up mode number information (see setup_nm_x()) and check them (see check_x_modes()) if	
	possibly plot resonance (see resonance_plot()) if first perturbation job	
	the vectorial perturbation variables (see x_vars.x_1_type) are calculated (see x_ops.calc_x())	the vectorial perturbation variables (see x_vars.x_1_type) are calculated (see x_ops.calc_x())
		write vectorial perturbation variables to output (see x_ops.print_output_x())
		interpolate vectorial perturbation variables on field-aligned grid (see interp_hel_on_grid())
	calculate tensorial perturbation variables (see x_ops.calc_x())	
	calculate magnetic integrals (see calc_magn_ints())	
	write magnetic integrals to output (see x_ops.print_output_x())	
solution driver	calculate vacuum (see calc_vac())	calculate vacuum (see calc_vac())
	set up the solution grid (see setup_grid_sol()) and write to HDF5 output (see print_output_gri	
	solve system of equations (see solve_ev_system_slepc())	
Richardson converged?	check for convergence of Richardson levels by comparing solution eigenvalues (see check_co	
PB3D finalize	clean up	

5.4 POST Overview

The POST program for post-processing of PB3D results consists of just one driver.

- In the first part, everything is set up (see init_post()):
  - This includes initializing the variables written to HDF5 in PB3D.
  - Furthermore, the output grids are created
  - Also, 1-D flux plots are produced, which are part of the plots that PB3D is able to produce (see `plot_flux_q`, `plot_resonance` and `plot_magn_grid` in table 4.3 in Code outputs) can be created here.
- After that, new outputs are calculated and plot (see run_driver_post()):
  - Quantities are calculated on the output grids.
  - Using these, plots are first produced that do not depend on the solution vector from PB3D, and are also part of the plots that PB3D is able to produce (see `plot_B`, `plot_J`, `plot_kappa` in table 4.3 in Code outputs).
  - Finally, other plots are produced that do depend on these (see `plot_sol_xi`, `plot_sol_Q`, `plot_E_rec` in table 4.3 in Code outputs).

POST also knows the concept of equilibrium jobs, but not that of perturbation jobs (see Equilibrium & Perturbation Jobs). This is necessary for larger outputs that quickly overflow memory. The HDF5 output files are updated at every equilibrium job, but, with reasonable safety, they can already be looked at in a visualizer.

Note

1. This is not yet implemented.

Chapter 6

Tutorial

Footnotes are situated at the end of the chapter

This page contains a hands-on tutorial to get you used to running PB3D to calculate 3-D ideal linear high-n MHD stability, and POST to post-process the results.

Be sure to check out the Installation instruction first.

If at any point a routine is not clear, don't hesitate to consult the appendices.

6.1 Getting the Equilibrium

PB3D calculates stability of an equilibrium magnetic toroidal configuration. It is written in a modular way to isolate this main task from the actual code that was used to obtain the equilibrium configuration, and the format that this code may use. Currently, two formats are possible:

- 3-D VMEC [8].
- Axisymmetric HELENA [11].

VMEC is a general 3-D code built on an interesting idea: The poloidal coordinate θ is deformed in such as a way to make the Fourier basis that is used to describe the cylindrical coordinates $R(s,\theta,\varphi)$ and $Z(s,\theta,\varphi)$ as narrow as possible, where s is the flux label ($s = \sqrt{\Psi_{\text{tor}}/\Psi_{\text{tor,edge}}}$) and φ is the regular cylindrical angle (1). This is done by introducing a deformation factor $\lambda(s,\theta,\varphi)$ defined so that the straight-field line coordinate θ_{F} is given by $\theta_{\text{F}} = \theta + \lambda$.

The fact that R, Z and λ are given as a double Fourier series in θ and ζ ensures that the angular derivatives can be done analytically. This property is used in PB3D. Apart from that, the flux functions that describe the profiles on the flux surfaces, the rotational transform $\iota(s)$ and pressure $p(s)$, are also returned.

For HELENA, the situation is different, as this code considers axisymmetric equilibria only. Its output consists, aside from flux functions safety factor $q(\psi)$, presure $p(\psi)$ and toroidal covariant magnetic field $F(\psi)$, where $\psi = \Psi_{\text{pol}}/\Psi_{\text{pol,edge}}$ is the normalized poloidal angle, of a 2-D table for the upper metric factors $h^{\psi\psi}$ and $h^{\psi\theta}$ and the lower metric factor $g_{\zeta\zeta}$. HELENA uses the same angular coordinates θ and $\zeta = -\varphi$ as PB3D (see 1).

Note

The modular nature of PB3D enables easy integration of other equilibrium codes, if necessary.

To get the appropriate equilibrium code results into PB3D, it suffices to place the output files in the same directory in which PB3D is run:

- for HELENA, it is a text file, corresponding to the mapping file (`fort.12`),
- for VMEC, it is the so-called `wout` file,

and to specify their names as the second input argument, the first one being the file with user inputs (see Setting Up the Input File).

Note

For VMEC, the `wout` file has to be in Netcdf's `.nc` format. The old `.txt` format is no longer supported.

In any case, if you forget how to run the PB3D (or POST) programs, you can always just type
```
./PB3D --help
```

or even just
```
./PB3D
```

Note

The HELENA version used here outputs the variables `IAS` and `B0` to the mapping file, after the variable `JS0` and after the variable `RAXIS`. Some versions do not write these by default. If yours is one of the above, the solution is to change
```
write(nmap,8) JS0
```
to
```
write(nmap,8) JS0, IAS
```
and
```
write(nmap,8) RAXIS
```
to
```
write(nmap,8) RAXIS, B0
```
in the subroutine `mapping` and recompile and re-run HELENA.

6.2 Checking the equilibrium (optional)

It is always important that the equilibrium has been properly calculated, well-converged, etc.

It is possible to use PB3D to check this. This will be quickly described in this optional section.

The checks described below are only possible with the code compiled with the debug flag on (see Installation). Using the debug version, the tests can be triggered using the `--test` option when running PB3D (or POST), and using the interactive user interface.

They generally output HDF5 files that can be read with ParaView or VisIt to check the difference between two plots. After having a look in the code (search for `ltest`), this should be self-explanatory.

Note

Most of the checks are only useful for developers. However, the tests on `B_F` and on the pressure balance consistency are vital.

Table 6.1 Table 1. the most important tests

tests	purpose
calculation of `g_V`	Check whether equilibrium metric coefficients calculated for VMEC equilibria through application of transformation matrices match with a direct implementation of the VMEC metric coefficients. Note that the reason why the transformation matrices are used, is that derivatives can be easily calculated as well, with high precision.
calculation of `jac_V`	Similar to the test on the calculation of `g_V`, but this time for the Jacobian of VMEC coordinates.
consistency of HELENA metric factors	Check whether the metric factors returned by HELENA are consistent with the ones calculated from the flux quantities, R and Z. Basically, this is a measure for the accuracy with which the pressure balance is satisfied.
harmonic content of HELENA output	Analyze harmonic content of HELENA variables.
calculation of `D1 D2 h_H`	Check whether the cross-derivatives are correctly and accurately calculated.
calculation of `T_EF`	Check is $\mathcal{T}_E^F$ complies with the (unpublished) theory.
calculation of `jac_F`	Similar to the test on the calculation of `jac_V`, but this time for the Jacobian of Flux coordinates.
calculation of `B_F`	Similar to the test on the calculation of `jac_F`, but this time for the covariant components and magnitude of the magnetic field in Flux coordinates.
consistency of pressure balance	Check how well the pressure balance is satisfied. This is a very crucial requirement for the correct calculation of stability: If the pressure balance is not satisfied up to high precision, there is not much for PB3D to do.
calculation of `DU`	Check how well the poloidal derivative of the geodesical component of the minimizing perturbation, U, is calculated by comparing it to the numerical derivative.

For the specific case of VMEC equilibria, finally, there is the additional `--plot_VMEC_modes` option that produces outputs of the harmonics of the different quantities: R, Z, λ, $\mathcal{J}$ and $\vec{B}$.

It is very important that these are well-converged, as the internal consistency depends strongly on this. Therefore, if this is not the case, be prepared to receive unaccurate stability results.

6.3 Setting Up the Input File

Now that we have the equilibrium files, it is time to have a look at the input file.

```
1 &inputdata_PB3D
2     min_n_par_X             = 500
3     min_par_X               = 0.0
4     max_par_X               = 2.0
5     alpha                   = 0.0
6     min_r_sol               = 0.1
7     max_r_sol               = 1.0
8     n_r_sol                 = 500
9     prim_X                  = 10
10     n_mod_X                 = 20
11     BC_style                = 1 1
12     U_style                 = 3
13     max_tot_mem             = 8000
14     max_it_rich             = 2
15     use_normalization       = .true.
16 /
```

```
17 //! [some extra options for PB3D]
18     rich_restart_lvl        = 1
19     tol_rich                = 1.0E-4
20     tol_SLEPC               = 1.0E-8 1.0E-8
21 //! [some extra options for PB3D]
```

Have a look:

- `min_n_par_X` is the number of points in the parallel grid.
- They range from `min_par_X` to `max_par_X`. As this is an axisymmetric example, they can simply be chosen to be a fundamental period of 2π. For 3-D equilibria, this number has to be increased until convergence is reached [17].
- `alpha` indicates the field line at which the perturbations are situated.
- `n_r_sol` is the number of points in the solution grid, that ranges from `min_r_sol` to `max_r_sol` in the PB3D normal coordinate (see num_vars.use_pol_flux_f).
- `n_mod_X` is the number of resonating Harmonics with `prim_X` the primary mode number.
- `BC_style(1:2)` indicates the style with which the boundary conditions are implemented deep in the plasma and on the plasma edge.
- `max_tot_mem` is the total virtual memory that PB3D may consume.
- `use_normalization` indicates whether normalization is used (recommended), but it is useful to turn this off for debugging.

Note

`BC_style(2)` is currently set to 1, indicating that the mode amplitude is zero at the edge. In the upcoming version of PB3D, the vacuum perturbation on the potential energy will be taken into account, which enables other boundary condition styles at the plasma edge.

There are also some extra options listed (but not loaded in PB3D as they com after the end of the `inputdata_PB3D` block of the Fortran namelist) below:

```
18     rich_restart_lvl        = 1
19     tol_rich                = 1.0E-4
20     tol_SLEPC               = 1.0E-8 1.0E-8
```

They include:

- `rich_restart_lvl` is used to restart a simulation at a certain Richardson level. The Richardson level 1 refers to the first simulation, with the number of parallel points used in the calculation of line integrals along the magnetic field set by `min_n_par_X` (see 2.1). For every subsequent Richardson level, this number is approximately doubled in order to gain a grid size that is twice as fine. The results from these different levels are extrapolated to a theoretically infinitely fine grid. These operations are done in module rich_ops.
- `tol_rich` is the tolerance at which Richardson extrapolation is considered to have converged.
- `tol_SLEPC` is an array of tolerances used for the different SLEPc eigenvalue solutions.

There are many, many more input parameters. A short description of these can be found in Input variables.

6.4 Running PB3D

This tutorial provides a basic VMEC equilibrium file to experiment with, called `cbm18`, which represents a simple circular test case designed to be ballooning unstable. [5] Some properties are given in table 6.2.

Table 6.2 Table 1. equilibrium parameters

toroidal current I_{tot}	$1.614MA$
β_{pol}	0.443
aspect ratio ϵ	1.5

Put this equilibrium file, as well as the input file in a separate run directory; let's call it `cbm18`.

In order to produce outputs, PB3D needs three sub-directories in the run directory, called `Data/`, `Scripts/` and `Plots/`.

- In `Data/`, datafiles are stored that are used for plots.
- In `Scripts/`, scripts are generated that use the datafiles from `Data`.
- The `Plots/`, the resulting plots are located, both for external plots, and for HDF5 plots that can be read later in visualization software such as VisIt or ParaView. In Output Plots, this is explained in more detail.

If these directories are not present, you will receive error message, so go ahead and create them.

We can now run the code using `mpirun`. For example, with

```
mpirun -np 4 ./PB3D PB3D.input wout.nc
```

4 processes are used. You can find the VMEC file `here`.

There are optional run-time options that can be triggered, such as `--jump_to_sol`, which can be used to easily change solution preferences, such as discretization order, etc. For an overview, see Command-line inputs; they are not treated here.

You will see formatted output on the screen, with indentation that consistent indentation according to the depth of a routine in the program. The same output is also written to an output file, unformatted but still indented (see Log file).

The resulting eigenvalues can also be read in a separate output file (see Eigenvalue summary file).

Take a look at these output files and shuffle a bit with the input parameters if you want. For example, change `n_mod_X` to another value in order to get a different number Fourier harmonics, or `n_r_sol` to change the number of normal grid points in the solution. The solution should not change a lot if you have enough harmonics and normal grid poins.

Note

For VMEC, the magnetic axis is a bit problematic. It is necessary, therefore, to choose `min_r_sol` to be slightly larger than 0; 0.01 for example.

6.5 Run Script Tools

PB3D comes with a bunch of extra run tools, which can greatly improve user experience.

Note

These tools now come in a separate git repository, `PB3D_tools`.

In table 6.3, an overview is given:

Table 6.3 Table 2. extra tools

`run`	Run script that can be used for PB3D and POST with an extensive range of features including • the possibility to do a parameter scan for a variety of input variables • automatic support for schedulers such as PBS and SLURM • support for running on local folders when on a HPC cluster • etc... Note Use of this runscript is recommended, as it will make your life *much* easier, especially when you are working on a cluster.
`extract_jobs_info`	Extracts the eigenvalues from the results of a parameter scan produced by `run` with input variable modifications.
`inspect_SLURM_jobs`	Finds the directories of all the jobs currently running in SLURM and in every directory, it gives the tail of `PB3D.out`. The user can decide to cancel the job.

6.6 Post-processing With POST

POST provides a way to visualize the results from PB3D. Many options are possible, and they can be activated using `plot_` flags (see Input file).

An example is given in `POST.input`

```
1 &INPUTDATA_POST
2     plot_resonance          = .true.
3     plot_magn_grid          = .true.
4     plot_flux_q             = .true.
5     plot_sol_xi             = .true.
6     plot_sol_Q              = .true.
7     plot_E_rec              = .true.
8     plot_B                  = .true.
9     plot_J                  = .true.
10     plot_kappa              = .true.
11     min_r_plot              = 0.0
12     max_r_plot              = 1.0
13     n_theta_plot            = 101
14     min_theta_plot          = 0.00
15     max_theta_plot          = 2.00
16     n_zeta_plot             = 1
17     min_zeta_plot           = 0.00
18     max_zeta_plot           = 0.00
19     POST_style              = 1
20     plot_grid_style         = 0
21     pert_mult_factor_POST   = 0.0000
22     max_tot_mem             = 8000
23 /
24 //! [some extra options for POST]
25     ex_plot_style           = 2
26     PB3D_rich_lvl           = 1
27 //! [some extra options for POST]
```

It can be seen that apart from the `plot_` flags, there are also

- `min_r_plot` and `max_r_plot`, which are the direct equivalent of `min_r_sol` and `max_r_sol` in PB3D.
- `min_theta_plot`, `max_theta_plot` and `n_theta_plot` which indicate the range of the poloidal variable that is plot, as well as the the number of point. There is an equivalent for the toroidal variable.

- `POST_style` that allows the user to change between different styles, such as whether the output grid is chosen to be fill the torus in a regular way, for example, or whether it follows the PB3D equilibrium grid.
- `plot_grid_style`, subsequently, can be used to change the way in which these grids are produced. For example, they could be changed to slab plots or plots with a straightened toroidal angle.
- `pert_mult_factor_POST` can be used to deform the output grids by the perturbation itself.

Also here, there are also some extra options listed below:

```
25    ex_plot_style          = 2
26    PB3D_rich_lvl          = 1
```

They include:

- `ex_plot_style` to indicate which program is to be used for external plotting. Changing it to 2 would engage Bokeh and Mayavi.
- `PB3D_rich_lvl` can be optionally employed to specify explicitely which Richardson level to use for the post-processing.

All of these options are explained in Input file. Again, there are also run-time options that can be found in Command-line inputs.

Now, run the POST program, for example using 4 MPI processes:

```
mpirun -np 4 ./POST POST.input PB3D_out.h5
```

An interesting output is the energy reconstruction, which is activated with `plot_E_rec`. This calculation reinserts the solution normal mode into the perturbed plasma potential terms, in order to check whether the Rayleigh quotient still holds, and to see which terms are dominant and which ones are less important. The results are stored in a separate files as well (see Energy reconstruction file).

If `--do_execute_command_line` was not explicitely used, the actual external plot programs are not used yet, but their usage is given in a shell commands script file (see Shell commands file). Run this shell with

```
./POST_shell_commands.sh
```

If everything is okay, you should now see output in the `Plots/` folder. If an error occurs, please verfiy whether you have appropriate versions of the external plot programs (i.e. GnuPlot, Bokeh, ...)

Also, open some of the `.xmf` files in VisIt or ParaView to explore these plots.

To finish this tutorial, feel free to adjust some input parameters. For example, you can change the output grid to become 3-D by setting `n_zeta_plot` and `max_zeta_plot`, or you could deform it using `pert_mult_factor_POST`.

Note

1. The cylindrical angle φ is often the inverse of the one used in nuclear fusion research, as it can lead to left-handed coordinate systems. In VMEC, it is opted to use it anyway, so it has a left-handed coordinate system. The poloidal angle is chosen to be anticlockwise if you look at a cross-section of the plasma that lies to the right of the Z-axis with the Z-axis pointing up, as in PB3D. In PB3D, however, the toroidal angle $\zeta = -\varphi$ is chosen as the inverse of the cylindrical angle, so it is right-handed.

Chapter 7

Installation

7.1 Introduction

PB3D is written in Fortran, and makes use of multiple numerical libraries:

- `blas / lapack`
  - for basic linear algebra
- `pblas / blacs / scalapack`
  - for parallelized basic linear algebra
- `HDF5`
  - for storage files
  - works in parallel
- `NetCDF`
  - to read input of VMEC
  - sequential
- `PETSc` / `SLEPc`
  - for linear algebra of large, sparse matrices
  - can reach $\sim \mathcal{O}(n)$ complexity
  - Minimal installation instructions:
    1. Configure PETSc using `./configure PETSC_ARCH=complex COPTFLAGS=-O3 CXXOPTFLAGS=-O3 FOPTFLAGS=-O3 \`
       `--with-scalar-type=complex \`
       `--download-metis \`
       `--download-mumps \`
       `--download-parmetis \`
       `--with-scalapack-lib="-L[SCALAPACKDIR]/lib -lscalapack" \`
       `--with-valgrind-dir=/usr \`
       `--with-debugging=no \`
       `--with-fortran-kernels=1`
       (If you want debug, remove `--with-debugging=no`, `--with-fortran-kernels=1`, COPTSFLAGS, CXXOPTFLAGS and FOPTFLAGS, and change PETSC_ARCH to `debug-complex`.)
       (If you use a different HDF5, add `--with-hdf5-dir=[HDF5DIR]`.)

2. follow instructions to do makes and tests.
3. Set global variables `export SLEPC_DIR=[SLEPC DIR]`, `export PETSC_DIR=[PETSC_DIR]` and `export PETSC_ARCH=[debug-]complex` (depending on whether it is debug or not).
4. Configure SLEPc using `./configure`
5. follow instructions to do makes and tests

- StrumPack
  - for linear algebra of structured matrices [1]
  - the vacuum potential energy perturbation calculations use such matrices as they are generated with a $1/r$ kernel.
  - can reach $\sim \mathcal{O}(n \log(n))$ complexity
  - we use the dense version of the package, which for Fortran is 1.1.1.
  - Minimal installation instructions:
    1. Download, untar and go to root folder.
    2. Go to examples, copy `Makefile.gnu` to `Makefile.inc` and adapt:
       (a) for example, it might be necessary to add `-lblacs` before `-lscalapack`
       (b) if you use intel, have a look at the link line advisor, and update the `LIB`
       (c) possibly, you might have to set the `CC` and `FC` compiler commands to, for example, `mpicc` respectively `mpiifort`
    3. Create directories `lib` and `inc`.
    4. Run `make f90_example`
    5. Make a library of `StrumpackDensePackage.o` using `ar -rcs libstrumpack.a *.o` in src.
    6. In inc, create a symlink `libstrumpack.a` to `libstrumpack.a` in src.
    7. In lib, create a symlink `strumpackdensepackage.mod` to `strumpackdensepackage.mod` in examples.

- libstell
  - part of Stellopt suite, which contains VMEC
  - provides routines to read VMEC output data

- pspline
  - Princeton Spline and Hermite Cubic Interpolation Routines
  - Minimal installation instructions:
    1. Export `FFLAGS` en `CFLAGS` if you want to optimize:
       * copy `share/Make.overwrite.sample` to `share/Make.overwrite`
       * Run the gmake below and note the flags, then edit these and put them in `share/Make.overwrite`
       * if GCC:
         - `FFLAGS = -c -O3 -m64 -fno-range-check -fdollar-ok -cpp ; export FFLAGS`
         - `CFLAGS = -c -O3 -m64 ; export CFLAGS`
         - `CXXFLAGS = -c -O3 -m64 ; export CXXFLAGS`
       * if Intel:
         - `FFLAGS = -c -O3 -nowarn -ftz -auto-scalar -traceback -align dcommons`
         - `MPI_FFLAGS = -c -O3 -nowarn -ftz -traceback -align dcommons -auto-scalar` (possibly add `FC=mpiifort`, `CXX=mpicpc`, `CC=mpiicc` if on a cluster)
    2. Compile with `gmake NETCDF_DIR= FORTRAN_VARIANT=[VARIANT]` with `[VARIANT]` either `GCC` or `Intel` (add `DEBUG=1` and remove the FLAGS if debug wanted). Possibly also add `OBJ=$[COMP_DIR]` if on a cluster, with `[COMP_DIR]` the directory where to put the resulting library.
    3. Run tests from `README_Pspline`.

They should probably be installed in this order. On linux distributions such as Ubuntu, they may be available as packages.

Furthermore, PB3D comes bundled with some other, smaller libraries:

- `fftpack`
  - to calculate the fast Fourier transform
- `foul`
  - the Fortran Output Library

These do not have to be installed separately.

7.2 Compilation

When all dependencies are satisfied, the program is then compiled in the standard way:

- Including the headers of all the libraries in the compilation of the object files:
  - This is done using `-I[path_to_library]`.
  - Make sure you add the `-c` option to create only object files.
- Linking with the actual libraries
  - This is done using `-L[path_to_library] -l[library_name]`.

7.3 Makefile Example

```
1 ################################################################################
2 #
3 #   Example makefile for the program PB3D (Peeling Ballooning in 3D)
4 #   \author Author: Toon Weyens
5 #
6 #   Don't forget to set the directories:
7 #       - LIBSTELL_DIR
8 #       - HDF5_DIR
9 #       - NETCDFF_DIR (note: Fortran library)
10 #       - PETSC_DIR
11 #       - SLEPC_DIR
12 #       - STRUMPACK_DIR
13 ################################################################################
14
15 ################################################################################
16 #   Include
17 ################################################################################
18 ## [PETSc and SLEPc trick]
19 include  $(PETSC_DIR)/lib/petsc/conf/variables
20 include  $(SLEPC_DIR)/lib/slepc/conf/slepc_variables
21 ## [PETSc and SLEPc trick]
22
23 ## [PETSc and SLEPc trick inc]
24 INCLUDE = $(PETSC_FC_INCLUDES) $(SLEPC_INCLUDE)
25 ## [PETSc and SLEPc trick inc]
26 ## [Libstell special]
27 INCLUDE += -I$(LIBSTELL_DIR)/libstell_dir
28 ## [Libstell special]
29 INCLUDE += -I$(STRUMPACK_DIR)/include
30 ## [PB3D include]
31 INCLUDE += -I$(PB3D_DIR)/include
32 ## [PB3D include]
33 INCLUDE += -I/usr/include/hdf5/openmpi
34
```

```
35 ###############################################################################
36 #   Link
37 ###############################################################################
38 ## [PB3D libraries]
39 LIB_INTERNAL = libdfftpack.a libfoul.a libbspline.a
40 ## [PB3D libraries]
41
42 LINK := $(LIB_INTERNAL)
43
44 ## [PETSc and SLEPc trick lib]
45 LINK += $(PETSC_LIB)
46 LINK += $(SLEPC_LIB)
47 ## [PETSc and SLEPc trick lib]
48 LINK += $(LIBSTELL_DIR)/libstell.a
49 LINK += -L$(STRUMPACK_DIR)/lib -lstrumpack
50 LINK += -L$(HDF5_DIR) -lhdf5_fortran -lhdf5
51 LINK += -L$(NETCDFF_DIR)/lib -lnetcdff
52 LINK += -Wl,-R$(NETCDFF_DIR)/lib
53 LINK += -lscalapack -lblacs -lblas -lm
54 LINK += -lstdc++ -lmpi_cxx
55
56
57 ###############################################################################
58 #   Compiler
59 ###############################################################################
60 COMPILER=mpifort
61
62
63 ###############################################################################
64 #   Linker
65 ###############################################################################
66 LINKER=mpifort
67
68
69 ###############################################################################
70 #   Compiler flags
71 #   options (used with -D[name]):
72 #       ldebug: debug
73 #       lIB: infiniband
74 #       lwith_gnu: use GNU compiler [default]
75 #       lwith_intel: use INTEL compiler, (checked for version 12.0.2)
76 #   note: INTEL warning 6536 is suppressed, which informs about extra "USE".
77 #   note: INTEL warning 6843 is suppressed, which informs about empty
78 #       intent(out) variables
79 ###############################################################################
80 COMP_FLAGS = -finit-real=snan -g -Og -Wall -Wextra -pedantic \
81     -fimplicit-none -fbacktrace -fno-omit-frame-pointer \
82     -fcheck=all -cpp -Dldebug# debug, profiling with gprof2dot, GCC
83 #COMP_FLAGS = -O3 -fbacktrace -g -fimplicit-none -fno-omit-frame-pointer \
84     #-cpp# optimized, GCC
85
86 #COMP_FLAGS = -O0 -DlIB -Dldebug -g -heap-arrays 100 -recursive \
87     #-ftrapuv -check bounds -check uninit -traceback -implicitnone \
88     #-fno-omit-frame-pointer -cpp -Dlwith_intel -diag-disable 6536 \
89     #-diag-disable 6843# debug, profiling with gprof2dot, INTEL
90 #COMP_FLAGS = -O3 -DlIB -traceback -g -heap-arrays 100 -recursive \
91     #-implicitnone -fno-omit-frame-pointer -cpp -Dlwith_intel \
92     #-diag-disable 6536 -diag-disable 6843# optimized, INTEL
93
94 COMP_FLAGS_EX= -O2 -w
95
96 COMP_FLAGS_F= -O2 -funroll-loops -fexpensive-optimizations
97
98
99 ###############################################################################
100 #   Link flags
101 ###############################################################################
102 LINK_FLAGS = -fPIC -finit-real=snan# debug
103 #LINK_FLAGS = -fPIC# optimized
104
105
106 ###############################################################################
107 #   Prepare
108 ###############################################################################
109 # Add "Modules" and "Libraries" to the search path for the prerequisites
110 VPATH = Modules:Libraries
111
112 # Contains list of source files (.o) and dependencies
113 DEPLIST = PB3D.dep
114 OBJLIST = ObjectList# defines "ObjectFiles"
115
116 # Includes source files and dependency list
117 include $(DEPLIST)# Dependencies of all the objects
118 include $(OBJLIST)# Names of all the objects
119
120
121 ###############################################################################
```

```
122 #   Rules
123 ###########################################################################
124 all:    PB3D POST
125
126 PB3D:   $(ObjectFiles) $(LIB_INTERNAL) PB3D.o
127     $(LINKER) -o $@ $(ObjectFiles) PB3D.o $(LINK) $(LINK_FLAGS)
128
129 POST:   $(ObjectFiles) $(LIB_INTERNAL) POST.o
130     $(LINKER) -o $@ $(ObjectFiles) POST.o $(LINK) $(LINK_FLAGS)
131
132 libdfftpack.a:  dfft.o
133     ar -rcs libdfftpack.a dfft.o
134
135 libfoul.a:  foul.o
136     ar -rcs libfoul.a foul.o
137
138 libbspline.a:   bspline_sub_module.o
139     ar -rcs libbspline.a bspline_sub_module.o
140
141 %.o: %.f90
142     $(COMPILER) $(INCLUDE) $(COMP_FLAGS) -c $<
143
144 %.o: %.f
145     $(COMPILER) $(COMP_FLAGS_F) -c $<
146
147 dfft.o: dfft.f
148     $(COMPILER) $(COMP_FLAGS_EX) -c $<
149
150 foul.o: foul.f90
151     $(COMPILER) $(COMP_FLAGS_EX) -c $<
152
153 bspline_sub_module.o: bspline_sub_module.f90
154     $(COMPILER) $(COMP_FLAGS_EX) -c $<
155
156 clean:
157     @rm -f *.o *.a *.mod *~ fort.*
158
159 clean_all:
160     @rm -f *.o *.mod *~ fort.* PB3D POST
```

Note

1. PETSc and SLEPc don't like to be included in another makefile. The trick is to include two files:
   ```
   19 include  $(PETSC_DIR)/lib/petsc/conf/variables
   20 include  $(SLEPC_DIR)/lib/slepc/conf/slepc_variables
   ```
   which will load the variables `PETSC_FC_INCLUDES` and `SLEPC_INCLUDE`, used in
   ```
   24 INCLUDE = $(PETSC_FC_INCLUDES) $(SLEPC_INCLUDE)
   ```
   as well as the variables `PETSC_LIB` and `SLEPC_LIB`, used in
   ```
   45 LINK += $(PETSC_LIB)
   46 LINK += $(SLEPC_LIB)
   ```
2. There are versions of libstell that do not use the standard convention. In this case you have to look for the `*.mod` files. In the example makefile this is done with
   ```
   27 INCLUDE += -I$(LIBSTELL_DIR)/libstell_dir
   ```
   instead of the standard `inc` directory.
3. In
   ```
   31 INCLUDE += -I$(PB3D_DIR)/include
   ```
   there are includefiles that contain macros and wrappers specifically for PB3D.
4. In
   ```
   39 LIB_INTERNAL = libdfftpack.a libfoul.a libbspline.a
   ```
   linking is done with external libraries that are bundled with PB3D.

Chapter 8

Todo List

Module vac_ops

The vacuum part of PB3D is still under construction and not usable yet.

Subprogram vac_ops::calc_gh **(vac, n_r_in, lims_r_in, x_vec_in, G_in, H_in)**

For 3-D vacuums, step sizes are constant. Subsequent Richardson levels should therefore make use of the fact that the contribution to the points inherited from the previous levels can be just scaled by 1/2 and do not need to be recalculated. Currently, the copy is done correctly in interlaced_vac_copy(), but they are afterwards overwritten.

Appendices

Appendix A

Programs

A.1 /opt/PB3D/POST.f90 Program Rerefence

Peeling Ballooning in 3D: postprocessing.

Author

Toon Weyens, Contact: weyenst@gmail.com

Version

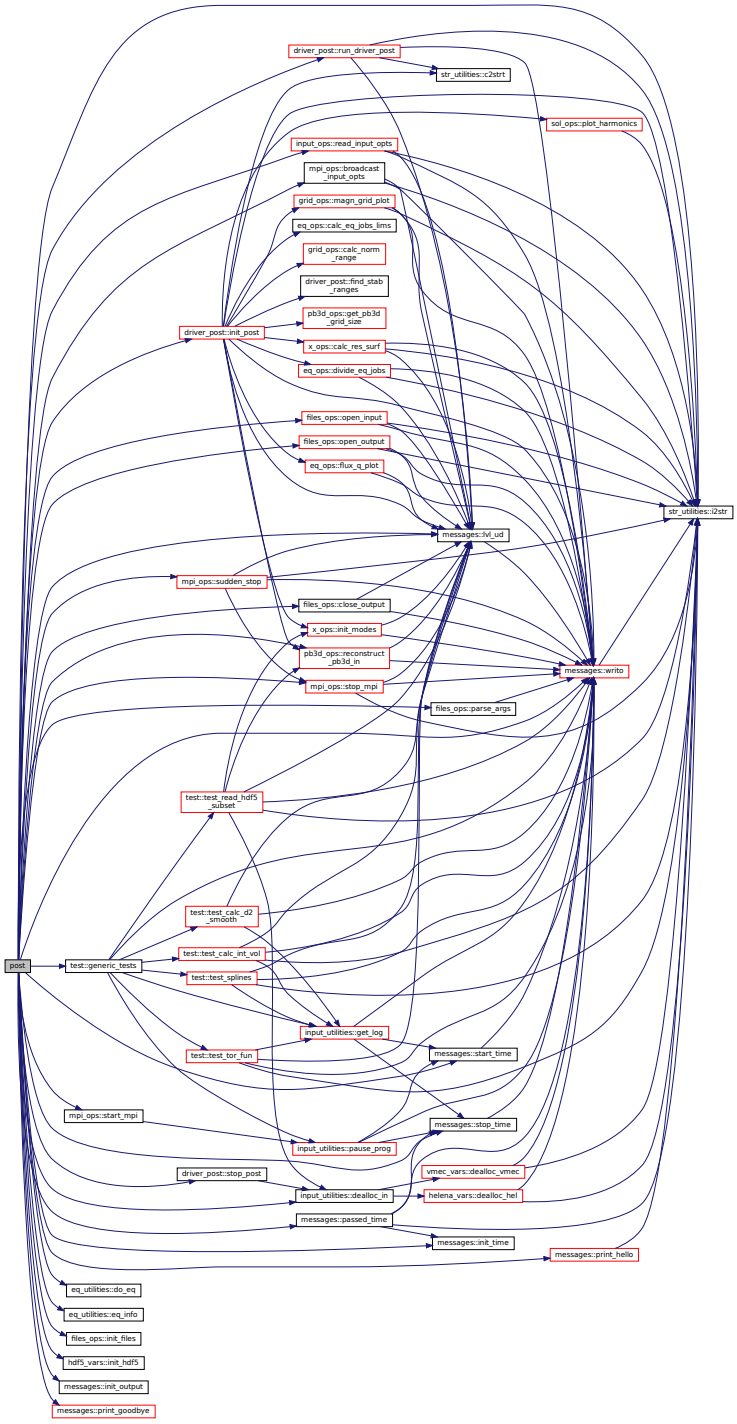
2.41

See also

References: [16] [17]

Definition at line 40 of file POST.

Here is the call graph for this function:



A.2 /opt/PB3D/PB3D.f90 Program Rerefence

Peeling Ballooning in 3D.

Author

Toon Weyens, Contact: weyenst@gmail.com

Version

2.41

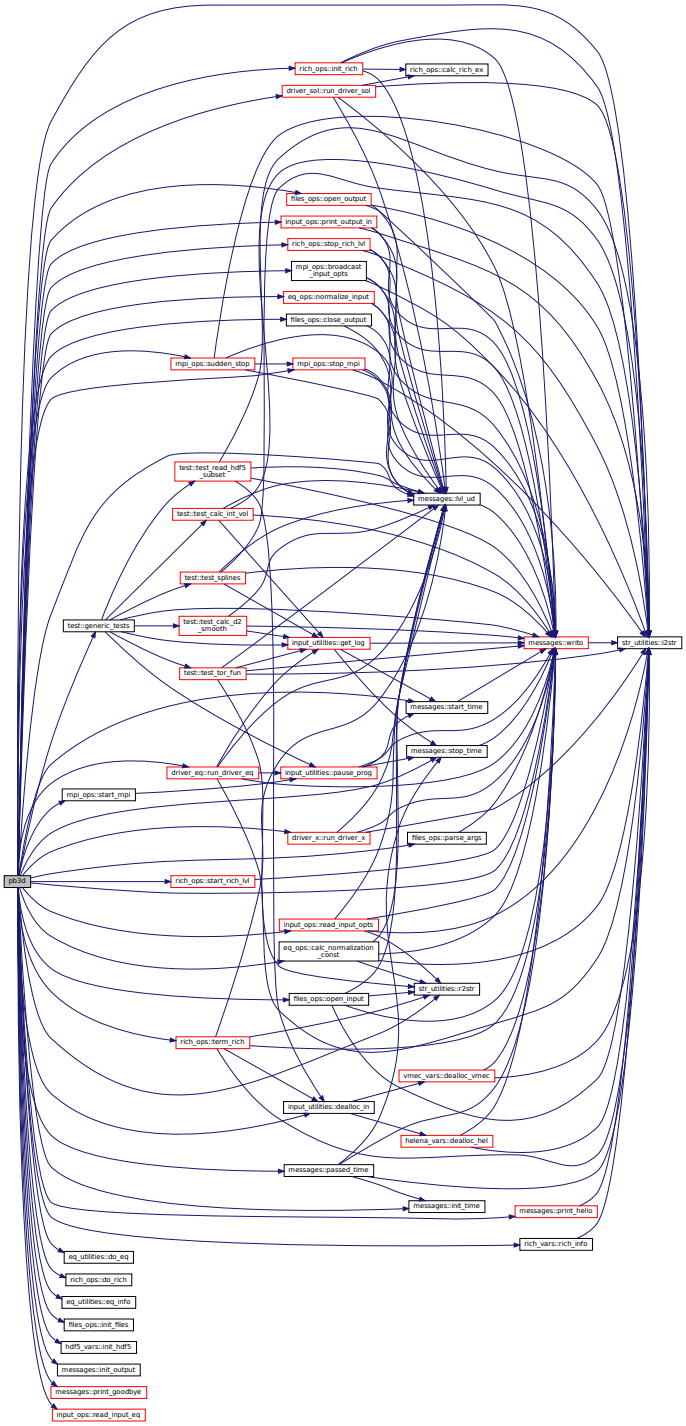
Date

See also

References: [16] [17]

Definition at line 40 of file PB3D.

Here is the call graph for this function:



Appendix B

Modules

B.1 driver_eq Module Reference

Driver of the equilibrium part of PB3D.

Functions/Subroutines

- integer function, public run_driver_eq (grid_eq_out, grid_eq_B_out, eq_1_out, eq_2_out, vac)
  Main driver of PB3D equilibrium part.

B.1.1 Detailed Description

Driver of the equilibrium part of PB3D.

B.1.2 Function/Subroutine Documentation

B.1.2.1 run_driver_eq()

```
integer function, public driver_eq::run_driver_eq (
               type(grid_type), intent(inout), target grid_eq_out,
               type(grid_type), intent(inout), pointer grid_eq_B_out,
               type(eq_1_type), intent(inout) eq_1_out,
               type(eq_2_type), intent(inout) eq_2_out,
               type(vac_type), intent(inout) vac )
```

Main driver of PB3D equilibrium part.

- sets up ([out] means for output):
  - `grid_eq` [out] (for HELENA, only first Richardson level)
  - `grid_eq_B` [out] (for VMEC, equal to grid_eq_out)
  - `eq_1` [out] (only first Richardson level)
  - `eq_2` [out] (for HELENA, only first Richardson level) where output means
  - on the equilibrium grid if `X_grid` style is 1,3 (no change).
  - on a redistributed grid if `X_grid_style` is 2
- writes to HDF5:
  - `grid_eq` (for HELENA, only first Richardson level)
  - `grid_eq_B` (for VMEC, equal to grid_eq)
  - `eq_1` (only first Richardson level)
  - `eq_2` (only for HELENA)
- deallocates:
  - `grid_eq` [out] before setting up
  - `grid_B_eq` [out] before setting up
  - `eq_2` [out] before setting up

Returns

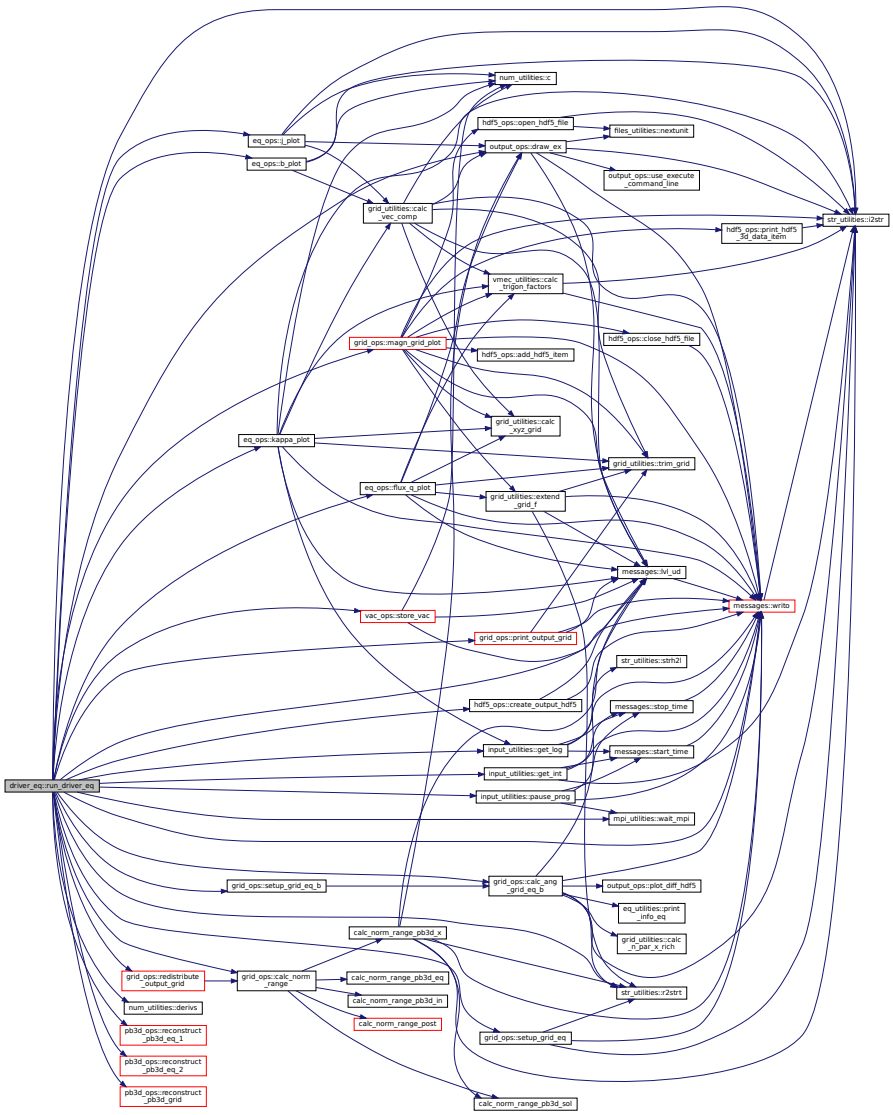
ierr

Parameters

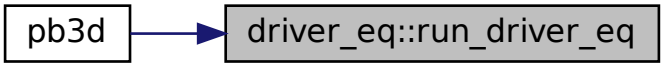
`in,out`	*grid_eq_out*	output equilibrium grid
`in,out`	*grid_eq_b_out*	output field-aligned equilibrium grid
`in,out`	*eq_1_out*	flux equilibrium variables in output grid
`in,out`	*eq_2_out*	metric equilibrium variables in output grid
`in,out`	*vac*	vacuum variables

Definition at line 49 of file driver_eq.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.2 driver_post Module Reference

Main driver of PostProcessing of program Peeling Ballooning in 3D.

Functions/Subroutines

- integer function, public init_post ()
  Initializes the POST driver.
- integer function, public run_driver_post ()
  The main driver routine for postprocessing.
- subroutine, public stop_post ()
  Cleans up main driver for postprocessing.
- integer function open_decomp_log ()
  Opens the decomposition log file.
- integer function write_decomp_log (X_id, E_pot_int, E_kin_int)
  Write to decomposition log file.
- subroutine find_stab_ranges (sol, min_id, max_id, last_unstable_id)
  finds the plot ranges `min_id` and `max_id`.
- subroutine plot_sol_val_comp (sol_val_comp)
  Plots difference between Eigenvalues and energy fraction.
- integer function setup_out_grids (grids_out, XYZ_eq, XYZ_sol)
  Sets up the output grids for a particular parallel job.

B.2.1 Detailed Description

Main driver of PostProcessing of program Peeling Ballooning in 3D.

B.2.2 Function/Subroutine Documentation

B.2.2.1 find_stab_ranges()

```
subroutine driver_post::find_stab_ranges (
               type(sol_type), intent(in) sol,
               integer, dimension(3), intent(inout) min_id,
               integer, dimension(3), intent(inout) max_id,
               integer, intent(inout) last_unstable_id )
```

finds the plot ranges `min_id` and `max_id`.

There are three ranges, calculated using n_sol_plotted, which indicates:

1. how many of the first EV's in the unstable range
2. how many of the last EV's in the unstable range
3. how many of the first EV's in the stable range
4. how many of the last EV's in the stable range

have to be plotted.

This yields maximally three different ranges: One starting at the first unstable EV, one centered around the zero of the EV's and one ending at the last EV. These ranges can be disjoint but do not have to be.
Also, it is possible that a range does not exist, for example if there are no unstable EV's.

Note

A negative value for the elements in n_sol_plotted means ”all values in range”:

1. or 2. full unstable range
2. or 4. full stable range

Parameters

in	*sol*	solution variables
in,out	*min_id*	min. index of range 1, 2 and 3
in,out	*max_id*	max. index of range 1, 2 and 3
in,out	*last_unstable↩ _id*	index of last unstable EV

Definition at line 1037 of file driver_POST.f90.

Here is the caller graph for this function:

B.2.2.2 init_post()

```
integer function, public driver_post::init_post
```

Initializes the POST driver.

- set up preliminary variables
  - global variables
  - read grids (full)
  - eq_1 (full) and n & m (full)
- set up output grids:
  - POST_style = 1: extended grid
  - POST_style = 2: field-aligned grid
- clean up
- set up final variables
  - normal limits
  - read grids (divided), eq_1 (divided) and sol (divided) variables
- 1-D output plots
  - resonance plot
  - flux quantities plot
  - magnetic grid plot

- prepare Eigenvalue plots
  - calculates resonant surfaces
  - plots Eigenvalues
  - finds stability ranges for all Eigenvalues to be plot
  - plots harmonics for every Eigenvalue
  - calculates the parallel ranges of the equilibrium jobs
- clean up

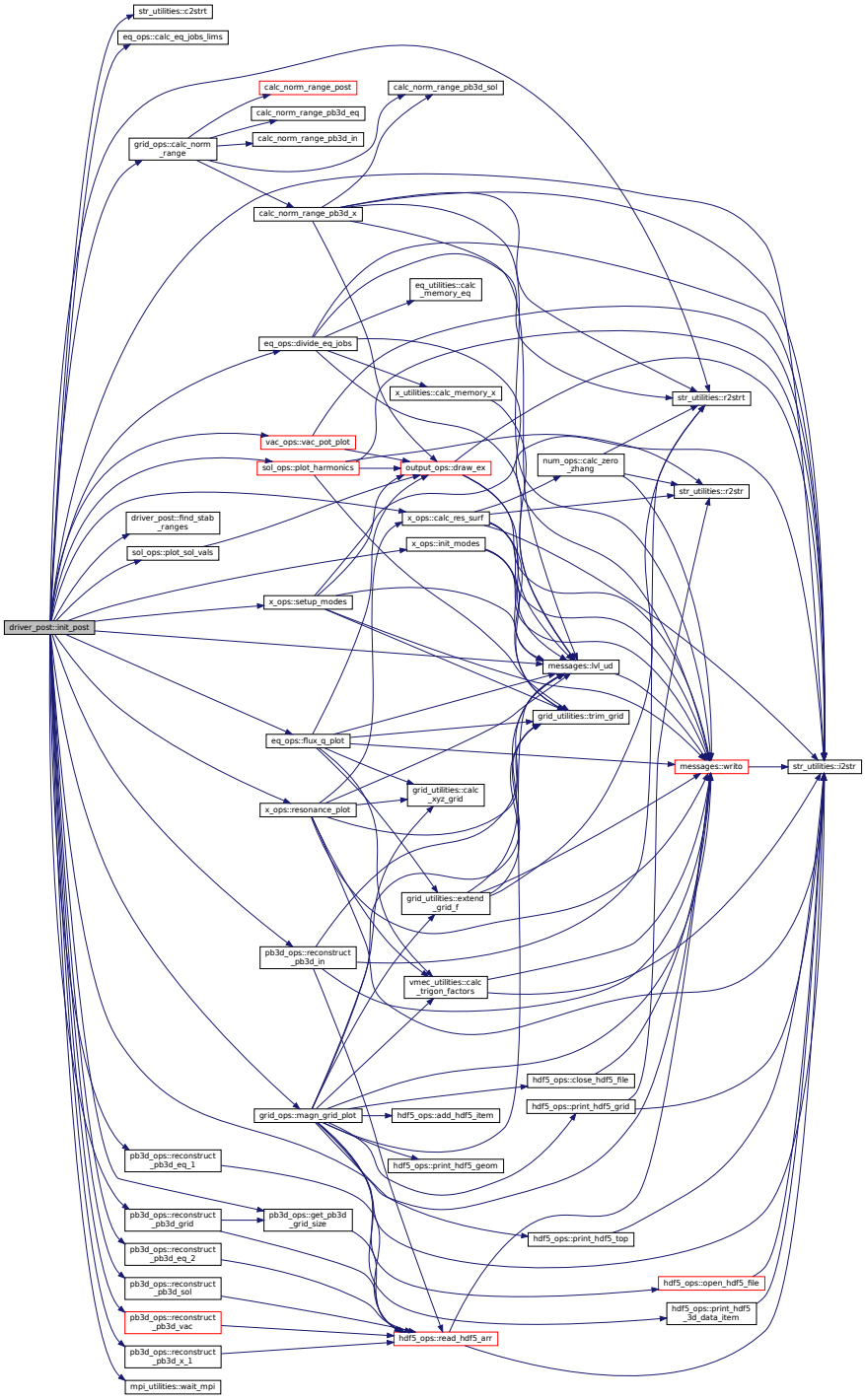
In the actual driver, more detailed plots are possibly made for all requested Eigenvalues if `full_output`.

Returns

ierr

Definition at line 77 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.2.2.3 open_decomp_log()

```
integer function driver_post::open_decomp_log
```

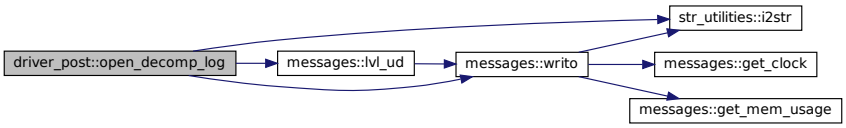
Opens the decomposition log file.

Returns

ierr

Definition at line 841 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.2.2.4 plot_sol_val_comp()

```
subroutine driver_post::plot_sol_val_comp (
              complex(dp), dimension(:,:,:), intent(inout) sol_val_comp )
```

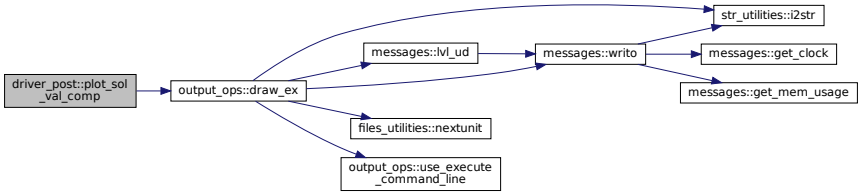
Plots difference between Eigenvalues and energy fraction.

Parameters

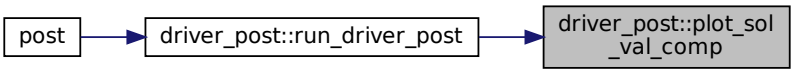
in,out	*sol_val_comp*	fraction between total E_{pot} and E_{kin}, compared with EV

Definition at line 1113 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.2.2.5 run_driver_post()

```
integer function, public driver_post::run_driver_post
```

The main driver routine for postprocessing.

Note

The PB3D output is given on different grids for different styles of the equilibrium model:

- VMEC: field-aligned grid on which EV problem has been solved.
- HELENA: output grid on which equilibrium and metric variables are tabulated.

Furthermore, if Richardson extrapolation is used, the VMEC grids and variables are contained in multiple HDF5 variables. These variables are needed here both in a field-aligned and a plot grid.

A consequence of the above is that for VMEC the field-aligned output is already given, but the output on the plot grid has to be calculated from scratch, while for HELENA both outputs have to be interpolated from the output tables.

The general workflow is as follows:

- take a subset of the output grids for the current equilibrium job.

- for `POST_style = 1` (extended grid):

  - VMEC: recalculate variables

  - HEL: interpolate variables for `POST_style = 2` (B-aligned grid):

  - VMEC: read subset of variables

  - HEL: interpolate variables

- create helper variables

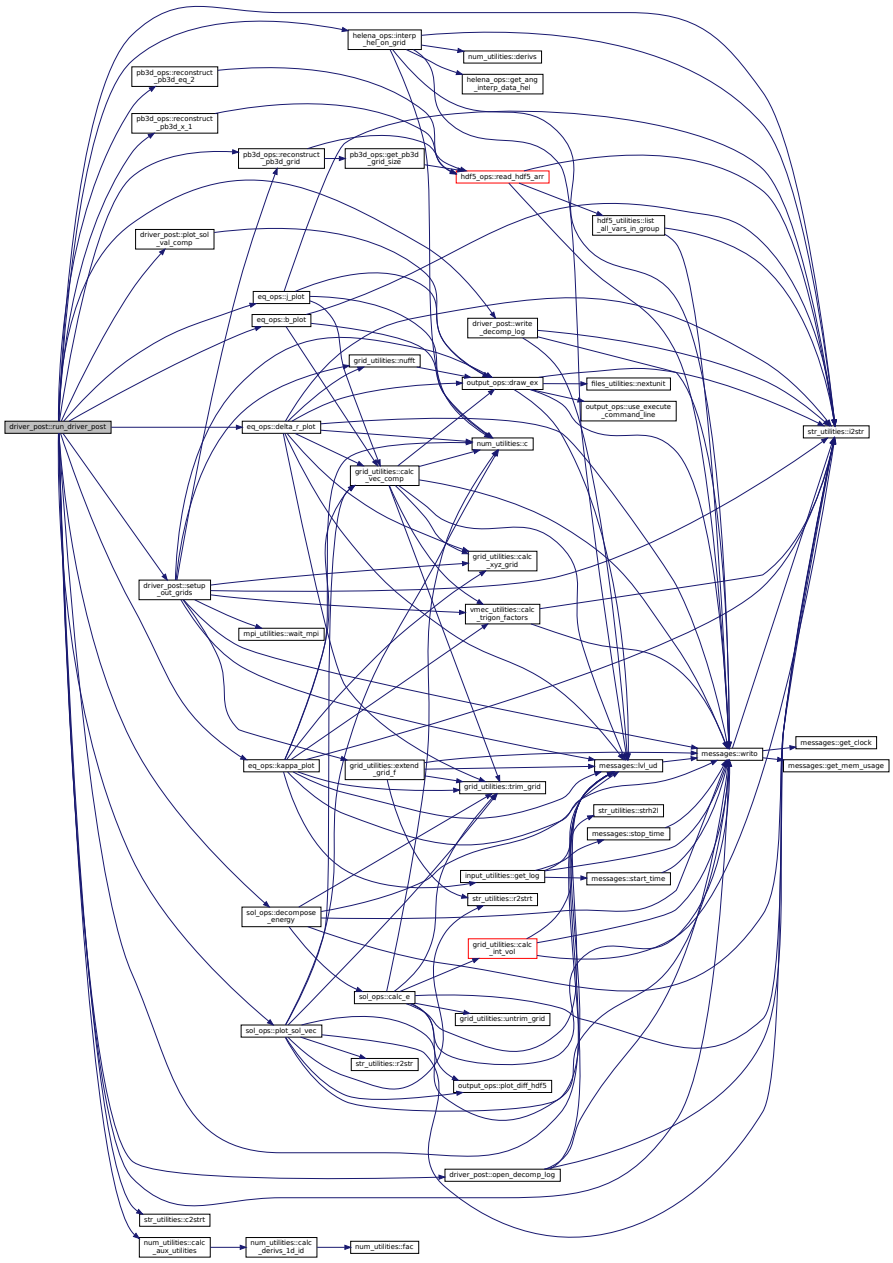
- create plots and outputs

Returns

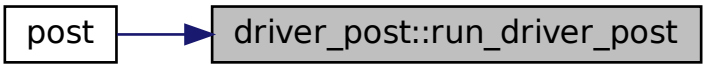
ierr

Definition at line 542 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.2.2.6 setup_out_grids()

```
integer function driver_post::setup_out_grids (
               type(grid_type), dimension(3), intent(inout) grids_out,
               real(dp), dimension(:,:,:,:), intent(inout), allocatable XYZ_eq,
               real(dp), dimension(:,:,:,:), intent(inout), allocatable XYZ_sol )
```

Sets up the output grids for a particular parallel job.

Three grids are returned:

- eq
- X
- sol

The normal coordinates of these grids correspond to the one used in PB3D. For X this is determined by `x_grid_style`.

The angular components of the eq and X grids is determined by `post_style`:

- 1: extended grid with `min_theta_plot`, `max_theta_plot`, `min_zeta_plot` and `max_zeta_plot`.
- 2: PB3D grid.

Returns

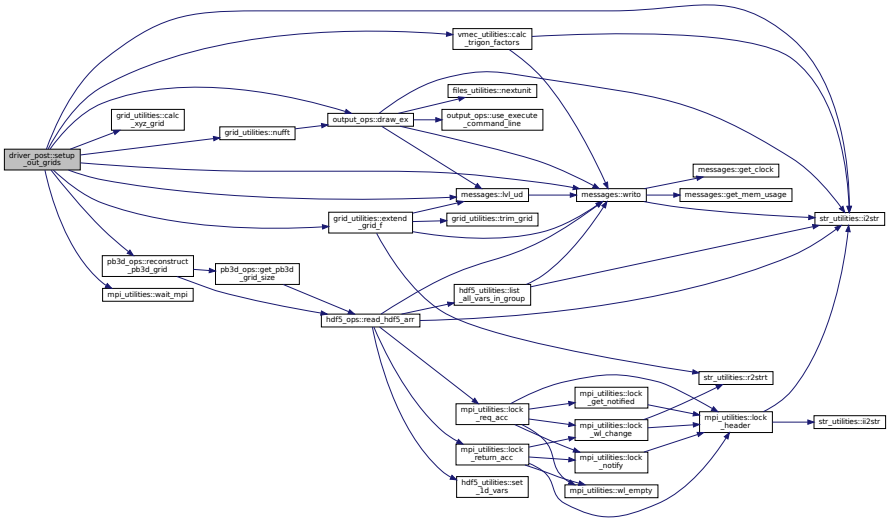
ierr

Parameters

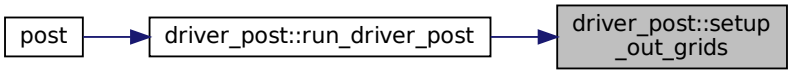
`in,out`	*grids_out*	eq, X and sol output grids (full parallel and normal range)
`in,out`	*xyz_eq*	`X`, `Y` and `Z` on output equilibrium grid
`in,out`	*xyz_sol*	`X`, `Y` and `Z` on output solution grid

Definition at line 1186 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



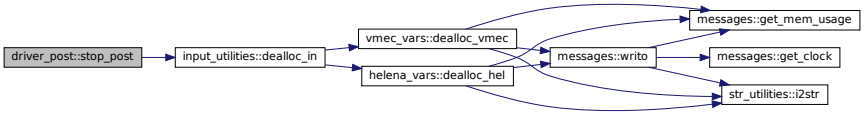
B.2.2.7 stop_post()

`subroutine, public driver_post::stop_post`

Cleans up main driver for postprocessing.

Definition at line 816 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.2.2.8 write_decomp_log()

```
integer function driver_post::write_decomp_log (
               integer, intent(in) X_id,
               complex(dp), dimension(7), intent(in) E_pot_int,
               complex(dp), dimension(2), intent(in) E_kin_int )
```

Write to decomposition log file.

Returns

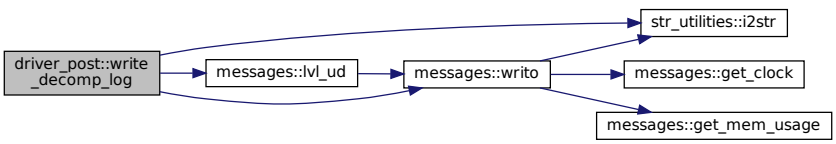
ierr

Parameters

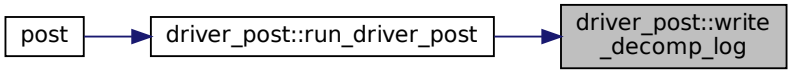
in	*x_id*	nr. of Eigenvalue
in	*e_pot_int*	E_pot integrated for requested solutions
in	*e_kin_int*	E_kin integrated for requested solutions

Definition at line 922 of file driver_POST.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.3 driver_sol Module Reference

Driver of the solution part of PB3D.

Functions/Subroutines

- integer function, public run_driver_sol (grid_eq, grid_X, grid_sol, X, vac, sol)
  Main driver of PB3D solution part.
- integer function interp_v (mds_i, grid_i, X_i, mds_o, grid_o, X_o)
  Interpolate tensorial perturbation quantities in the third dimension.

B.3.1 Detailed Description

Driver of the solution part of PB3D.

B.3.2 Function/Subroutine Documentation

B.3.2.1 interp_v()

```
integer function driver_sol::interp_v (
               type(modes_type), intent(in), target mds_i,
               type(grid_type), intent(in) grid_i,
               type(x_2_type), intent(in), target X_i,
               type(modes_type), intent(in), target mds_o,
               type(grid_type), intent(in) grid_o,
               type(x_2_type), intent(inout), target X_o )
```

Interpolate tensorial perturbation quantities in the third dimension.

The input grid should not be divided, though the output grid can be.

The procedure considers all possible mode number combinations. For `X_style` 2 (fast), each secondary mode only lives in a certain normal range of the plasma. Therefore, each secondary mode pair also has a limited normal range, given by the overlap of the ranges of the members.

The interpolated mode number combinations have a normal range that might slightly differ from the input ranges, in which case extrapolation can be done if the method allows for it.

Note

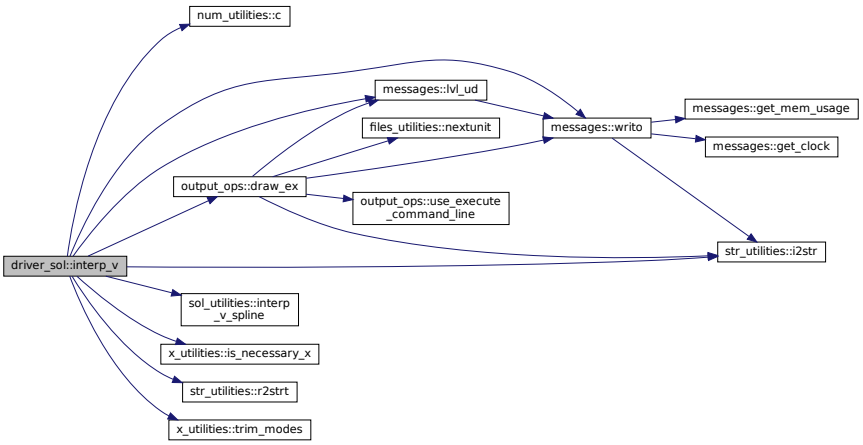
If the input grid is too coarse, but the interpolated grid is not, it is in theory possible that there are mode numbers and therefore mode number pairs that exist in the input grid, though they do so in the interpolated grid. In this case, they can not be calculated and are currently set to zero. However, the current implementation of setup_modes() produces an error if less than the full range of mode numbers is accurately present.

Parameters

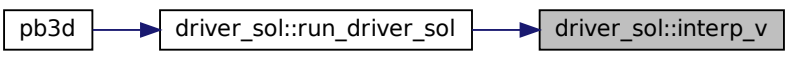
in	*mds_i*	general modes variables for input
in	*grid_i*	grid at which X_i is tabulated
in	*x_i*	tensorial perturbation variable on input grid
in	*mds_o*	general modes variables for output
in	*grid_o*	grid at which X_o is interpolated
in,out	*x_o*	interpolated tensorial perturbation variable

Definition at line 284 of file driver_sol.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.3.2.2 run_driver_sol()

```
integer function, public driver_sol::run_driver_sol (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in), target grid_X,
               type(grid_type), intent(inout) grid_sol,
               type(x_2_type), intent(in) X,
               type(vac_type), intent(inout) vac,
               type(sol_type), intent(inout) sol )
```

Main driver of PB3D solution part.

- sets up:
  - `grid_sol` (only first Richardson level)
  - `sol`
- writes to HDF5:
  - `grid_sol` (only first Richardson level)
  - `sol`
- deallocates:
  - sol before setting up (but after guess)

Returns

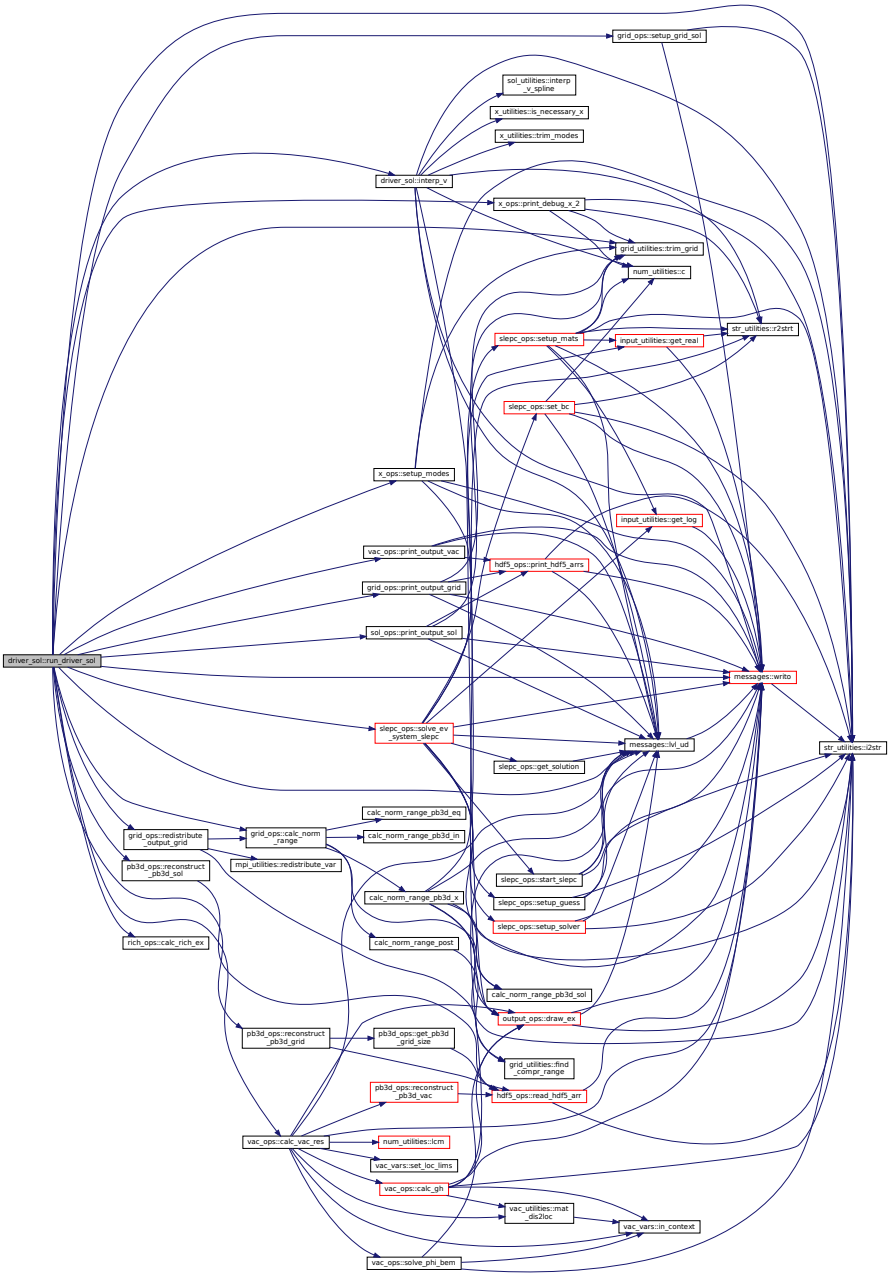
ierr

Parameters

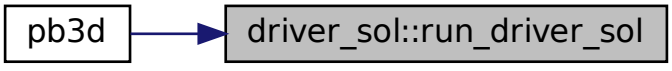
`in`	*grid_eq*	equilibrium grid
`in`	*grid_x*	perturbation grid
`in,out`	*grid_sol*	solution grid
`in`	*x*	integrated tensorial perturbation variables
`in,out`	*vac*	vacuum variables
`in,out`	*sol*	solution variables

Definition at line 43 of file driver_sol.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.4 driver_x Module Reference

Driver of the perturbation part of PB3D.

Functions/Subroutines

- integer function, public run_driver_x (grid_eq, grid_eq_B, grid_X, grid_X_B, eq_1, eq_2, X_1, X_2)
  Main driver of PB3D perturbation part.
- integer function run_driver_x_0 (grid_eq, grid_eq_B, grid_X, grid_X_B, eq_1, eq_2, eq_2_B)
  part 0 of driver_x: perturbation grid as well as reconstruction of variables.
- integer function run_driver_x_1 (grid_eq, grid_X, eq_1, eq_2, X_1)
  Part 1 of driver_x: Vectorial jobs.
- integer function run_driver_x_2 (grid_eq_B, grid_X, grid_X_B, eq_1, eq_2_B, X_1, X_2_int)
  Part 2 of driver_X: Tensorial jobs.
- subroutine print_info_x_2 ()
  Prints information for tensorial perturbation job.

B.4.1 Detailed Description

Driver of the perturbation part of PB3D.

B.4.2 Function/Subroutine Documentation

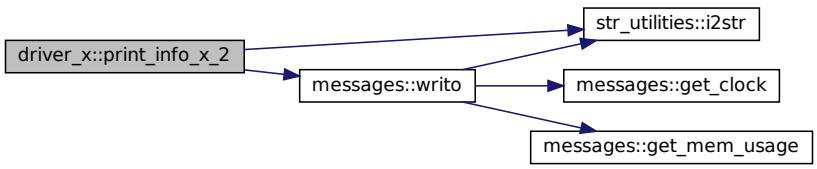
B.4.2.1 print_info_x_2()

`subroutine driver_x::print_info_x_2`

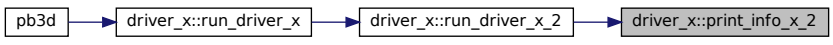
Prints information for tensorial perturbation job.

Definition at line 699 of file driver_X.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.4.2.2 run_driver_x()

```
integer function, public driver_x::run_driver_x (
               type(grid_type), intent(in), target grid_eq,
               type(grid_type), intent(in), pointer grid_eq_B,
               type(grid_type), intent(inout), target grid_X,
               type(grid_type), intent(inout), pointer grid_X_B,
               type(eq_1_type), intent(in), target eq_1,
               type(eq_2_type), intent(inout), target eq_2,
               type(x_1_type), intent(inout) X_1,
               type(x_2_type), intent(inout) X_2 )
```

Main driver of PB3D perturbation part.

- sets up:
  - [0] `grid_X` (for HELENA, only first Richardson level)
  - [0] `grid_X_B` (for VMEC, equal to grid_X_out)
  - [1] `X_1`
  - [2] `X_2`
- writes to HDF5:
  - [0] `grid_X` (for HELENA, only first Richardson level)
  - [0] `grid_X_B` (for VMEC, equal to grid_X)
  - [1] `X_1` (only for HELENA, only first Richardson level)
  - [2] `X_2`
- deallocates:
  - `X_1` before setting up
  - `grid_X` before setting up
  - `grid_X_B` before setting up

([x] indicates driver x)

Note

`eq_2` needs to be intent(inout) because interp_HEL_on_grid() requires this for generality. The variable is not modified in this driver, though.

Returns

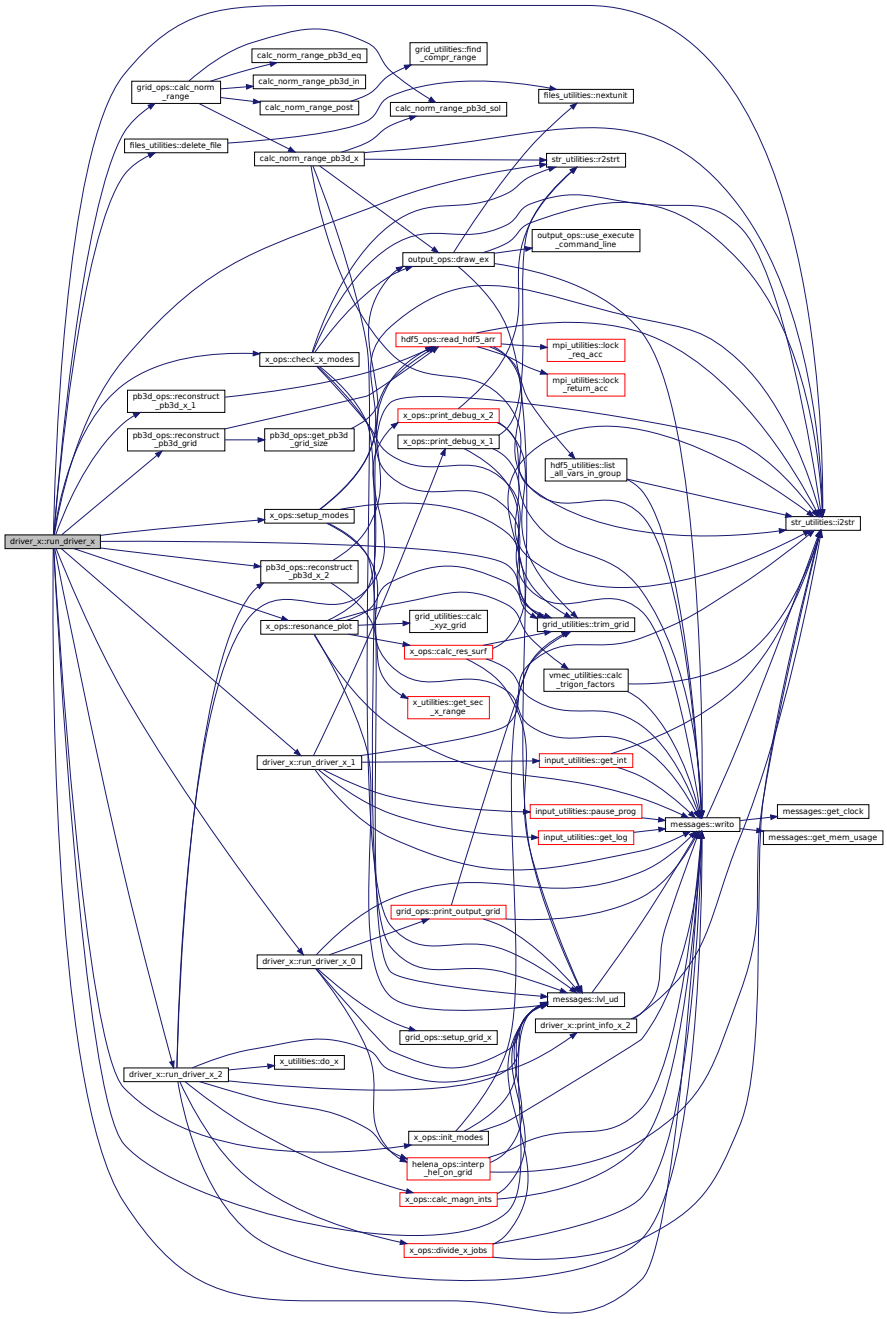
ierr

Parameters

`in`	*grid_eq*	equilibrium grid (should be in but needs inout for interp_HEL_on_grid)
`in`	*grid_↩ eq_b*	field-aligned equilibrium grid (should be in but needs inout for interp_HEL_on_grid)
`in,out`	*grid_x*	perturbation grid
`in,out`	*grid_x_b*	field-aligned perturbation grid
`in`	*eq_1*	flux equilibrium variables
`in,out`	*eq_2*	metric equilibrium variables (should be in but needs inout for interp_HEL_on_grid)
`in,out`	*x_1*	vectorial perturbation variables
`in,out`	*x_2*	tensorial perturbation variables

Definition at line 59 of file driver_X.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.4.2.3 run_driver_x_0()

```
integer function driver_x::run_driver_x_0 (
                type(grid_type), intent(in), target grid_eq,
                type(grid_type), intent(in), pointer grid_eq_B,
                type(grid_type), intent(inout), target grid_X,
                type(grid_type), intent(inout), pointer grid_X_B,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(inout), target eq_2,
                type(eq_2_type), intent(inout), pointer eq_2_B )
```

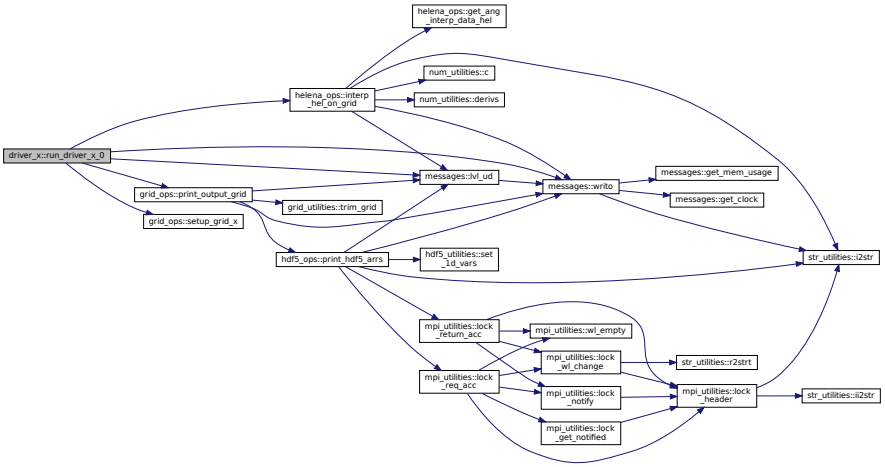
part 0 of driver_x: perturbation grid as well as reconstruction of variables.

Parameters

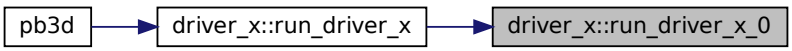
in	*grid_eq*	equilibrium grid
in	*grid_eq_b*	field-aligned equilibrium grid
in,out	*grid_x*	perturbation grid
in,out	*grid_x_b*	field-aligned perturbation grid
in	*eq_1*	flux equilibrium variables
in,out	*eq_2*	metric equilibrium variables
in,out	*eq_2_b*	field-aligned metric equilibrium variables

Definition at line 267 of file driver_X.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.4.2.4 run_driver_x_1()

```
integer function driver_x::run_driver_x_1 (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               type(x_1_type), intent(inout) X_1 )
```

Part 1 of driver_x: Vectorial jobs.

Note

Everything is done here in the original grids, not necessarily field-aligned.

Returns

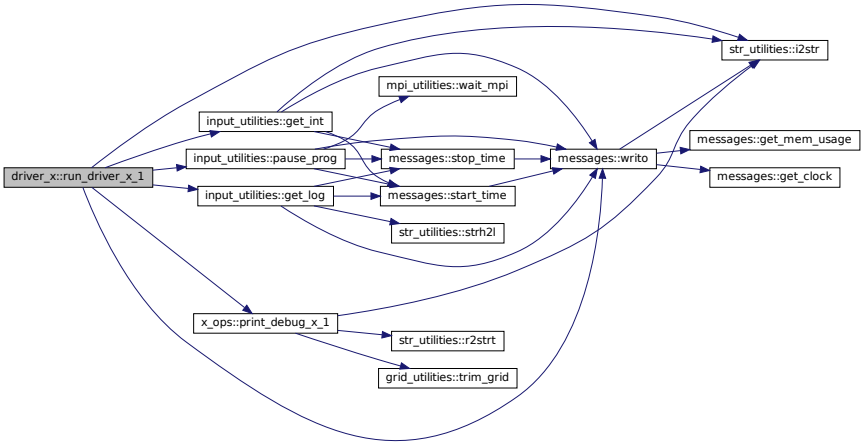
ierr

Parameters

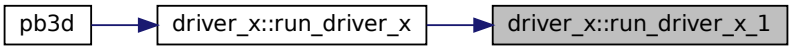
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables
in,out	*x_1*	vectorial X variables

Definition at line 376 of file driver_X.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.4.2.5 run_driver_x_2()

```
integer function driver_x::run_driver_x_2 (
               type(grid_type), intent(in), pointer grid_eq_B,
               type(grid_type), intent(in), target grid_X,
               type(grid_type), intent(in), pointer grid_X_B,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in), pointer eq_2_B,
               type(x_1_type), intent(in) X_1,
               type(x_2_type), intent(inout) X_2_int )
```

Part 2 of driver_X: Tensorial jobs.

Note

Everything is done in the field-aligned grids, where HELENA vectorial perturbation variables are first interpolated.

Returns

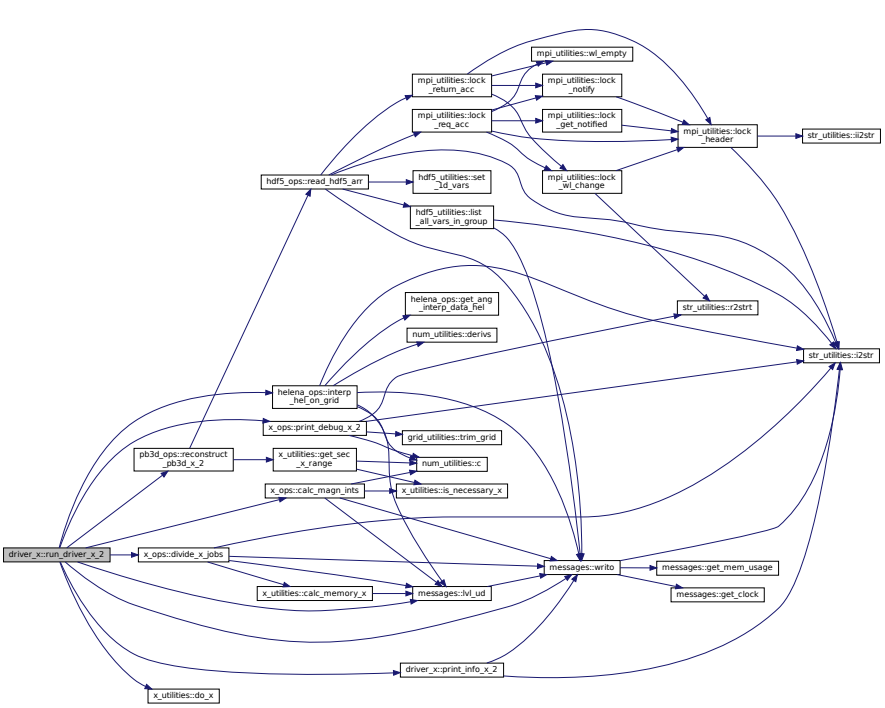
ierr

Parameters

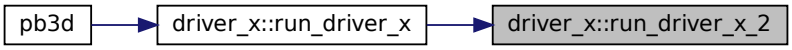
in	*grid_eq_b*	field-aligned equilibrium grid
in	*grid_x*	perturbation grid
in	*grid_x_b*	field-aligned perturbation grid
in	*eq_1*	flux equilibrium variables
in	*eq_2_b*	field-aligned metric equilibrium variables
in	*x_1*	vectorial X variables
in,out	*x_2_int*	tensorial X variables

Definition at line 526 of file driver_X.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.5 dtorh Module Reference

Calculation of toroidal functions $P^m_{n-1/2}(z)$ and $Q^m_{n-1/2}(z)$.

Functions/Subroutines

- integer function, public dtorh1 (z, m, nmax, pl, ql, newn, mode, ipre)
  Calculates toroidal harmonics of a fixed order `m` and degrees up to `nmax`.

B.5.1 Detailed Description

Calculation of toroidal functions $P^m_{n-1/2}(z)$ and $Q^m_{n-1/2}(z)$.

Note

Copied and adapted from the DTORH1 routine by Segura and Gil [14] .

B.5.2 Function/Subroutine Documentation

B.5.2.1 dtorh1()

```
integer function, public dtorh::dtorh1 (
               real(dp), intent(in) z,
               integer, intent(in) m,
               integer, intent(in) nmax,
               real(dp), dimension(0:nmax), intent(inout) pl,
               real(dp), dimension(0:nmax), intent(inout) ql,
               integer, intent(inout) newn,
               integer, intent(in), optional mode,
               integer, intent(in), optional ipre )
```

Calculates toroidal harmonics of a fixed order `m` and degrees up to `nmax`.

Optionally, the `mode` can be specified to be different from 0 [default]

- if `mode=1`:
  - The set of functions evaluated is:

$$\frac{P^m_{n-1/2}(z)}{\Gamma(m+1/2)}, \quad \frac{Q^m_{n-1/2}(z)}{\Gamma(m+1/2)},$$

    which are respectively stored in the arrays `pl(n)`, `ql(n)`.
  - newn refers to this new set of functions.
  - Note 1. and note 2. also apply in this case.
- if `mode=2`:
  - The code performs as for mode 1, but the restriction $z < 20$.
  - For the evaluation of the continued fraction is not considered.

Also, the parameter `ipre` can be used to select a different precision:

- For `ipre=1`, the precision is 10^{-12}, taking $\epsilon < 10^{-12}$
- For `ipre=2`, the precision is 10^{-8}, taking $\epsilon < 10^{-8}$

where ϵ controls the accuracy.

Warning

Use `mode` 2 only if high `m` 's for $z > 20$ are required. The evaluation of the cf may fail to converge for too high `z` 's.

Note

1. For a precision of 10^{-12}, if $z > 5$ and $z/m > 0.22$, the code uses a series expansion for `pl(0)`. When $z < 20$ and $z/m < 0.22$ a continued fraction is applied.
2. For a precision of 10^{-8}, if $z > 5$ and $z/m > 0.12$, the code uses a series expansion for `pl(0)`. When $z < 20$ and $z/m < 0.12$ a continued fraction is applied.

Returns

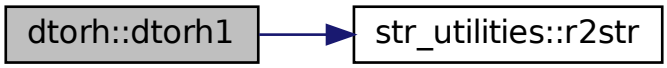
ierr

Parameters

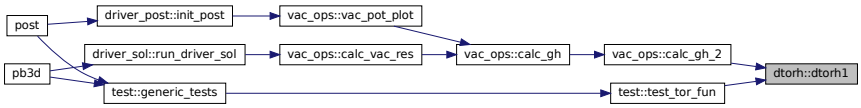
`in`	*z*	(real) point at which toroidal harmonics are evaluated
`in`	*m*	order of toroidal harmonics (`m`>`0`)
`in`	*nmax*	maximum degree of toroidal harmonics (`nmax`>`0`)
`in,out`	*pl*	toroidal harmonics of first kind $P^m_{n-1/2}(z)$
`in,out`	*ql*	toroidal harmonics of second kind $Q^m_{n-1/2}(z)$
`in,out`	*newn*	maximum order of functions calculated when `pl(nmax+1)`>`1/tiny` with `tiny`= 10^{-290}
`in`	*mode*	mode that controls output
`in`	*ipre*	precision

Definition at line 75 of file dtorh.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6 eq_ops Module Reference

Operations on the equilibrium variables.

Interfaces and Types

- interface calc_eq
  Calculate the equilibrium quantities on a grid determined by straight field lines.
- interface calc_g_c
  Calculate the lower metric elements in the C(ylindrical) coordinate system.
- interface calc_g_f
  Calculate the metric coefficients in the F(lux) coordinate system.
- interface calc_g_h
  Calculate the lower metric coefficients in the equilibrium H(ELENA) coordinate system.
- interface calc_g_v
  Calculate the lower metric coefficients in the equilibrium V(MEC) coordinate system.
- interface calc_jac_f
  Calculate $\mathcal{J}_F$, the jacobian in Flux coordinates.
- interface calc_jac_h
  Calculate $\mathcal{J}_H$, the jacobian in HELENA coordinates.
- interface calc_jac_v
  Calculate $\mathcal{J}_V$, the jacobian in V(MEC) coordinates.
- interface calc_rzl
  Calculate R, Z & λ and derivatives in VMEC coordinates.
- interface calc_t_hf

Calculate $\overline{\overline{T}}^F_H$, the transformation matrix between H(ELENA) and F(lux) coordinate systems.

- interface calc_t_vc

  Calculate $\overline{\overline{T}}^V_C$, the transformation matrix between C(ylindrical) and V(mec) coordinate system.
- interface calc_t_vf

  Calculate $\overline{\overline{T}}^F_V$, the transformation matrix between V(MEC) and F(lux) coordinate systems.
- interface print_output_eq

  Print equilibrium quantities to an output file:
- interface redistribute_output_eq

  Redistribute the equilibrium variables, but only the Flux variables are saved.

Functions/Subroutines

- integer function create_vmec_input (grid_eq, eq_1)

  Creates a VMEC input file.
- integer function, public flux_q_plot (grid_eq, eq)

  Plots the flux quantities in the solution grid.
- integer function, public calc_derived_q (grid_eq, eq_1, eq_2)

  Calculates derived equilibrium quantities system.
- integer function, public calc_normalization_const ()

  Sets up normalization constants.
- subroutine, public normalize_input ()

  Normalize input quantities.
- integer function, public b_plot (grid_eq, eq_1, eq_2, rich_lvl, plot_fluxes, XYZ)

  Plots the magnetic fields.
- integer function, public j_plot (grid_eq, eq_1, eq_2, rich_lvl, plot_fluxes, XYZ)

  Plots the current.
- integer function, public kappa_plot (grid_eq, eq_1, eq_2, rich_lvl, XYZ)

  Plots the curvature.
- integer function, public delta_r_plot (grid_eq, eq_1, eq_2, XYZ, rich_lvl)

  *Plots **HALF** of the change in the position vectors for 2 different toroidal positions, which can correspond to a ripple.*
- integer function, public divide_eq_jobs (n_par_X, arr_size, n_div, n_div_max, n_par_X_base, range_↩ name)

  Divides the equilibrium jobs.
- integer function, public calc_eq_jobs_lims (n_par_X, n_div)

  Calculate `eq_jobs_lims`.
- integer function test_t_ef (grid_eq, eq_1, eq_2)

  See if `T_EF` it complies with the theory of [15].
- integer function test_d12h_h (grid_eq, eq)

  Tests whether $\frac{\partial^2}{\partial u_i \partial u_j} h_H$ is calculated correctly.
- integer function test_jac_f (grid_eq, eq_1, eq_2)

  Performs tests on $\mathcal{J}_F$.
- integer function test_g_v (grid_eq, eq)

  Tests whether g_V is calculated correctly.
- integer function test_jac_v (grid_eq, eq)

  Tests whether $\mathcal{J}_V$ is calculated correctly.
- integer function test_b_f (grid_eq, eq_1, eq_2)

  Tests whether $\vec{B}_F$ is calculated correctly.
- integer function test_p (grid_eq, eq_1, eq_2)

  Performs tests on pressure balance.

Variables

- logical, public debug_calc_derived_q = .false.
  plot debug information for calc_derived_q()
- logical, public debug_j_plot = .false.
  plot debug information for j_plot()
- logical, public debug_create_vmec_input = .false.
  plot debug information for create_vmec_input()

B.6.1 Detailed Description

Operations on the equilibrium variables.

B.6.2 Function/Subroutine Documentation

B.6.2.1 b_plot()

```
integer function, public eq_ops::b_plot (
                type(grid_type), intent(inout) grid_eq,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(in) eq_2,
                integer, intent(in), optional rich_lvl,
                logical, intent(in), optional plot_fluxes,
                real(dp), dimension(:,:,:,:), intent(in), optional XYZ )
```

Plots the magnetic fields.

If multiple equilibrium parallel jobs, every job does its piece, and the results are joined automatically by plot_HDF5.

The outputs are given in contra- and covariant components and magnitude in multiple coordinate systems, as indicated in calc_vec_comp().

The starting point is the fact that the magnetic field is given by

$$\vec{B} = \frac{\vec{e}_\theta}{\mathcal{J}},$$

in F coordinates. The F covariant components are therefore given by

$$B_i = \frac{g_{i3}}{\mathcal{J}} = \frac{\vec{e}_i \cdot \vec{e}_3}{\mathcal{J}},$$

and the only non-vanishing contravariant component is

$$B^3 = \frac{1}{\mathcal{J}}.$$

These are then all be transformed to the other coordinate systems.

Note

1. Vector plots for different Richardson levels can be combined to show the total grid by just plotting them all individually.
2. The metric factors and transformation matrices have to be allocated.

Returns

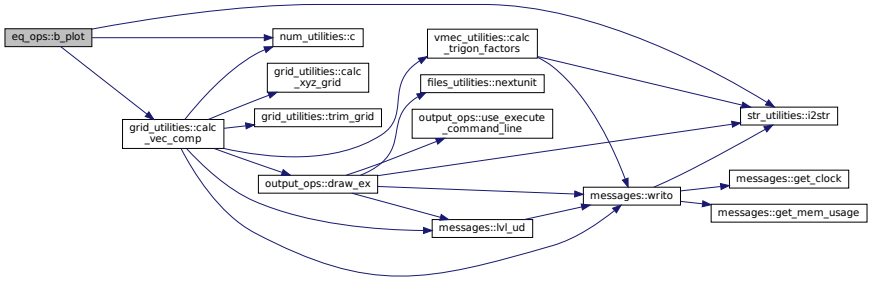
ierr

Parameters

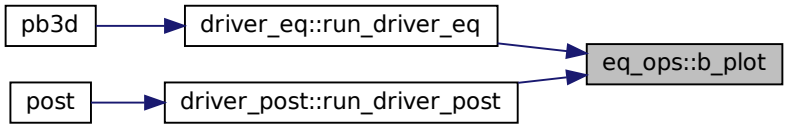
in,out	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables
in	*rich_lvl*	Richardson level
in	*plot_fluxes*	plot the fluxes
in	*xyz*	X, Y and Z of grid

Definition at line 5700 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.2.2 calc_derived_q()

```
integer function, public eq_ops::calc_derived_q (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout), target eq_2 )
```

Calculates derived equilibrium quantities system.

- magnetic shear s

- normal curvature `kappa_n`
- geodesic curvature `kappa_g`
- parallel current `sigma`

Naive implementations of these quantities, with the exception of `S`, give numerically very unfavorable results, so special care must be taken in this procedure.

This is an important issue as these derived equilbrium quantities are important building blocks of the perturbed potential energy, including the drivers of instabilities.

The general formulas are derived under the first section, VMEC. For axisymmetric HELENA, however, simplified equations are possible and these are presented afterwards.

B.6.3 VMEC

B.6.3.1 The shear

The local shear S is calculated using equation 3.22 from [15] :

$S = -\frac{1}{\mathcal{J}} \frac{\partial}{\partial \theta} \left(\frac{h^{\psi\alpha}}{h^{\psi\psi}} \right)$

which doesn't pose any particular problems as there are only angular derivatives.

B.6.3.2 The curvature

This is calculated using the parallel derivative of the parallel unit vector (i.e. theta if using poloidal flux and zeta if using toroidal flux).

By taking instead the derivative of the covariant basis vector and realizing that the difference with the real curvature lies solely in a component along the parallel direction, which cancels out when taking the normal or geodesic components, the situation becomes easier.

Asuming the poloidal flux is used as normal coordinate, the taks is therefore how to calculate $\frac{1}{|\vec{e}_\theta|^2} \frac{\partial}{\partial \theta} \vec{e}_\theta$.

By transforming both the derivative as well as the basis vector to Equilibrium coordinates, this can be written as

$\sum_{i=2,3} \sum_{j=2,3} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,i)) \frac{\partial}{\partial u_{\mathsf{E}}^i} \left(\mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,j) \, \vec{e}_{j,\mathsf{E}} \right)$

where the summations only run over the angular coordinates because $\vec{e}_\theta$ never has any component along $\vec{e}_{\psi,\mathsf{E}}$.

This double summation formula can then be written as a matrix equation

$$\begin{pmatrix} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,2) & \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,3) \end{pmatrix} \left[\begin{pmatrix} \frac{\partial}{\partial y^2} \vec{e}_2 & \frac{\partial}{\partial y^2} \vec{e}_3 \\ \frac{\partial}{\partial u^3} \vec{e}_2 & \frac{\partial}{\partial u^3} \vec{e}_3 \end{pmatrix}_{\mathsf{E}} \begin{pmatrix} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,2) \\ \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,3) \end{pmatrix} + \begin{pmatrix} \frac{\partial}{\partial y^2} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,2) & \frac{\partial}{\partial y^2} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,3) \\ \frac{\partial}{\partial u^3} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,2) & \frac{\partial}{\partial u^3} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,3) \end{pmatrix} \begin{pmatrix} \vec{e}_2 \\ \vec{e}_3 \end{pmatrix} \right]$$

which can be represented shorthand as $\mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,2:3) \left[\mathcal{D}\vec{e}_{\mathsf{E}} \mathcal{T}_{\mathsf{F}}^{\mathsf{E}} (3,2:3)^T + \mathcal{D}\mathcal{T}\vec{e}_{\mathsf{E}}^T \right]$

It is relatively easy to set up the matrix $\mathcal{D}\vec{e}_{\mathsf{E}}$ as a function of covariant basis vectors in the Cylindrical coordinate system. The derivatives of the transformation matrix itself can likewise be found.

The steps used in this routine are therefore

1. Set up $\mathcal{D}\vec{e}_{\mathsf{E}}$, i.e. as a function of the three cylindrical covariant basis vectors.
2. Set up correctcion by the derivatives of the transformation matrix, with a single summation.
3. Decompose the normal $\frac{\nabla\psi}{|\nabla\psi|^2}$ and geodesic vectors $\frac{\nabla\psi\times\vec{B}}{B^2}$ as a function of the cylindrical contravariant basis vectors.
4. Double summation to reduce term $\sim \mathcal{D}\vec{e}_{\mathsf{E}}$ and correction by single summation to reduce term $\sim \mathcal{DT}$.
5. Dot these for each of the four (actually three because of symmetry) elements of $\mathcal{D}\vec{e}_{\mathsf{E}}$.
6. Divide by $|\vec{e}_\theta|^2$.
7. Possibly correct for toroidal flux.

An advantage of using this formulation is that no normal derivatives are needed, so that nothing has to implicitely cancel out.

B.6.3.3 The parallel current

The parallel current is calculated from the shear with the help of equation 3.29 of [15] :

$$\mu_0\sigma = -\frac{1}{B_\theta}\left[2\frac{\nabla\psi\times\vec{B}}{|\nabla\psi|^2}\cdot\frac{\partial(\nabla\psi)}{\partial\theta} + \mathcal{J}\,|\nabla\psi|^2\,S\right].$$

where a similar technique can be used as above, for the calculation of the curvature: As

$$\frac{\nabla\psi\times\vec{B}}{|\nabla\psi|^2} = \frac{B_\theta\vec{e}_\alpha - B_\alpha\vec{e}_\theta}{\mathcal{J}|\nabla\psi|^2},$$

The parallel derivative of $\nabla\psi$ can be rewritten in terms of contravariant components of derivatives of covariant basis vectors:

$$\begin{cases}\vec{e}_\alpha\cdot\dfrac{\partial(\nabla\psi)}{\partial\theta} = -\nabla\psi\cdot\dfrac{\partial\vec{e}_\theta}{\partial\alpha}\\ \vec{e}_\theta\cdot\dfrac{\partial(\nabla\psi)}{\partial\theta} = -\nabla\psi\cdot\dfrac{\partial\vec{e}_\theta}{\partial\theta}\end{cases},$$

so that the result is:

$$\frac{\nabla\psi\times\vec{B}}{|\nabla\psi|^2}\cdot\frac{\partial(\nabla\psi)}{\partial\theta} = \frac{1}{\mathcal{J}^2}\frac{\nabla\psi}{|\nabla\psi|^2}\cdot\left[g_{\alpha\theta}\frac{\partial\vec{e}_\theta}{\partial\theta} - g_{\theta\theta}\frac{\partial\vec{e}_\theta}{\partial\alpha}\right],$$

which is given by a formula similar to the one used above for the geodesical curvature.

In debug mode, it can be checked whether the current is indeed divergence-free, with the help of equation 3.33 of [15].

$$-2p'\int_{\theta_0}^{\theta}\mathcal{J}\kappa_g\mathsf{d}\theta = \sigma(\theta) - \sigma_0$$

and whether a direct, naive implementation of the parallel current from equation 3.26 of [15] agrees with the more accurate results used here:

$$\mu_0\sigma = \frac{\partial B_\psi}{\partial\alpha} - \frac{\partial B_\alpha}{\partial\psi} - \mu_0 p'\mathcal{J}\frac{B_\alpha}{B_\theta}.$$

The reason why they are generally different is that this implementation relies on the cancellation of large terms.

B.6.4 HELENA

B.6.4.1 The parallel current

As $B_\alpha = F(\psi)$ and $\vec{e}_{\alpha,\mathsf{F}} = \vec{e}_{\phi,\mathsf{H}}$, the naive expression for the shear becomes simply

$\sigma = -F' - \mu_0 p' \frac{F}{B^2}$,

which can be used like that.

B.6.4.2 The shear

The calculate the local shear S is looks like it is best to use equation 3.22 from [15] , just as in the VMEC case:

As a test, however, equation 3.29 of [15] can be used in stead:

$\mathcal{J}|\nabla\psi|^2 S + \mu_0 \mathcal{J} B^2 \sigma = -2\frac{F}{R}\left(\frac{Z_\theta}{R} + \frac{Z_\theta R_{\theta\theta} - R_\theta Z_{\theta\theta}}{R_\theta^2 + Z_\theta^2}\right)$.

from which σ can be obtained.

B.6.4.3 The curvature

Also the curvature expressions can be simplified for axisymmetric situations. The result is given by

$\kappa_n = \frac{qR}{F}\frac{Z_\theta\left(R_{\theta\theta} - q^2 R\right) - R_\theta Z_{\theta\theta}}{\left(R_\theta^2 + Z_\theta^2 + (qR)^2\right)\left(R_\theta^2 + Z_\theta^2\right)}$

$\kappa_g = qR\frac{R_\theta\left(2\left(R_\theta^2 + Z_\theta^2\right) + (qR)^2\right) - R\left(R_\theta R_{\theta\theta} + Z_\theta Z_{\theta\theta}\right)}{\left(R_\theta^2 + Z_\theta^2 + (qR)^2\right)^2}$

Note

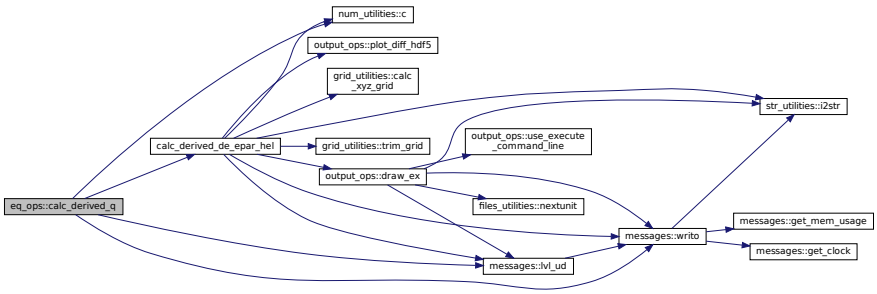
1. If the toroidal flux is used instead, the actual curvature obviously is unchanged, which implies that the normal component has to be multiplied by the safety factor and the geodesic component has to be divided by it.
2. The formulas for the normal and geodesic basis vectors for VMEC are
   - $\frac{\nabla\psi}{|\nabla\psi|^2} = -\frac{\mathcal{J}(1+\lambda_\theta)}{Z_\theta^2 + R_\theta^2 + (R_\zeta Z_\theta - R_\theta Z_\zeta)/R^2}\frac{1}{R}\left(-Z_\theta, R_\zeta Z_\theta - R_\theta Z_\zeta, R_\theta\right)_{\mathsf{C}}$
   - $\frac{\nabla\psi\times\vec{B}}{B^2} = \frac{1}{g_{\theta\theta}}\left(PR_\theta - QR_\zeta, -QR^2, PZ_\theta - QZ_\zeta\right)_{\mathsf{C}}$
   - $Q = g_{\theta\theta} - q g_{\theta\alpha}$ (using right-handed flux q_{F}, which is the inverse of the left-handed VMEC q_{V})
   - $P = \frac{Q\lambda_\zeta - g_{\alpha\theta}}{1+\lambda_\theta}$ (where the derivatives of R, Z and λ are in VMEC coordinates).
3. The formulas for the normal and geodesic basis vectors for HELENA are
   - $\frac{\nabla\psi}{|\nabla\psi|^2} = -\frac{1}{Z_\theta^2 + R_\theta^2}\frac{qR}{F}\left(-Z_\theta, 0, R_\theta\right)_{\mathsf{C}}$
   - $\frac{\nabla\psi\times\vec{B}}{B^2} = \frac{qR^2}{R_\theta^2 + Z_\theta^2 + q^2R^2}\left(-R_\theta, -\frac{R_\theta^2 + Z_\theta^2}{q}, -Z_\theta\right)_{\mathsf{C}}$

Parameters

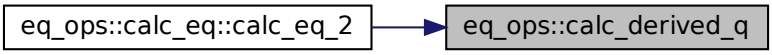
in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in,out	*eq_2*	metric equilibrium variables

Definition at line 4287 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.4 calc_eq_jobs_lims()

```
integer function, public eq_ops::calc_eq_jobs_lims (
              integer, intent(in) n_par_X,
              integer, intent(in) n_div )
```

Calculate `eq_jobs_lims`.

Take into account that every job has to start and end at the start and end of a fundamental integration interval, as discussed in divide_eq_jobs():

- for `magn_int_style=1` (trapezoidal), this is 1 point,
- for `magn_int_style=2` (Simpson 3/8), this is 3 points.

for POST, there are no Richardson levels, and there has to be overlap of one always, in order to have correct composite integrals of the different regions.

Note

The `n_par_X` passed into this procedure refers to the quantity that is already possibly halved if the Richardson level is higher than 1. This information is then reflected in the eq_jobs_lims, which refer to the local limits, i.e. only the parallel points currently under consideration.

Returns

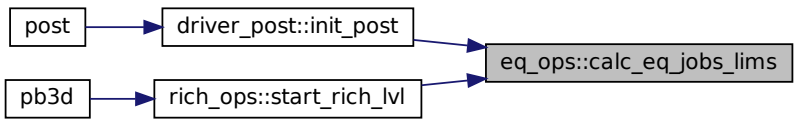
ierr

Parameters

`in`	*n_↩ par_x*	number of parallel points in this Richardson level
`in`	*n_div*	nr. of divisions of parallel ranges

Definition at line 6701 of file eq_ops.f90.

Here is the caller graph for this function:



B.6.4.5 calc_normalization_const()

```
integer function, public eq_ops::calc_normalization_const
```

Sets up normalization constants.

- VMEC version (`eq_style=1`):
  Normalization depends on `norm_style`:
  1. MISHKA
     - R_0: major radius (= average magnetic axis)
     - B_0: B on magnetic axis (theta = zeta = 0)
     - pres_0: reference pressure (= B_0^2/mu_0)
     - psi_0: reference flux (= R_0^2 B_0)
     - rho_0: reference mass density
  2. COBRA
     - R_0: major radius (= average geometric axis)

  - pres_0: pressure on magnetic axis
  - B_0: reference magnetic field (= sqrt(2pres_0mu_0 / beta))
  - psi_0: reference flux (= R_0^2 B_0 / aspr^2)
  - rho_0: reference mass density where aspr (aspect ratio) and beta are given by VMEC.

- HELENA version (`eq_style=2`):
  MISHKA Normalization is used by default and does not depend on `norm_style`

See also

read_hel()

Note

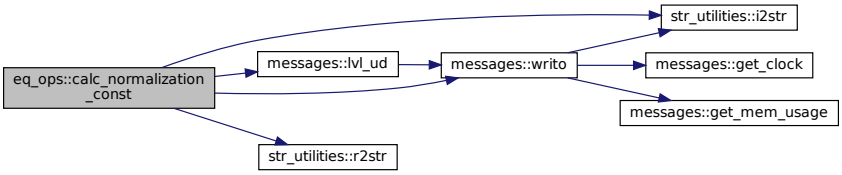
`rho_0` is not given through by the equilibrium codes and should be user-supplied

Returns

ierr

Definition at line 5405 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.6.4.6 create_vmec_input()

```
integer function eq_ops::create_vmec_input (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1 )
```

Creates a VMEC input file.

Optionally, a perturbation can be added: Either the displacement of the plasma position can be described (`pert_style` 1), or ripple in the toroidal magnetic field (`pert_style` 2), with a fixed toroidal mode number.

Both perturbation styles can have various prescription types:

1. file with Fourier modes in the geometrical angular coordinate
2. same but manually
3. file with perturbation from a 2-D map in the geometric angular coordinate.

For `pert_style` 2, a file has to be provided that describes the translation between position perturbation and magnetic perturbation for curves of constant geometrical angle. This file can be generated for an already existing ripple case using POST with `--compare_tor_pos` with `n_zeta_plot = 3` and `min_theta_plot` and `max_theta_plot` indicating half a ripple period.

The output from this VMEC run can then be used to iteratively create a new file to translate toroidal magnetic field ripple to position perturbation.

Note

Meaning of the indices of `B_F`, `B_F_dum`:

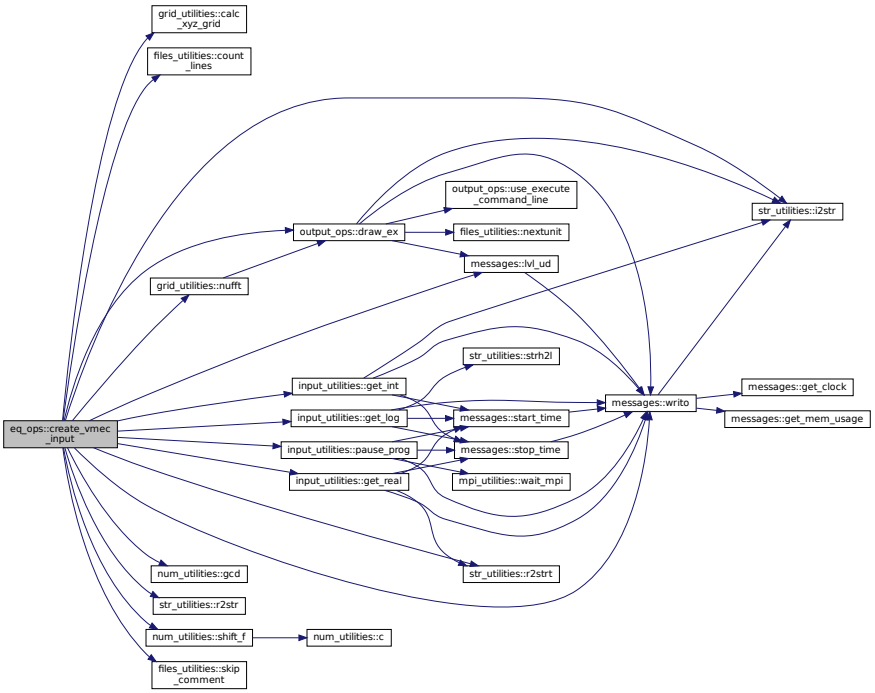
- `(pol modes, cos/sin)` for `B_F_dum`
- `(tor_modes, pol modes, cos/sin (m theta), R/Z)` for `B_F`

Parameters

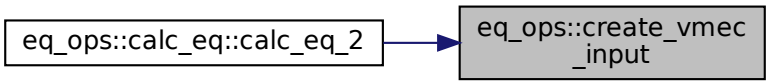
in	*grid_eq*	equilibrium grid varibles
in	*eq_1*	flux equilibrium quantities

Definition at line 784 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.7 delta_r_plot()

```
integer function, public eq_ops::delta_r_plot (
              type(grid_type), intent(inout) grid_eq,
              type(eq_1_type), intent(in) eq_1,
              type(eq_2_type), intent(in) eq_2,
              real(dp), dimension(:,:,:,:), intent(in) XYZ,
              integer, intent(in), optional rich_lvl )
```

Plots **HALF** of the change in the position vectors for 2 different toroidal positions, which can correspond to a ripple.

Also calculates **HALF** of the relative magnetic perturbation, which also corresponds to a ripple.

Finally, if the output grid contains a fundamental interval 2π, the proportionality between both is written to a file.

Note

The metric factors and transformation matrices have to be allocated.

Returns

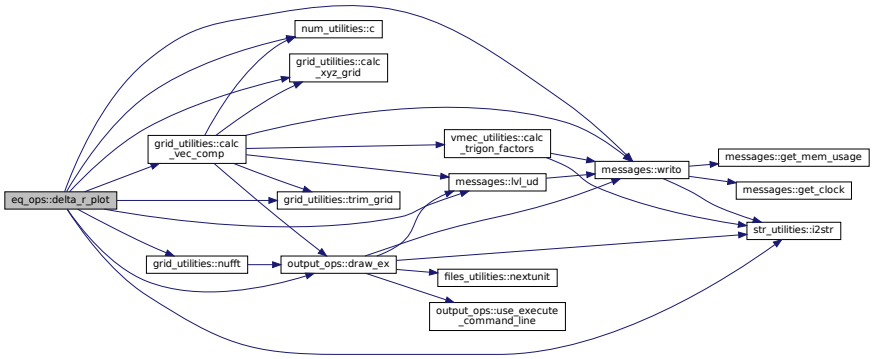
ierr

Parameters

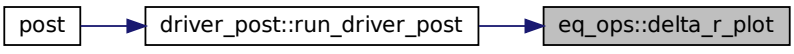
in,out	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables
in	*xyz*	X, Y and Z of grid
in	*rich_lvl*	Richardson level

Definition at line 6173 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.8 divide_eq_jobs()

```
integer function, public eq_ops::divide_eq_jobs (
               integer, intent(in) n_par_X,
               integer, dimension(2), intent(in) arr_size,
               integer, intent(inout) n_div,
               integer, intent(in), optional n_div_max,
               integer, intent(in), optional n_par_X_base,
               character(len=*), intent(in), optional range_name )
```

Divides the equilibrium jobs.

For PB3D, the entire parallel range has to be calculated, but due to memory limits this has to be split up in pieces. Every piece has to be able to contain the equilibrium variables (see note below), as well as the vectorial perturbation variables. These are later combined into tensorial variables and integrated.

The equilibrium variables have to be operated on to calculate them, which translates to a scale factor `mem_scale_fac`. However, in the perturbation phase, when they are just used, this scale factor is not needed.

In its most extreme form, the division in equilibrium jobs would be the individual calculation on a fundamental integration integral of the parallel points:

- for `magn_int_style=1` (trapezoidal), this is 1 point,
- for `magn_int_style=2` (Simpson 3/8), this is 3 points.

For HELENA, the parallel derivatives are calculated discretely, the equilibrium and vectorial perturbation variables are tabulated first in this HELENA grid. This happens in the first Richardson level. In all Richardson levels, afterwards, these variables are interpolated in the angular directions. In this case, therefore, there can be no division of this HELENA output interval for the first Richardson level.

This procedure does the job of dividing the grids setting the global variables `eq_jobs_lims`.

The integration of the tensorial perturbation variables is adjusted:

- If the first job of the parallel jobs and not the first Richardson level: add half of the integrated tensorial perturbation quantities of the previous level.
- If not the first job of the parallel jobs, add the integrated tensorial perturbation quantities to those of the previous parallel job, same Richardson level.

In fact, the equilibrium jobs have much in common with the Richardson levels, as is attested by the existence of the routines do_eq() and eq_info(), which are equivalent to do_rich() and rich_info().

In POST, finally, the situation is slightly different for HELENA, as all the requested variables have to fit, including the interpolated variables, as they are stored whereas in PB3D they are not. The parallel range to be taken is then the one of the output grid, including a base range for the variables tabulated on the HELENA grid. Also, for extended output grids, the size of the grid in the secondary angle has to be included in `n_par_X` (i.e. toroidal when poloidal flux is used and vice versa). Furthermore, multiple equilibrium jobs are allowed.

To this end, optionally, a base number can be provided for `n_par_X`, that is always added to the number of points in the divided `n_par_X`.

Note

For PB3D, only the variables `g_FD`, `h_FD` and `jac_FD` are counted, as the equilibrium variables and the transformation matrices are deleted after use. Also, `S`, `sigma`, `kappa_n` and `kappa_g` can be neglected as they do not contain derivatives and are therefore much smaller. in both routines calc_memory_eq() and calc_memory_x(), however, a 50% safety factor is used to account for this somewhat.

Returns

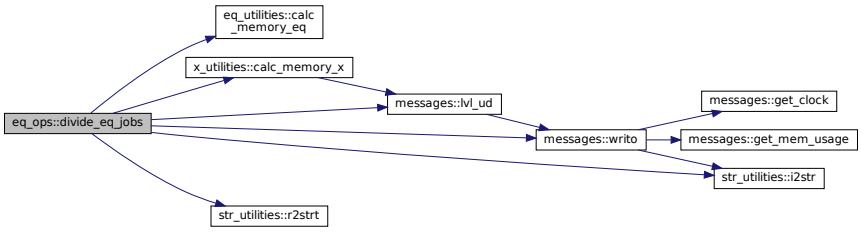
ierr

Parameters

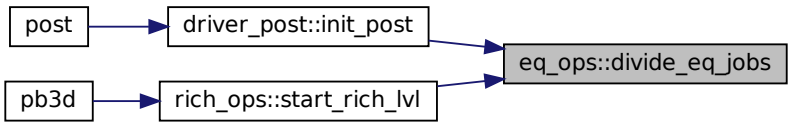
in	*n_par_x*	number of parallel points to be divided
in	*arr_size*	array size (using loc_n_r) for eq_2 and X_1 variables
in,out	*n_div*	final number of divisions
in	*n_div_max*	maximum n_div
in	*n_par_x_base*	base n_par_X, undivisible
in	*range_name*	name of range

Definition at line 6566 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.9 flux_q_plot()

```
integer function, public eq_ops::flux_q_plot (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq )
```

Plots the flux quantities in the solution grid.

- safety factor `q_saf`
- rotational transform `rot_t`
- pressure `pres`
- poloidal flux `flux_p`
- toroidal flux `flux_t`

Returns

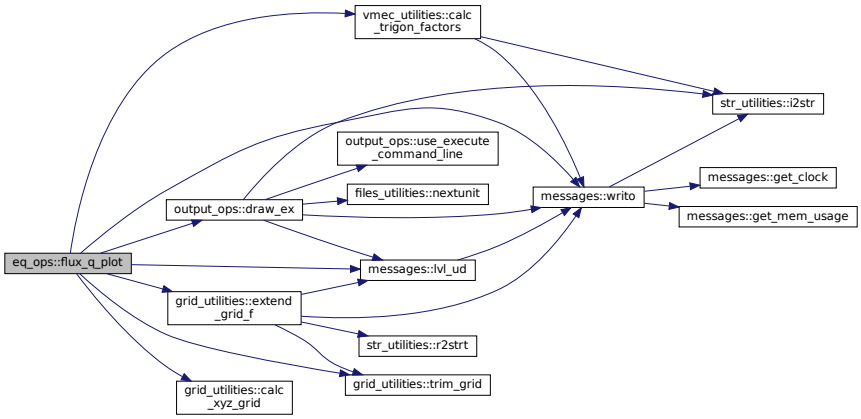
ierr

Parameters

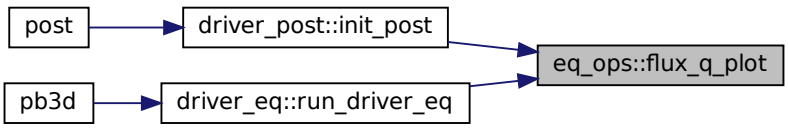
in	*grid_eq*	normal grid
in	*eq*	flux equilibrium variables

Definition at line 3810 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.10 j_plot()

```
integer function, public eq_ops::j_plot (
               type(grid_type), intent(inout) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               integer, intent(in), optional rich_lvl,
```

```
logical, intent(in), optional plot_fluxes,
real(dp), dimension(:,:,:,:), intent(in), optional XYZ )
```

Plots the current.

If multiple equilibrium parallel jobs, every job does its piece, and the results are joined automatically by plot_HDF5.

The outputs are given in contra- and covariant components and magnitude in multiple coordinate systems, as indicated in calc_vec_comp().

The starting point is the pressure balance

$$\nabla p = \vec{J} \times \vec{B},$$

which, using $\vec{B} = \frac{\vec{e}_\theta}{\mathcal{J}}$, reduces to

$$J^\alpha = -p'.$$

Furthermore, the current has to lie in the magnetic flux surfaces:

$$J^\psi = 0.$$

Finally, the parallel current σ gives an expression for the last contravariant component:

$$J^\theta = \frac{\sigma}{\mathcal{J}} + p' \frac{B_\alpha}{B_\theta}.$$

From these, the contravariant components can be calculated as

$$J_i = J^\alpha g_{\alpha,i} + J^\theta g_{\theta,i}.$$

These are then all be transformed to the other coordinate systems.

Note

1. Vector plots for different Richardson levels can be combined to show the total grid by just plotting them all individually.
2. The metric factors and transformation matrices have to be allocated.

Returns

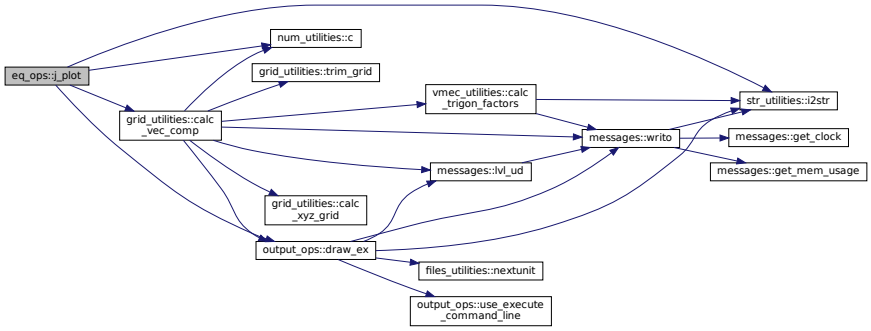
ierr

Parameters

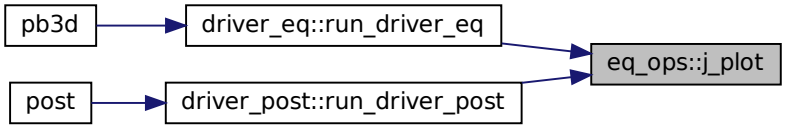
in,out	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables
in	*rich_lvl*	Richardson level
in	*plot_fluxes*	plot the fluxes
in	*xyz*	X, Y and Z of grid

Definition at line 5803 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.11 kappa_plot()

```
integer function, public eq_ops::kappa_plot (
               type(grid_type), intent(inout) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               integer, intent(in), optional rich_lvl,
               real(dp), dimension(:,:,:,:), intent(in), optional XYZ )
```

Plots the curvature.

If multiple equilibrium parallel jobs, every job does its piece, and the results are joined automatically by plot_HDF5.

The outputs are given in contra- and covariant components and magnitude in multiple coordinate systems, as indicated in calc_vec_comp().

The starting point is the curvature, given by

$$\vec{\kappa} = \kappa_n \frac{\nabla\psi}{|\nabla\psi|^2} + \kappa_g \frac{\nabla\psi \times \vec{B}}{B^2},$$

which can be used to find the covariant and contravariant components in Flux coordinates.

These are then transformed to Cartesian coordinates and plotted.

Note

1. Vector plots for different Richardson levels can be combined to show the total grid by just plotting them all individually.
2. The metric factors and transformation matrices have to be allocated.

Returns

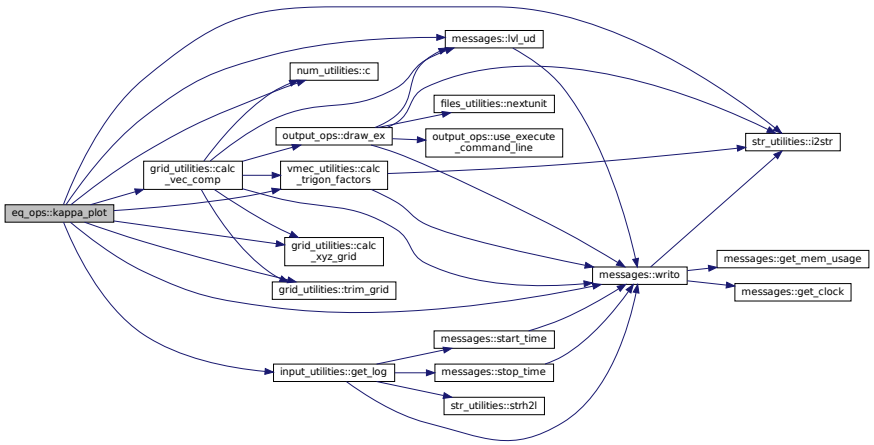
ierr

Parameters

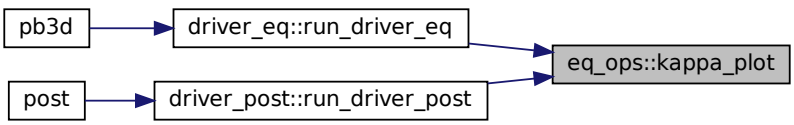
in,out	*grid_eq*	equilibrium grid
in	*eq_1*	metric equilibrium variables
in	*eq_2*	metric equilibrium variables
in	*rich_lvl*	Richardson level
in	*xyz*	X, Y and Z of grid

Definition at line 5971 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.12 normalize_input()

```
subroutine, public eq_ops::normalize_input
```

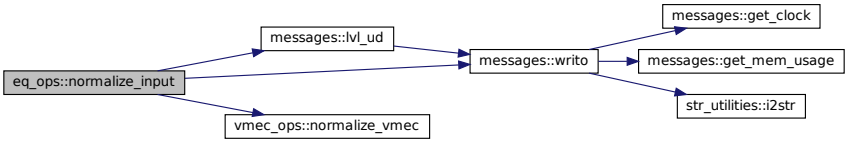
Normalize input quantities.

See also

calc_normalization_const()

Definition at line 5643 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.6.4.13 test_b_f()

```
integer function eq_ops::test_b_f (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2 )
```

Tests whether $\vec{B}_\mathrm{F}$ is calculated correctly.

Note

Debug version only

Returns

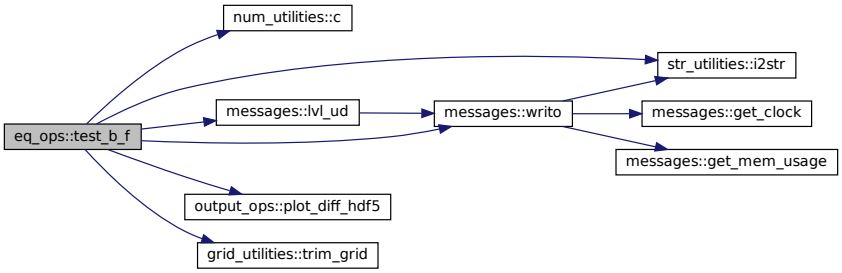
ierr

Parameters

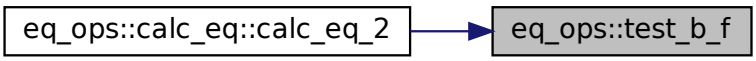
in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium

Definition at line 7344 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.14 test_d12h_h()

```
integer function eq_ops::test_d12h_h (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(in) eq )
```

Tests whether $\frac{\partial^2}{\partial u_i \partial u_j} h_{\mathrm{H}}$ is calculated correctly.

Note

Debug version only

Returns

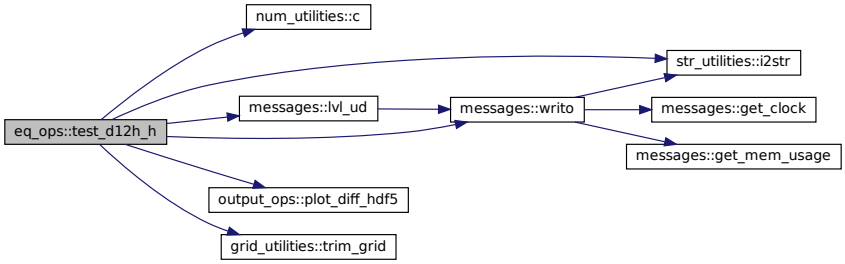
ierr

Parameters

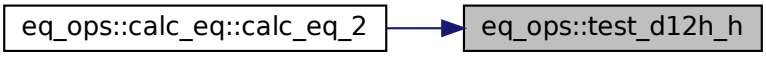
in	*grid_eq*	equilibrium grid
in	*eq*	metric equilibrium

Definition at line 6953 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.15 test_g_v()

```
integer function eq_ops::test_g_v (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(in) eq )
```

Tests whether g_V is calculated correctly.

Note

Debug version only

Returns

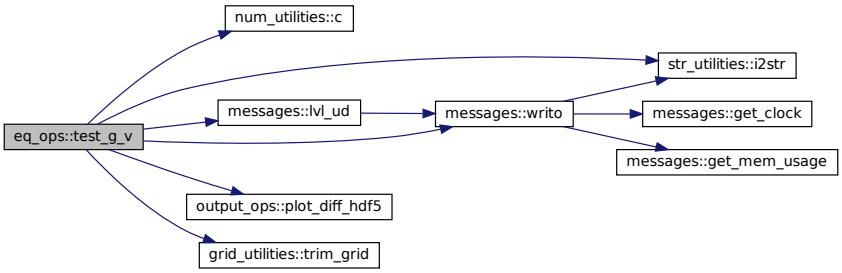
ierr

Parameters

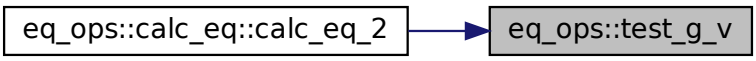
in	*grid_eq*	equilibrium grid
in	*eq*	metric equilibrium

Definition at line 7181 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.16 test_jac_f()

```
integer function eq_ops::test_jac_f (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in), target eq_1,
               type(eq_2_type), intent(in) eq_2 )
```

Performs tests on $\mathcal{J}_F$.

- comparing it with the determinant of g_F
- comparing it with the direct formula

Note

Debug version only

Returns

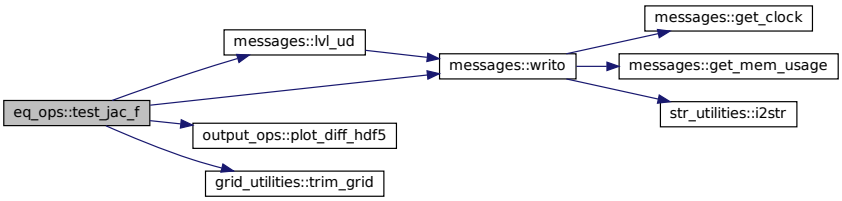
ierr

Parameters

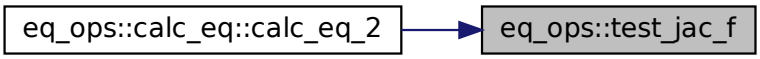
in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium

Definition at line 7064 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.17 test_jac_v()

```
integer function eq_ops::test_jac_v (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(in) eq )
```

Tests whether $\mathcal{J}_V$ is calculated correctly.

Note

Debug version only

Returns

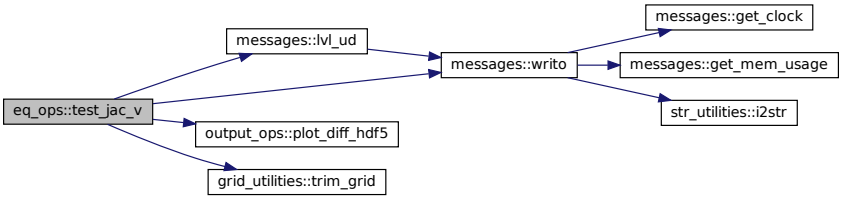
ierr

Parameters

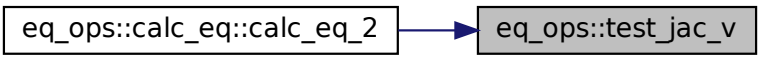
in	*grid_eq*	equilibrium grid
in	*eq*	metric equilibrium

Definition at line 7280 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.18 test_p()

```
integer function eq_ops::test_p (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2 )
```

Performs tests on pressure balance.

$$\mu_0 \frac{\partial p}{\partial u^2} = \frac{1}{\mathcal{J}} \left(\frac{\partial B_2}{\partial u^3} - \frac{\partial B_3}{\partial u^2} \right)$$
$$\mu_0 \mathcal{J} \frac{\partial p}{\partial u^3} = 0 \rightarrow \left(\frac{\partial B_1}{\partial u^3} = \frac{\partial B_3}{\partial u^1} \right),$$

working in the (modified) Flux coordinates $(\alpha, \psi, \theta)_{\mathrm{F}}$

Note

Debug version only

Returns

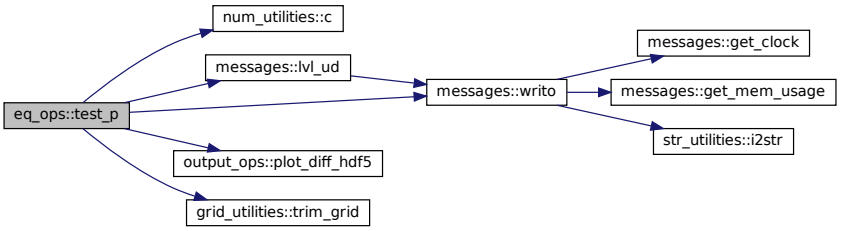
ierr

Parameters

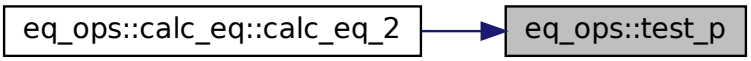
in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables

Definition at line 7502 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.4.19 test_t_ef()

```
integer function eq_ops::test_t_ef (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2 )
```

See if T_EF it complies with the theory of [15].

Note

Debug version only

Returns

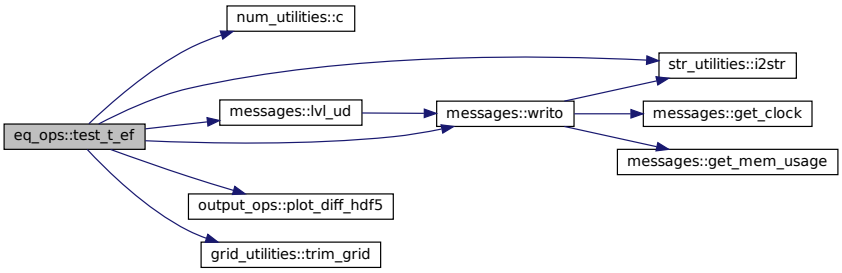
ierr

Parameters

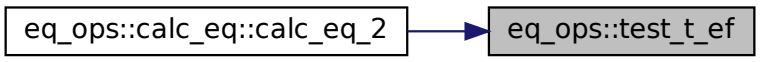
in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium

Definition at line 6779 of file eq_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.6.5 Variable Documentation

B.6.5.1 debug_calc_derived_q

```
logical, public eq_ops::debug_calc_derived_q = .false.
```

plot debug information for calc_derived_q()

Note

Debug version only

Definition at line 30 of file eq_ops.f90.

B.6.5.2 debug_create_vmec_input

```
logical, public eq_ops::debug_create_vmec_input = .false.
```

plot debug information for create_vmec_input()

Note

Debug version only

Definition at line 34 of file eq_ops.f90.

B.6.5.3 debug_j_plot

```
logical, public eq_ops::debug_j_plot = .false.
```

plot debug information for j_plot()

Note

Debug version only

Definition at line 32 of file eq_ops.f90.

B.7 eq_utilities Module Reference

Numerical utilities related to equilibrium variables.

Interfaces and Types

- interface calc_f_derivs

  Transforms derivatives of the equilibrium quantities in E coordinates to derivatives in the F coordinates.
- interface calc_inv_met

  Calculate $D_1^{m_1} D_2^{m_2} D_3^{m_3} X$ *from* $D_1^{i_1} D_2^{i_2} D_3^{i_3} X$ *and* $D_1^{j_1} D_2^{j_2} D_3^{j_3} Y$ *where* $XY = 1$ *and* $i, j = 0 \ldots m$*, according to [15].*
- interface transf_deriv

  Calculates derivatives in a coordinate system B from derivatives in a coordinates system A, making use of the transformation matrix $\overline{T}_B^A$*.*

Functions/Subroutines

- integer function, public calc_g (g_A, T_BA, g_B, deriv, max_deriv)

  Calculate the metric coefficients in a coordinate system B ! using the.
- integer function, public calc_memory_eq (arr_size, n_par, mem_size)

  Calculate memory in MB necessary for variables in equilibrium job.
- logical function, public do_eq ()

  If this equilibrium job should be done, also increment `eq_job_nr`.
- elemental character(len=max_str_ln) function, public eq_info ()

  Returns string with possible extension with equilibrium job as well as parallel job, or nothing if only one level and one parallel job.
- subroutine, public print_info_eq (n_par_X_rich)

  Prints information for equilibrium parallel job.

Variables

- logical, public debug_calc_inv_met_ind = .false.

  plot debug information for calc_inv_met_ind()

B.7.1 Detailed Description

Numerical utilities related to equilibrium variables.

B.7.2 Function/Subroutine Documentation

B.7.2.1 calc_g()

```
integer function, public eq_utilities::calc_g (
               real(dp), dimension(:,:,:,0:,0:,0:), intent(in) g_A,
               real(dp), dimension(:,:,:,0:,0:,0:), intent(in) T_BA,
               real(dp), dimension(:,:,:,0:,0:,0:), intent(inout) g_B,
               integer, dimension(3), intent(in) deriv,
               integer, intent(in) max_deriv )
```

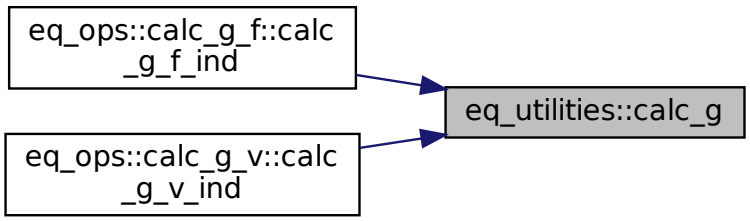
Calculate the metric coefficients in a coordinate system B ! using the.

Parameters

in	*g_a*	g_A
in	*t_ba*	$\overline{T}^A_B$
in,out	*g_b*	g_B
in	*deriv*	derivatives
in	*max_deriv*	maximum derivative

Definition at line 387 of file eq_utilities.f90.

Here is the caller graph for this function:



B.7.2.2 calc_memory_eq()

```
integer function, public eq_utilities::calc_memory_eq (
               integer, intent(in) arr_size,
               integer, intent(in) n_par,
               real(dp), intent(inout) mem_size )
```

Calculate memory in MB necessary for variables in equilibrium job.

The size of these variables is equal to the product of the non-parallel dimensions (e.g. `n_geo` x `loc_n_r`), times the number of variables.

The latter should be:

- PB3D: only take into account ($2 \cdot 6 + 1 = 13$) equilibrium variables `g_FD`, `h_FD` and `jac_FD`, as the perturbation variables are divided in jobs occupying the remaning space. These equilibrium variables are tabulated on the equilibrium grid. Note that they contain derivatives in extra dimensions, so that their size should be multiplied by (`max_deriv+1`)^3.
- POST: take into account these 13 equilibrium variables, as well as 4 variables `U` and `DU`, with double size due to being complex, and the additional dimension equal to `n_mod_X`, but without derivatives.

Returns

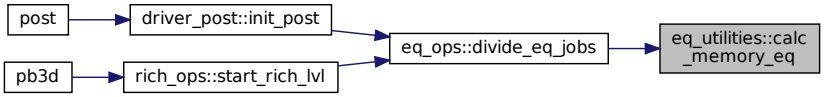
ierr

Parameters

`in`	*arr_size*	size of part of X array
`in`	*n_par*	number of parallel points
`in,out`	*mem_size*	total size

Definition at line 907 of file eq_utilities.f90.

Here is the caller graph for this function:



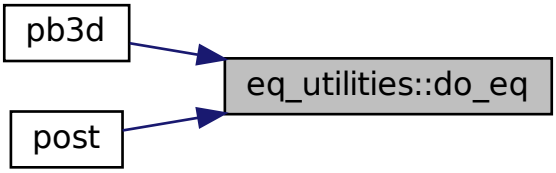
B.7.2.3 do_eq()

`logical function, public eq_utilities::do_eq`

If this equilibrium job should be done, also increment `eq_job_nr`.

Definition at line 949 of file eq_utilities.f90.

Here is the caller graph for this function:



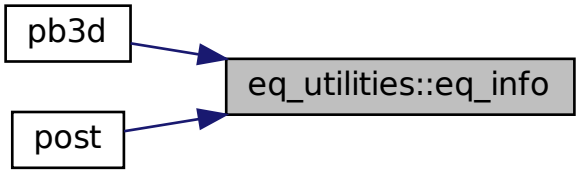
B.7.2.4 eq_info()

`elemental character(len=max_str_ln) function, public eq_utilities::eq_info`

Returns string with possible extension with equilibrium job as well as parallel job, or nothing if only one level and one parallel job.

Definition at line 974 of file eq_utilities.f90.

Here is the caller graph for this function:



B.7.2.5 print_info_eq()

```
subroutine, public eq_utilities::print_info_eq (
               integer, intent(in) n_par_X_rich )
```

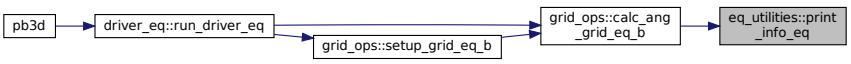
Prints information for equilibrium parallel job.

Parameters

in	*n_par_x_rich*	number of parallel points in this Richardson level

Definition at line 986 of file eq_utilities.f90.

Here is the caller graph for this function:



B.7.3 Variable Documentation

B.7.3.1 debug_calc_inv_met_ind

```
logical, public eq_utilities::debug_calc_inv_met_ind = .false.
```

plot debug information for calc_inv_met_ind()

Note

Debug version only

Definition at line 27 of file eq_utilities.f90.

B.8 eq_vars Module Reference

Variables that have to do with equilibrium quantities and the grid used in the calculations:

Interfaces and Types

- type eq_1_type
  flux equilibrium type
- type eq_2_type
  metric equilibrium type

Functions/Subroutines

- subroutine init_eq_1 (eq, grid, setup_E, setup_F)
  Initializes new flux equilibrium.
- subroutine init_eq_2 (eq, grid, setup_E, setup_F)
  Initializes new metric equilibrium.
- subroutine copy_eq_1 (eq_i, grid_i, eq_o)
  Deep copy of flux equilibrium variables.
- subroutine copy_eq_2 (eq_i, grid_i, eq_o)
  Deep copy of metric equilibrium variables.
- subroutine dealloc_eq_1 (eq)
  Deallocates flux equilibrium quantities.
- subroutine dealloc_eq_2 (eq)
  Deallocates metric equilibrium quantities.

Variables

- real(dp), public r_0
  independent normalization constant for nondimensionalization
- real(dp), public pres_0
  independent normalization constant for nondimensionalization
- real(dp), public rho_0
  independent normalization constant for nondimensionalization
- real(dp), public b_0
  derived normalization constant for nondimensionalization
- real(dp), public psi_0
  derived normalization constant for nondimensionalization
- real(dp), public t_0
  derived normalization constant for nondimensionalization
- real(dp), public vac_perm = mu_0_original
  either usual mu_0 (default) or normalized
- real(dp), public max_flux_e
  max. flux in Equilibrium coordinates, set in calc_norm_range_PB3D_in
- real(dp), public max_flux_f
  max. flux in Flux coordinates, set in calc_norm_range_PB3D_in
- integer, public n_alloc_eq_1s
  nr. of allocated eq_1 variables
- integer, public n_alloc_eq_2s
  nr. of allocated eq_2 variables

B.8.1 Detailed Description

Variables that have to do with equilibrium quantities and the grid used in the calculations:

- The equilibrium variables are comprised of the variables that result from the equilibrium calculation, such as pressure, rotational transform, etc.
- The flux variables are tabulated on a 1D grid.
- The metric variables are tabulated on a 3D grid
  - In the normal coordinate, they are tabulated in the equilibrium grid.
  - In the angular coordinates, they are tabulated in the solution grid (VMEC), or in the equilibrium grid followed by an adaptation to the solution grid (HELENA).
- However, this is not necessarily always the case, as it depends on the grid angles being aligned with the grid (e.g. Providing theta and zeta in the equilibrium grid so that the magnetic field lines are followed.

Note

In general in PB3D, there are two kinds of variables, differing from one another in the way in which they are tabulated:

- variables tabulated on the full output grid of the equilibrium code
- variables tabulated in an internal grid of this code

In many places in the code a range in the normal coordinate is selected for each of the variables on different processes. This selection has to be done correctly and things can get a little bit complicated if trimmed grids are used (grids that have no overlap between processes).

See also

grid_ops()

B.8.2 Function/Subroutine Documentation

B.8.2.1 copy_eq_1()

```
subroutine eq_vars::copy_eq_1 (
               class(eq_1_type), intent(in) eq_i,
               type(grid_type), intent(in) grid_i,
               type(eq_1_type), intent(inout) eq_o )
```

Deep copy of flux equilibrium variables.

Parameters

in	*eq_i*	eq_1 to be copied
in	*grid_i*	grid of eq_i
in,out	*eq_o*	copied eq_1

Definition at line 439 of file eq_vars.f90.

B.8.2.2 copy_eq_2()

```
subroutine eq_vars::copy_eq_2 (
               class(eq_2_type), intent(in) eq_i,
               type(grid_type), intent(in) grid_i,
               type(eq_2_type), intent(inout) eq_o )
```

Deep copy of metric equilibrium variables.

Returns

ierr

Parameters

in	*eq_i*	eq_2 to be copied
in	*grid_i*	grid of eq_i
in,out	*eq_o*	copied eq_1

Definition at line 477 of file eq_vars.f90.

B.8.2.3 dealloc_eq_1()

```
subroutine eq_vars::dealloc_eq_1 (
               class(eq_1_type), intent(inout) eq )
```

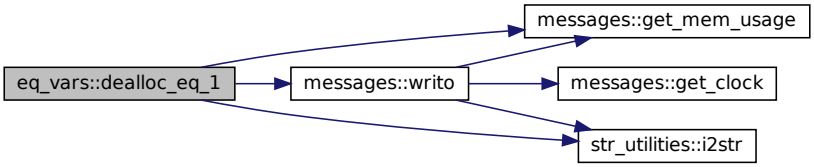
Deallocates flux equilibrium quantities.

Parameters

in,out	*eq*	equilibrium to be deallocated

Definition at line 531 of file eq_vars.f90.

Here is the call graph for this function:



B.8.2.4 dealloc_eq_2()

```
subroutine eq_vars::dealloc_eq_2 (
               class(eq_2_type), intent(inout) eq )
```

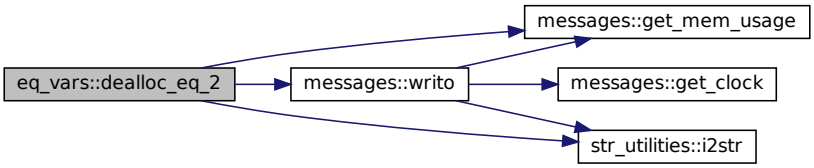
Deallocates metric equilibrium quantities.

Parameters

in,out	*eq*	equilibrium to be deallocated

Definition at line 578 of file eq_vars.f90.

Here is the call graph for this function:



B.8.2.5 init_eq_1()

```
subroutine eq_vars::init_eq_1 (
               class(eq_1_type), intent(inout) eq,
               type(grid_type), intent(in) grid,
               logical, intent(in), optional setup_E,
               logical, intent(in), optional setup_F )
```

Initializes new flux equilibrium.

The normal and angular grid can be in any coord. system, as only the grid sizes are used, not the coordinate values.

Optionally, it can be chosen individually whether the E or F(D) quantities are allocated. D means that the derivatives are in F as well. The rationale behind this is that the E quantities are only used in the pre-perturbation phase.

Note

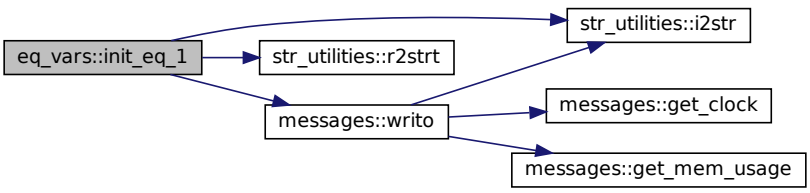
1. The E quantities are not written in eq_ops.print_output_eq().
2. The quantities that do not have a derivative are considered F quantities. Alternatively, all quantities that have only one version, are considered F quantities, such as rho, kappa_n, ...
3. The maximum derivative degree for flux quantities is one higher than max_deriv, because the derivative of some of them appear in the transformation matrices.

Parameters

in,out	*eq*	equilibrium to be initialized
in	*grid*	equilibrium grid
in	*setup_e*	whether to set up E
in	*setup_f*	whether to set up F

Definition at line 174 of file eq_vars.f90.

Here is the call graph for this function:



B.8.2.6 init_eq_2()

```
subroutine eq_vars::init_eq_2 (
               class(eq_2_type), intent(inout) eq,
               type(grid_type), intent(in) grid,
               logical, intent(in), optional setup_E,
               logical, intent(in), optional setup_F )
```

Initializes new metric equilibrium.

See also

For explanation see init_eq_1().

Note

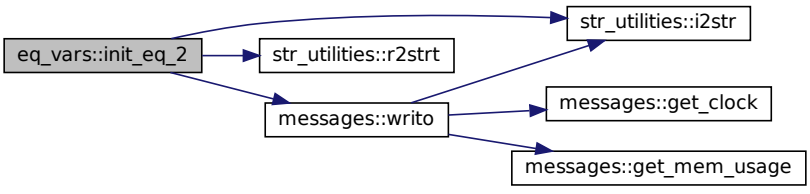
The maximum derivative degree for R, Z and lambda is one higher than `max_deriv`, because their first derivative already appears in g_C and h_C.

Parameters

`in,out`	*eq*	equilibrium to be initialized
`in`	*grid*	equilibrium grid
`in`	*setup_e*	whether to set up E
`in`	*setup_f*	whether to set up F

Definition at line 273 of file eq_vars.f90.

Here is the call graph for this function:



B.8.3 Variable Documentation

B.8.3.1 b_0

```
real(dp), public eq_vars::b_0
```

derived normalization constant for nondimensionalization

Definition at line 45 of file eq_vars.f90.

B.8.3.2 max_flux_e

```
real(dp), public eq_vars::max_flux_e
```

max. flux in Equilibrium coordinates, set in calc_norm_range_PB3D_in

Definition at line 49 of file eq_vars.f90.

B.8.3.3 max_flux_f

```
real(dp), public eq_vars::max_flux_f
```

max. flux in Flux coordinates, set in calc_norm_range_PB3D_in

Definition at line 50 of file eq_vars.f90.

B.8.3.4 n_alloc_eq_1s

```
integer, public eq_vars::n_alloc_eq_1s
```

nr. of allocated eq_1 variables

Note

Debug version only

Definition at line 53 of file eq_vars.f90.

B.8.3.5 n_alloc_eq_2s

```
integer, public eq_vars::n_alloc_eq_2s
```

nr. of allocated eq_2 variables

Note

Debug version only

Definition at line 55 of file eq_vars.f90.

B.8.3.6 pres_0

```
real(dp), public eq_vars::pres_0
```

independent normalization constant for nondimensionalization

Definition at line 43 of file eq_vars.f90.

B.8.3.7 psi_0

```
real(dp), public eq_vars::psi_0
```

derived normalization constant for nondimensionalization

Definition at line 46 of file eq_vars.f90.

B.8.3.8 r_0

```
real(dp), public eq_vars::r_0
```

independent normalization constant for nondimensionalization

Definition at line 42 of file eq_vars.f90.

B.8.3.9 rho_0

```
real(dp), public eq_vars::rho_0
```

independent normalization constant for nondimensionalization

Definition at line 44 of file eq_vars.f90.

B.8.3.10 t_0

```
real(dp), public eq_vars::t_0
```

derived normalization constant for nondimensionalization

Definition at line 47 of file eq_vars.f90.

B.8.3.11 vac_perm

```
real(dp), public eq_vars::vac_perm = mu_0_original
```

either usual mu_0 (default) or normalized

Definition at line 48 of file eq_vars.f90.

B.9 files_ops Module Reference

Operations related to files !

Functions/Subroutines

- subroutine, public init_files ()
  Initialize the variables for the module.
- integer function, public parse_args ()
  Parses the command line arguments.
- integer function, public open_input ()
  Open the input files.
- integer function, public open_output ()
  Open the output files.
- subroutine, public close_output ()
  Closes the output file.

Variables

- character(len=max_str_ln), dimension(:), allocatable, public opt_args
  optional arguments that can be passed using `--[name]`

B.9.1 Detailed Description

Operations related to files !

B.9.2 Function/Subroutine Documentation

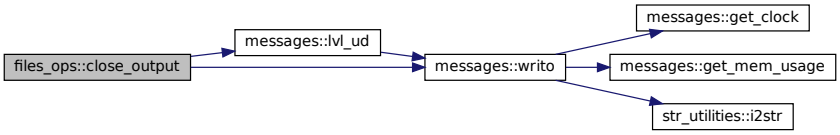
B.9.2.1 close_output()

`subroutine, public files_ops::close_output`

Closes the output file.

Definition at line 703 of file files_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

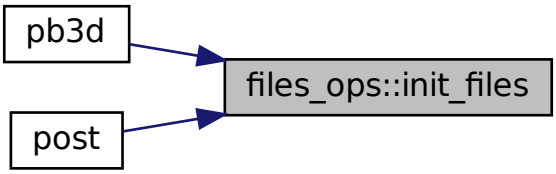
B.9.2.2 init_files()

```
subroutine, public files_ops::init_files
```

Initialize the variables for the module.

Definition at line 25 of file files_ops.f90.

Here is the caller graph for this function:



B.9.2.3 open_input()

```
integer function, public files_ops::open_input
```

Open the input files.

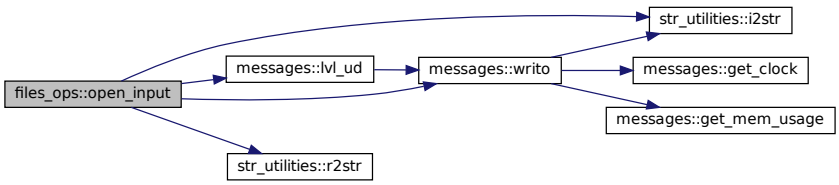
- input file with user options
- equilibrium file in
  - NetCDF for VMEC
  - plain for HELENA

Returns

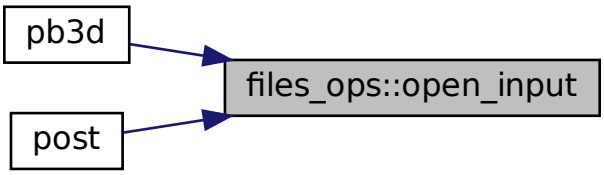
ierr

Definition at line 178 of file files_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.9.2.4 open_output()

`integer function, public files_ops::open_output`

Open the output files.

- output file `.txt`
- shell commands file `.sh`
- HDF5 file: only for PB3D, not for POST.
- memory usage file `.dat`

Also sets some output variables.

Note

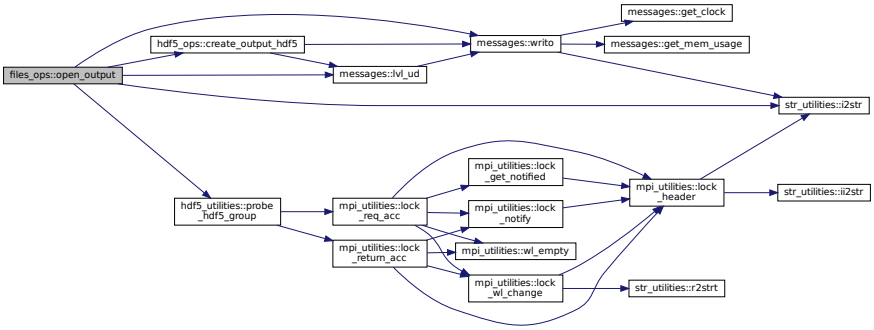
1. memory usage file is only for debug version.
2. There can be resart of a Richardson level for PB3D
3. There can also be a direct jump to the solution for PB3D, if the equilibrium and perturbation phases are already done and saved (see init_files()).
4. In the case of a Richardson restart, PB3D reopens the HDF5 file.

Returns

ierr

Definition at line 529 of file files_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.9.2.5 parse_args()

```
integer function, public files_ops::parse_args
```

Parses the command line arguments.

Note

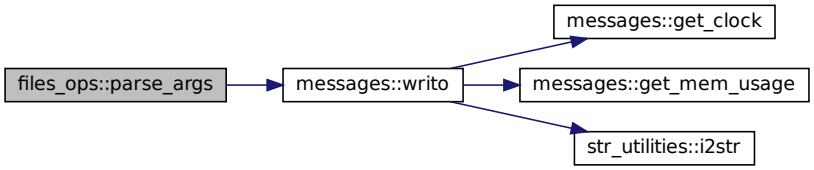
The input arguments are saved in `command_arg`

Returns

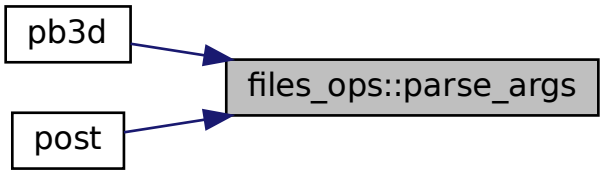
ierr

Definition at line 79 of file files_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.9.3 Variable Documentation

B.9.3.1 opt_args

```
character(len=max_str_ln), dimension(:), allocatable, public files_ops::opt_args
```

optional arguments that can be passed using `--[name]`

Definition at line 19 of file files_ops.f90.

B.10 files_utilities Module Reference

Numerical utilities related to files.

Functions/Subroutines

- integer function, public nextunit (unit)
  Search for available new unit.
- integer function, public skip_comment (file_i, file_name)
  Skips comment when reading a file.
- subroutine, public get_file_info (file_name, file_size, acc_time, mod_time)
  Gets file information.
- character(len=max_str_ln) function, public get_full_pb3d_name (rich_lvl)
  Returns the name of the PB3D output file.
- integer function, public delete_file (file_name)
  Removes a file.
- integer function, public count_lines (file_i)
  Count non-comment lines in a file.

B.10.1 Detailed Description

Numerical utilities related to files.

B.10.2 Function/Subroutine Documentation

B.10.2.1 count_lines()

```
integer function, public files_utilities::count_lines (
               integer, intent(in) file_i )
```

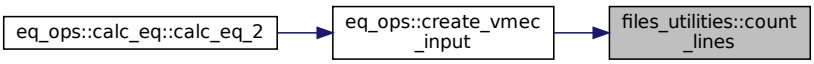
Count non-comment lines in a file.

Parameters

in	*file_i*	file identifier

Definition at line 170 of file files_utilities.f90.

Here is the caller graph for this function:



B.10.2.2 delete_file()

```
integer function, public files_utilities::delete_file (
               character(len=*), intent(inout) file_name )
```

Removes a file.

Returns

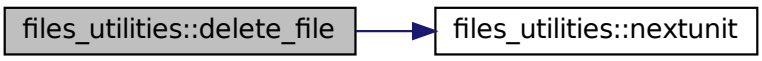
istat

Parameters

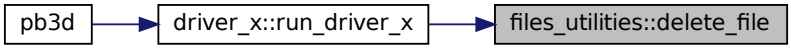
in,out	*file_name*	the name that is deleted

Definition at line 144 of file files_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.10.2.3 get_file_info()

```
subroutine, public files_utilities::get_file_info (
               character(len=*), intent(in) file_name,
               integer, intent(inout), optional file_size,
               integer, intent(inout), optional acc_time,
               integer, intent(inout), optional mod_time )
```

Gets file information.

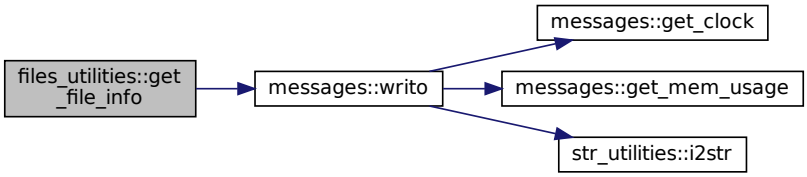
The time informations can be converted to strings using the intrinsic function ”ctime”.

Parameters

in	*file_name*	name of file
in,out	*file_size*	file size
in,out	*acc_time*	file access time
in,out	*mod_time*	file modification time

Definition at line 86 of file files_utilities.f90.

Here is the call graph for this function:



B.10.2.4 get_full_pb3d_name()

```
character(len=max_str_ln) function, public files_utilities::get_full_pb3d_name (
               integer, intent(in), optional rich_lvl )
```

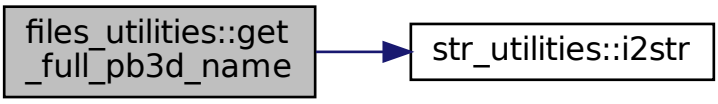
Returns the name of the PB3D output file.

Optionally, the Richardson level can be appended as `_R_[lvl]`.

If not positive, it is ignored.

Definition at line 119 of file files_utilities.f90.

Here is the call graph for this function:



B.10.2.5 nextunit()

```
integer function, public files_utilities::nextunit (
               integer, intent(out), optional unit )
```

Search for available new unit.

`lun_min` and `lun_max` define the range of possible luns to check.

The unit value is returned by the function, and also by the optional argument. This allows the function to be used directly in an open statement, and optionally save the result in a local variable.

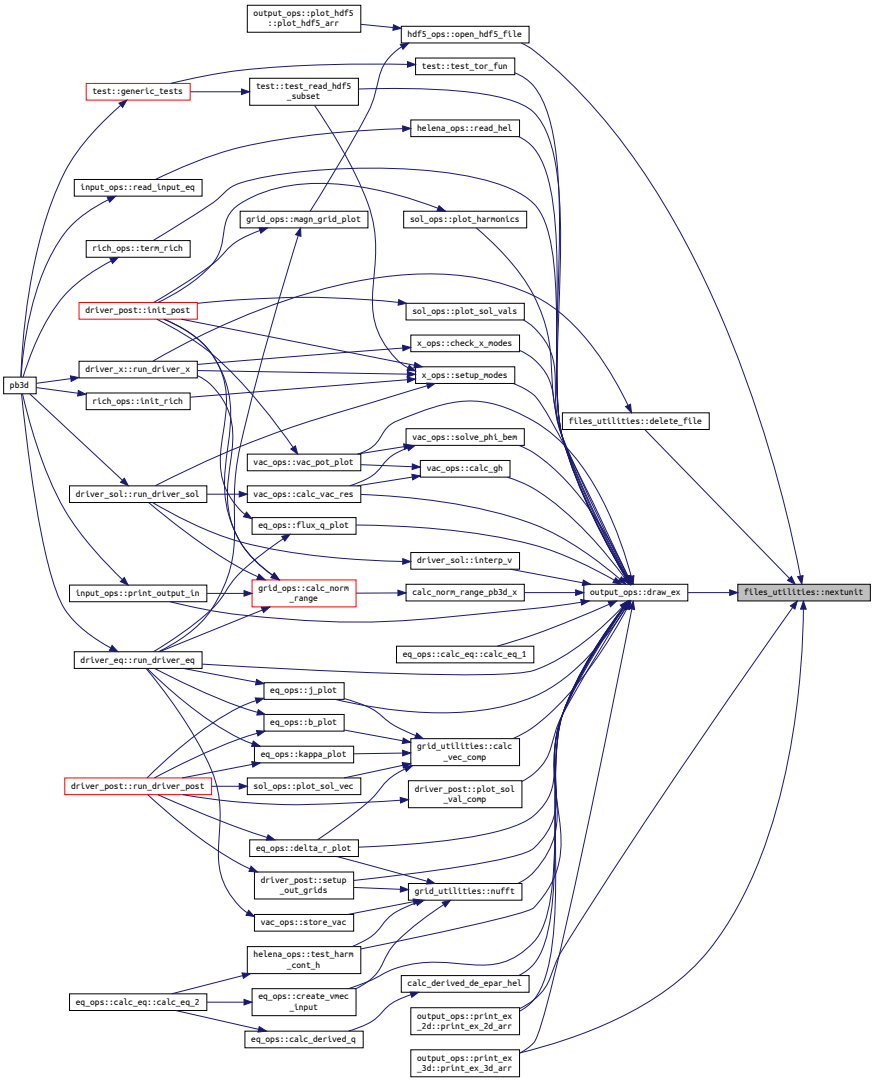
If no units are available, -1 is returned.

See also

Adapted from http://fortranwiki.org/fortran/show/newunit

Definition at line 27 of file files_utilities.f90.

Here is the caller graph for this function:



B.10.2.6 skip_comment()

```
integer function, public files_utilities::skip_comment (
              integer, intent(in) file_i,
              character(len=*), intent(in), optional file_name )
```

Skips comment when reading a file.

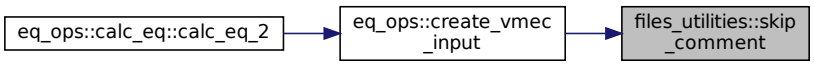
By comment, a line is meant that starts with the character #.

Returns

ierr

Definition at line 55 of file files_utilities.f90.

Here is the caller graph for this function:



B.11 grid_ops Module Reference

Operations that have to do with the grids and different coordinate systems.

Functions/Subroutines

- integer function, public calc_norm_range (style, in_limits, eq_limits, X_limits, sol_limits, r_F_eq, r_F_X, r_F_sol, jq)
  Calculates normal range for the input grid, the equilibrium grid and/or the solution grid.
- integer function, public setup_grid_eq (grid_eq, eq_limits)
  Sets up the equilibrium grid.
- integer function, public setup_grid_eq_b (grid_eq, grid_eq_B, eq, only_half_grid)
  Sets up the field-aligned equilibrium grid.
- integer function, public setup_grid_x (grid_eq, grid_X, r_F_X, X_limits)
  Sets up the general perturbation grid, in which the perturbation variables are calculated.
- integer function, public setup_grid_sol (grid_eq, grid_X, grid_sol, r_F_sol, sol_limits)
  Sets up the general solution grid, in which the solution variables are calculated.
- integer function, public calc_ang_grid_eq_b (grid_eq, eq, only_half_grid)
  Calculate equilibrium grid that follows magnetic field lines.
- integer function, public redistribute_output_grid (grid, grid_out, no_outer_trim)
  Redistribute a grid to match the normal distribution of solution grid.
- integer function, public magn_grid_plot (grid)
  Plots the grid in real 3-D space.
- integer function, public print_output_grid (grid, grid_name, data_name, rich_lvl, par_div, remove_↩ previous_arrs)
  Print grid variables to an output file.

Variables

- logical, public debug_calc_ang_grid_eq_b = .false.
  plot debug information for calc_ang_grid_eq_b()

B.11.1 Detailed Description

Operations that have to do with the grids and different coordinate systems.

B.11.2 Function/Subroutine Documentation

B.11.2.1 calc_ang_grid_eq_b()

```
integer function, public grid_ops::calc_ang_grid_eq_b (
               type(grid_type), intent(inout) grid_eq,
               type(eq_1_type), intent(in), target eq,
               logical, intent(in), optional only_half_grid )
```

Calculate equilibrium grid that follows magnetic field lines.

This grid is different from the equilibrium grid from setup_grid_eq for HELENA, as the latter is the output grid from HELENA, which is situated in a single poloidal cross-section, as opposed to a really field-aligne grid.

For VMEC, this is not used as the grid is field-aligned from the start.

Note

1. The end-points are included for the grids in the parallel direction. This is to facilitate working with the trapezoidal rule or Simpson's 3/8 rule for integration. This is **NOT** valid in general!
2. by setting the flag `only_half_grid`, only the even points of the parallel grid are calculated, which is useful for higher Richardson levels with VMEC so that only new angular points are calculated and the old ones reused.

Returns

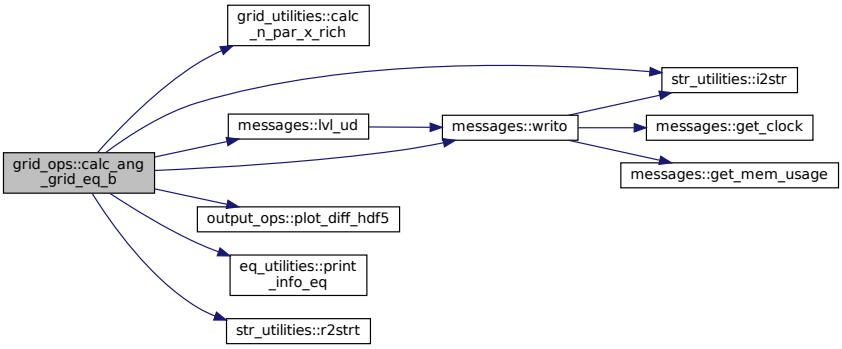
ierr

Parameters

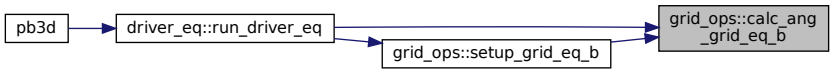
in,out	*grid_eq*	equilibrium grid of which to calculate angular part
in	*eq*	flux equilibrium variables
in	*only_half_grid*	calculate only half grid with even points

Definition at line 798 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.2 calc_norm_range()

```
integer function, public grid_ops::calc_norm_range (
               character(len=*), intent(in) style,
               integer, dimension(2), intent(inout), optional in_limits,
               integer, dimension(2), intent(inout), optional eq_limits,
               integer, dimension(2), intent(inout), optional X_limits,
               integer, dimension(2), intent(inout), optional sol_limits,
               real(dp), dimension(:), intent(inout), optional r_F_eq,
               real(dp), dimension(:), intent(inout), optional, allocatable r_F_X,
               real(dp), dimension(:), intent(inout), optional r_F_sol,
               real(dp), dimension(:), intent(in), optional jq )
```

Calculates normal range for the input grid, the equilibrium grid and/or the solution grid.

General workings, depending on `X_grid_style`:

1 (equilibrium)	2 (solution)	3 (enriched)
calc_eq()	calc_eq()	calc_eq()
	redistribute to sol	add points to optimize
	interpolate to sol	interpolate to X
calc_x()	calc_x()	calc_x()
redistribute to sol	copy to sol	redistribute to sol
interpolate to sol		interpolate to sol

Returns

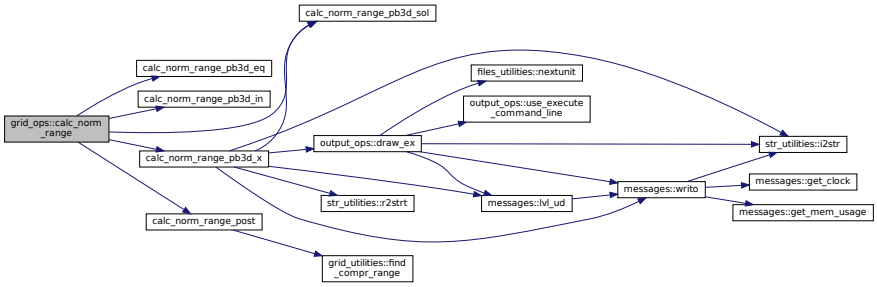
ierr

Parameters

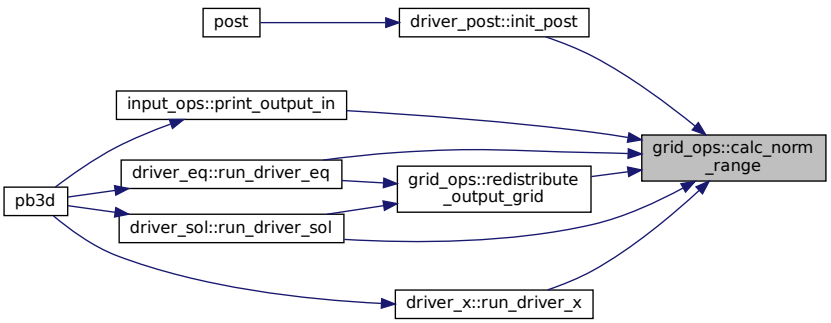
in	*style*	style of calculation (PB3D: in, eq, X or sol; POST)
in,out	*in_limits*	min. and max. index of in grid
in,out	*eq_limits*	min. and max. index of eq grid for this process
in,out	*x_limits*	min. and max. index of X grid for this process
in,out	*sol_limits*	min. and max. index of sol grid for this process
in,out	*r_f_eq*	equilibrium r_F
in,out	*r_f_x*	perturbation r_F
in,out	*r_f_sol*	solution r_F
in	*jq*	q_saf (pol. flux) or rot_t (tor. flux) in total Flux variables

Definition at line 46 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.3 magn_grid_plot()

```
integer function, public grid_ops::magn_grid_plot (
               type(grid_type), intent(in) grid )
```

Plots the grid in real 3-D space.

This creates an animation that can be used by ParaView or VisIt.

The equilibrium grid should contain the fieldline-oriented angles with `ang_1` the parallel angle and `ang_2` the field line label.

See also

See grid_vars.grid_type for a discussion on `ang_1` and `ang_2`.

Note

1. This procedure does not use `n_theta_plot` and `n_zeta_plot` from num_vars, but instead temporarily overwrites them with its own, since it is suposed to be 3-D also in the axisymmetric case.
2. The implementation is currently very slow.

Returns

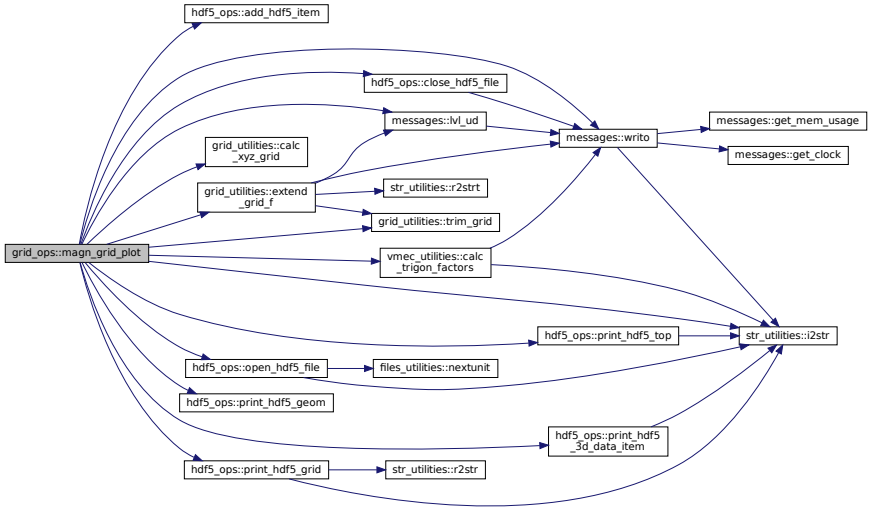
ierr

Parameters

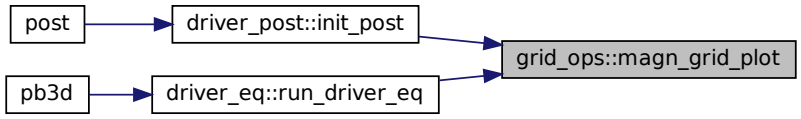
in	*grid*	fieldline-oriented equilibrium grid

Definition at line 1115 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.4 print_output_grid()

```
integer function, public grid_ops::print_output_grid (
               type(grid_type), intent(in) grid,
               character(len=*), intent(in) grid_name,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional par_div,
               logical, intent(in), optional remove_previous_arrs )
```

Print grid variables to an output file.

If `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

Optionally, it can be specified that this is a divided parallel grid, corresponding to the variable `eq_jobs_lims` with index `eq_job_nr`. In this case, the total grid size is adjusted to the one specified by `eq_jobs_lims` and the grid is written as a subset.

Note

`grid_` is added in front the data_name.

Returns

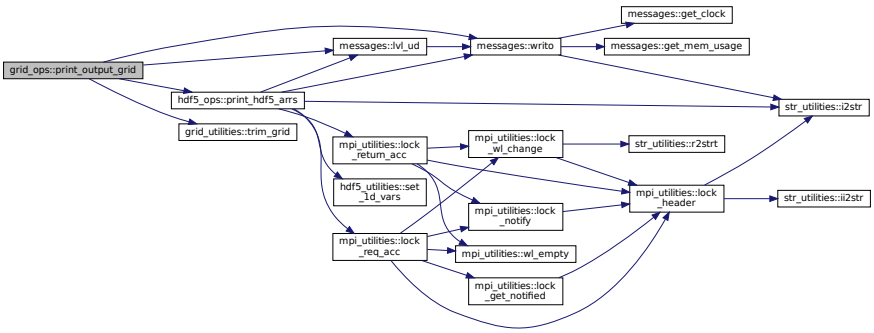
ierr

Parameters

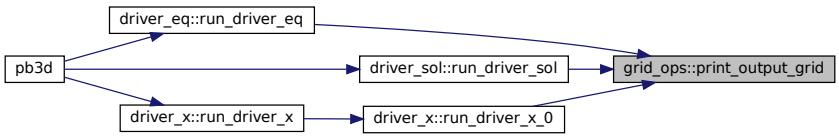
`in`	*grid*	grid variables
`in`	*grid_name*	name to display
`in`	*data_name*	name under which to store
`in`	*rich_lvl*	Richardson level to reconstruct
`in`	*par_div*	is a parallely divided grid
`in`	*remove_previous_arrs*	remove previous variables if present

Definition at line 1489 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.5 redistribute_output_grid()

```
integer function, public grid_ops::redistribute_output_grid (
               type(grid_type), intent(in) grid,
               type(grid_type), intent(inout) grid_out,
               logical, intent(in), optional no_outer_trim )
```

Redistribute a grid to match the normal distribution of solution grid.

The routine first calculates the smallest eq range that comprises the sol range. Then, it gets the lowest equilibrium limits able to setup an output grid that starts at index 1. After determining the output grid, it then sends the variables to their new processes using MPI.

Note

1. Only the Flux variables are saved.
2. the redistributed grid has trimmed outer limits, i.e. it starts at 1 and ends at the upper limit of the last process. This can be turned off optionally using `no_outer_trim`.

Returns

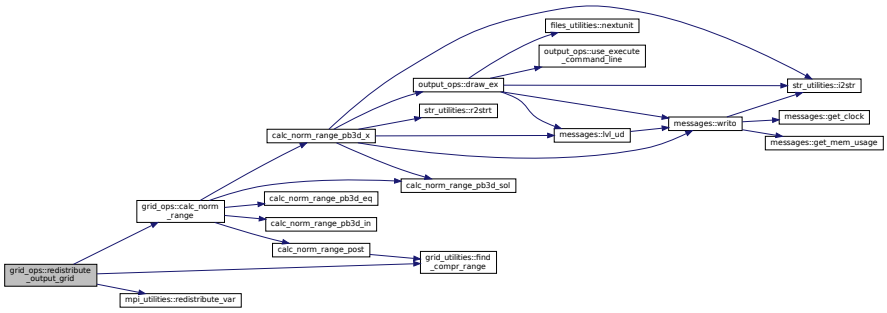
ierr

Parameters

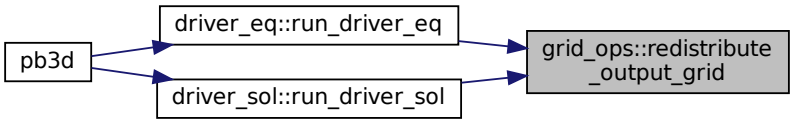
`in`	*grid*	equilibrium grid variables
`in,out`	*grid_out*	redistributed equilibrium grid variables
`in`	*no_outer_trim*	do not trim the outer limits

Definition at line 995 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.6 setup_grid_eq()

```
integer function, public grid_ops::setup_grid_eq (
               type(grid_type), intent(inout) grid_eq,
               integer, dimension(2), intent(in) eq_limits )
```

Sets up the equilibrium grid.

The following variables are calculated:

- equilibrium variables (eq)
- perturbation variables (X)

For the implementation of the equilibrium grid the normal part of the grid is always given by the output of the equilibrium code, but the angular part depends on the style:

- VMEC: The output is analytic in the angular coordinates, so field-aligned coordinates are used. Later, in calc_ang_grid_eq_b(), the angular variables are calculated.
- HELENA: The output is given on a poloidal grid (no toroidal dependency due to axisymmetry), which is used.

Returns

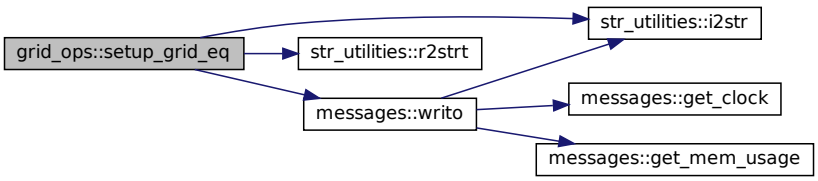
ierr

Parameters

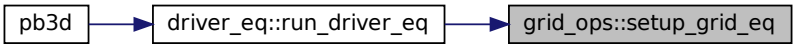
in,out	*grid_eq*	equilibrium grid
in	*eq_limits*	min. and max. index of eq grid of this process

Definition at line 529 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.7 setup_grid_eq_b()

```
integer function, public grid_ops::setup_grid_eq_b (
               type(grid_type), intent(inout) grid_eq,
               type(grid_type), intent(inout) grid_eq_B,
               type(eq_1_type), intent(in) eq,
               logical, intent(in), optional only_half_grid )
```

Sets up the field-aligned equilibrium grid.

This serves as a bridge to the solution grid, as it contains the same normal coordinate as the general grid, but the angular coordinates are defined by the solution grid.

Optionally, only half the grid can be calculated (i.e. only the even points), which is used for Richardson levels greater than 1.

Note

In contrast to `setup_grid_eq`, the angular coordinates are also calculated here.

Returns

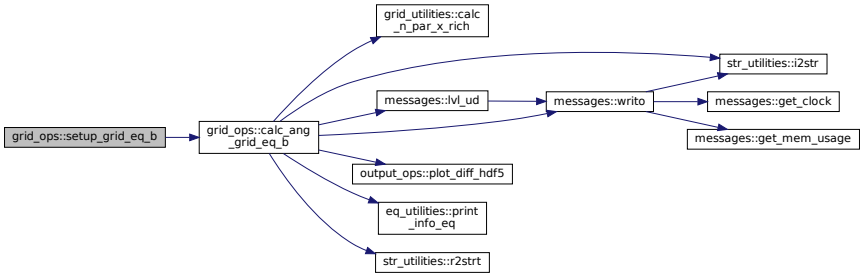
ierr

Parameters

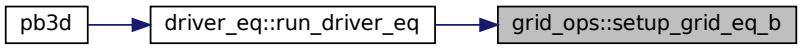
in,out	*grid_eq*	general equilibrium grid
in,out	*grid_eq_b*	field-aligned equilibrium grid
in	*eq*	flux equilibrium variables
in	*only_half_grid*	calculate only half grid with even points

Definition at line 601 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.8 setup_grid_sol()

```
integer function, public grid_ops::setup_grid_sol (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(grid_type), intent(inout) grid_sol,
               real(dp), dimension(:), intent(in) r_F_sol,
               integer, dimension(2), intent(in) sol_limits )
```

Sets up the general solution grid, in which the solution variables are calculated.

For the solution grid, only one parallel point is used, but possibly multiple geodesic points, equal to the number of field lines, `n_alpha`.

Returns

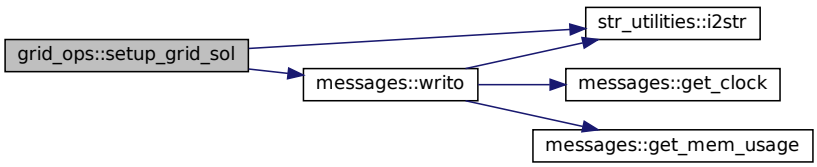
ierr

Parameters

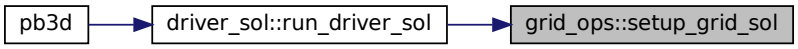
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in,out	*grid_sol*	solution grid
in	*r_f_sol*	points of solution grid
in	*sol_limits*	min. and max. index of sol grid of this process

Definition at line 728 of file grid_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.11.2.9 setup_grid_x()

```
integer function, public grid_ops::setup_grid_x (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(inout) grid_X,
               real(dp), dimension(:), intent(in), allocatable r_F_X,
               integer, dimension(2), intent(in) X_limits )
```

Sets up the general perturbation grid, in which the perturbation variables are calculated.

For `X_grid_style` 1, this grid is identical to the equilibrium grid, and for `X_grid_style` 2,it has the same angular extent but with different normal points, indicated by the variable `r_F_X`.

Returns

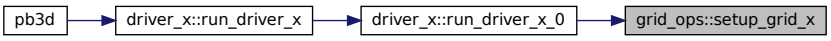
ierr

Parameters

in	*grid_eq*	equilibrium grid
in,out	*grid_x*	perturbation grid
in	*r_f_x*	points of perturbation grid
in	*x_limits*	min. and max. index of perturbation grid of this process

Definition at line 655 of file grid_ops.f90.

Here is the caller graph for this function:



B.11.3 Variable Documentation

B.11.3.1 debug_calc_ang_grid_eq_b

```
logical, public grid_ops::debug_calc_ang_grid_eq_b = .false.
```

plot debug information for calc_ang_grid_eq_b()

Note

Debug version only

Definition at line 25 of file grid_ops.f90.

B.12 grid_utilities Module Reference

Numerical utilities related to the grids and different coordinate systems.

Interfaces and Types

- interface calc_eqd_grid

  Calculate grid of equidistant points, where optionally the last point can be excluded.
- interface calc_tor_diff

  Calculates the toroidal difference for a magnitude calculated on three toroidal points: two extremities and one in the middle.
- interface coord_e2f

  Converts Equilibrium coordinates $(r,\theta,\zeta)_E$ *to Flux coordinates* $(r,\theta,\zeta)_F$.
- interface coord_f2e

  Converts Flux coordinates $(r,\theta,\zeta)_F$ *to Equilibrium coordinates* $(r,\theta,\zeta)_E$.

Functions/Subroutines

- integer function coord_e2f_rtz (grid_eq, r_E, theta_E, zeta_E, r_F, theta_F, zeta_F, r_E_array, r_F_array)

  version with r, theta and zeta
- integer function coord_e2f_r (grid_eq, r_E, r_F, r_E_array, r_F_array)

  version with only r
- integer function, public calc_xyz_grid (grid_eq, grid_XYZ, X, Y, Z, L, R)

  Calculates X, Y and Z on a grid `grid_XYZ`, determined through its E(quilibrium) coordinates.
- integer function, public extend_grid_f (grid_in, grid_ext, grid_eq, n_theta_plot, n_zeta_plot, lim_↩ theta_plot, lim_zeta_plot)

  Extend a grid angularly.
- integer function, public copy_grid (grid_A, grid_B, lims_B, i_lim)

  Copy a grid A to a new grid B, that was not yet initialized.
- integer function, public calc_int_vol (ang_1, ang_2, norm, J, f, f_int)

  Calculates volume integral on a 3D grid.
- integer function, public trim_grid (grid_in, grid_out, norm_id)

  Trim a grid, removing any overlap between the different regions.
- integer function, public untrim_grid (grid_in, grid_out, size_ghost)

  Untrims a trimmed grid by introducing an asymmetric ghost regions at the right.
- integer function, public calc_vec_comp (grid, grid_eq, eq_1, eq_2, v_com, norm_disc_prec, v_mag, base_name, max_transf, v_flux_tor, v_flux_pol, XYZ, compare_tor_pos)

  Calculates contra- and covariant components of a vector in multiple coordinate systems.
- integer function, public calc_n_par_x_rich (n_par_X_rich, only_half_grid)

  Calculates the local number of parallel grid points for this Richardson level, taking into account that it ould be half the actual number.
- integer function, public nufft (x, f, f_F, plot_name)

  calculates the cosine and sine mode numbers of a function defined on a non-regular grid.
- subroutine, public find_compr_range (r_F, lim_r, lim_id)

  finds smallest range that comprises a minimum and maximum value.
- integer function, public calc_arc_angle (grid, eq_1, eq_2, arc, use_E)

  Calculate arclength angle.

Variables

- logical, public debug_calc_int_vol = .false.

  plot debug information for calc_int_vol()
- logical, public debug_calc_vec_comp = .false.

  plot debug information for calc_vec_comp()

B.12.1 Detailed Description

Numerical utilities related to the grids and different coordinate systems.

B.12.2 Function/Subroutine Documentation

B.12.2.1 calc_arc_angle()

```
integer function, public grid_utilities::calc_arc_angle (
                type(grid_type), intent(in) grid,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(in) eq_2,
                real(dp), dimension(:,:,:), intent(inout), allocatable arc,
                logical, intent(in), optional use_E )
```

Calculate arclength angle.

This angle is based on the calculation of the arclength from the start of the grid, and then normalizing it from $\min(\gamma) \ldots \max(\gamma)$, where γ is the parallel angle.

By default, the Flux variables are used, but this can be changed with `use_E`.

Note

For field-aligned grids the projection is taken on the poloidal cross-section. For non-axisymmetric equilibria, this makes little sense.

Returns

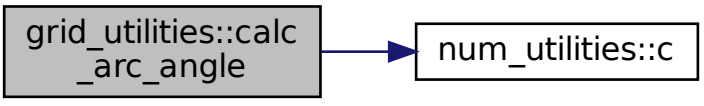
ierr

Parameters

in	*grid*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables
in,out	*arc*	arclength angle
in	*use_e*	use E variables instead of F

Definition at line 3039 of file grid_utilities.f90.

Here is the call graph for this function:



B.12.2.2 calc_int_vol()

```
integer function, public grid_utilities::calc_int_vol (
               real(dp), dimension(:,:,:), intent(in) ang_1,
               real(dp), dimension(:,:,:), intent(in) ang_2,
               real(dp), dimension(:), intent(in) norm,
               real(dp), dimension(:,:,:), intent(in) J,
               complex(dp), dimension(:,:,:,:), intent(in) f,
               complex(dp), dimension(:), intent(inout) f_int )
```

Calculates volume integral on a 3D grid.

Two angular and one normal variable have to be provided on a the grid.

If the `i`'th dimension of the grid is equal to one, the function to be integrated is assumed not to vary in this dimension.

Furthermore, if `i` is 1 or 2, the corresponding `i`'th (angular) variable is the only variable that is assumed to vary in that dimension. The other angular variable as well as the normal variable are assumed to be constant like the function itself, resulting in a factor 2π. However, if `i` is 3, an error is displayed as this does not represent a physical situation.

A common case through which to understand this is the axisymmetric case where the first angular variable θ varies in the dimensions 1 and 3, the second angular variable ζ varies only in the dimension 2, and the normal variable only varies in the dimension 3.

Alternatively, there is the case of a grid-aligned set of coordinates θ and α, where the first dimension corresponds to the direction along the magnetic field line, the second to the geodesical direction and the third to the normal direction. If the calculations for different field lines are decoupled, the variation in the second dimension is not taken into account and no integration happens along it.

Internally, the angular variables and the normal variable are related to the coordinates (x, y, z) that correspond to the three dimensions. They thus form a computational orthogonal grid to which the original coordinates are related through the transformation of Jacobians:

$$\mathcal{J}_{xyz} = \mathcal{J}_{\mathsf{F}} \frac{\partial r_{\mathsf{F}}}{\partial z} \left(\frac{\partial \gamma_1}{\partial x} \frac{\gamma_2}{\partial y} - \frac{\partial \gamma_1}{\partial y} \frac{\gamma_2}{\partial x} \right),$$

where γ_1 and γ_2 are `ang_1` and `ang_2`, so that the integral becomes

$$\sum_{x,y,z} \left[f(x,y,z)\, \mathcal{J}_{\mathsf{F}} \frac{\partial r_{\mathsf{F}}}{\partial z} \left(\frac{\partial \gamma_1}{\partial x} \frac{\gamma_2}{\partial y} - \frac{\partial \gamma_1}{\partial y} \frac{\gamma_2}{\partial x} \right) \Delta_x \Delta_y \Delta_z \right]$$

where Δ_x, Δ_y and Δ_z are all trivially equal to 1.

The integrand has to be evaluated at the intermediate positions inside the cells. This is done by taking the average of the $2^3 = 8$ points for `fj` = fJ_{F} as well as the transformation of the Jacobian to (x, y, z) coordinates.

See also

See grid_vars.grid_type for a discussion on `ang_1` and `ang_2`.

Note

1. if the coordinates are independent, this method is equivalent to the repeated numerical integration using the trapezoidal method, **NOT** Simpson's 3/8 rule!
2. by setting `debug_calc_int_vol`, this method can be compared to the trapezoidal and simple method for independent coordinates, again **NOT** for Simpson's 3/8 rule!
3. The Simpson's 3/8 rule could be developed but it is not of great importance.

Returns

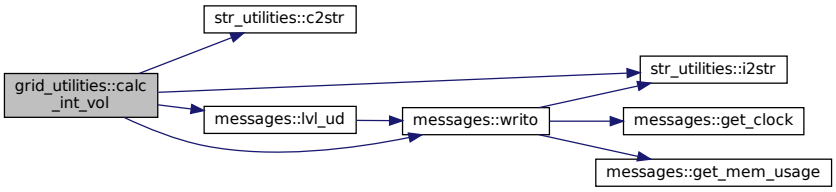
ierr

Parameters

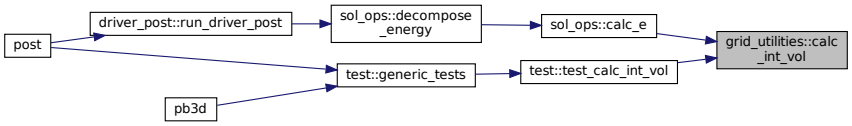
in	*ang↩ _1*	coordinate variable γ_1
in	*ang↩ _2*	coordinate variable γ_2
in	*norm*	coordinate variable r_F
in	*j*	Jacobian
in	*f*	input f(n_par,n_geo,n_r,size_X^2)
in,out	*f_int*	output integrated f

Definition at line 1320 of file grid_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.12.2.3 calc_n_par_x_rich()

```
integer function, public grid_utilities::calc_n_par_x_rich (
              integer, intent(inout) n_par_X_rich,
              logical, intent(in), optional only_half_grid )
```

Calculates the local number of parallel grid points for this Richardson level, taking into account that it ould be half the actual number.

See also

calc_ang_grid_eq_b().

Returns

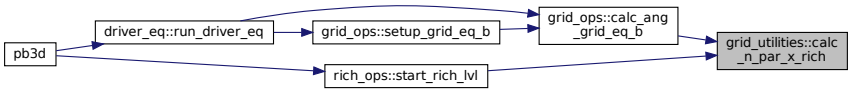
ierr

Parameters

in,out	*n_par_x_rich*	n_par_X for this Richardson level
in	*only_half_grid*	calculate only half grid with even points

Definition at line 2862 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.2.4 calc_vec_comp()

```
integer function, public grid_utilities::calc_vec_comp (
               type(grid_type), intent(in) grid,
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               real(dp), dimension(:,:,:,:), intent(inout) v_com,
               integer, intent(in) norm_disc_prec,
               real(dp), dimension(:,:,:), intent(inout), optional v_mag,
               character(len=*), intent(in), optional base_name,
               integer, intent(in), optional max_transf,
               real(dp), dimension(:,:), intent(inout), optional, allocatable v_flux_tor,
               real(dp), dimension(:,:), intent(inout), optional, allocatable v_flux_pol,
               real(dp), dimension(:,:,:,:), intent(in), optional XYZ,
               logical, intent(in), optional compare_tor_pos )
```

Calculates contra- and covariant components of a vector in multiple coordinate systems.

Chain of coordinate systems considered:

1. Flux F: (α, ψ, θ),
2. Magnetic M: (ψ, θ, ζ),
   - H: (ψ, θ, ϕ) if HELENA is used,
   - V: (r, θ, ζ) if VMEC is used,
3. Cylindrical C: (R, φ, Z),
4. Cartesian X: (X, Y, Z), as well as the magnitude, starting from the input in Flux coordinates. Note that the Cartesian components in X, Y and Z can be used to plot the real vector and that covariant components should be equal to contravariant components in this coordinate system.

Also, the fluxes can be calculated and plot by providing a `base_name`.

By default, the cartesian components are returned, but this can be indicated differently by providing `max_transf`.

Note

1. Plots for different Richardson levels can be combined to show the total grid by just plotting them all individually and sequentially.
2. The metric factors and transformation matrices have to be allocated. They can be calculated using the routines from eq_ops, for `deriv = [0,0,0]`.
3. For VMEC, the trigonometric factors of `grid_XYZ` must be calculated beforehand. Optionally, by providing `X`, `Y` and `Z`, the ones calculated in this routine are overwritten. This is useful for, for example, slab geometries.
4. The normalization factors are taken into account and the output is transformed back to unnormalized values.

Returns

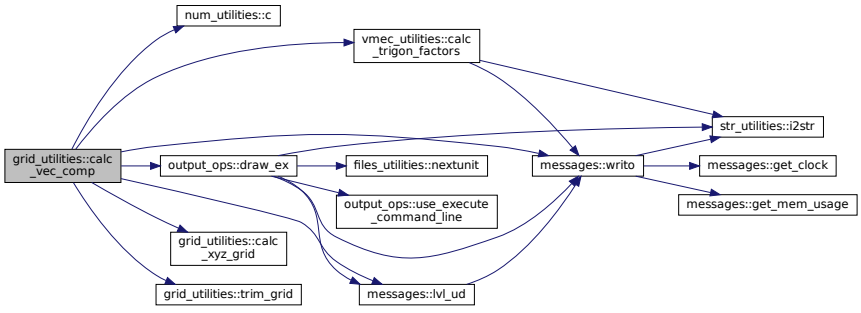
ierr

Parameters

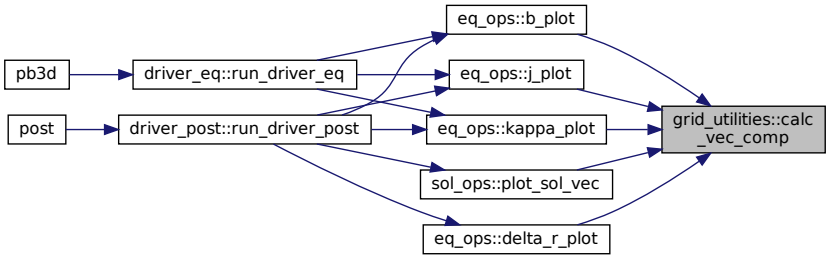
`in`	*grid*	grid on which vector is calculated
`in`	*grid_eq*	grid on which equilibrium variables are calculated
`in`	*eq_1*	flux equilibrium quantities
`in`	*eq_2*	metric equilibrium variables
`in,out`	*v_com*	covariant and contravariant components (`dim1,dim2,dim3,3,2`)
`in`	*norm_disc_prec*	precision for normal derivatives
`in,out`	*v_mag*	magnitude (`dim1,dim2,dim3`)
`in`	*base_name*	base name for output plotting
`in`	*max_transf*	maximum transformation level (2: Magnetic, 3: Equilibrium, 4: Cylindrical, 5: Cartesian [def])
`in,out`	*v_flux_pol*	poloidal flux as function of normal coordinate for all toroidal positions
`in,out`	*v_flux_tor*	toroidal flux as function of normal coordinate for all poloidal positions
`in`	*xyz*	X, Y and Z of grid
`in`	*compare_tor_pos*	compare toroidal positions

Definition at line 1859 of file grid_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.12.2.5 calc_xyz_grid()

```
integer function, public grid_utilities::calc_xyz_grid (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_XYZ,
               real(dp), dimension(:,:,:), intent(inout) X,
               real(dp), dimension(:,:,:), intent(inout) Y,
               real(dp), dimension(:,:,:), intent(inout) Z,
               real(dp), dimension(:,:,:), intent(inout), optional L,
               real(dp), dimension(:,:,:), intent(inout), optional R )
```

Calculates X, Y and Z on a grid `grid_XYZ`, determined through its E(quilibrium) coordinates.

Furthermore, a grid `grid_eq` must be provided, which is the grid in which the variables concerning R and Z are tabulated, i.e. the full equilibrium grid in E(quilibrium) coordinates.

Of this grid, however, only `r_E` is used, and the rest ignored. It can therefore be provided without the angular part, i.e. by reconstructing it with a subset.

If VMEC is the equilibrium model, this routine also optionally calculates λ on the grid, as this is also needed some times. for HELENA this variable is not used.

Optionally, R and λ (only for VMEC) can be returned.

Note

1. For VMEC, the trigonometric factors of `grid_XYZ` must be calculated beforehand.
2. The normalization factor `R_0` for length is taken into account and the output is transformed back to unnormalized values.

Returns

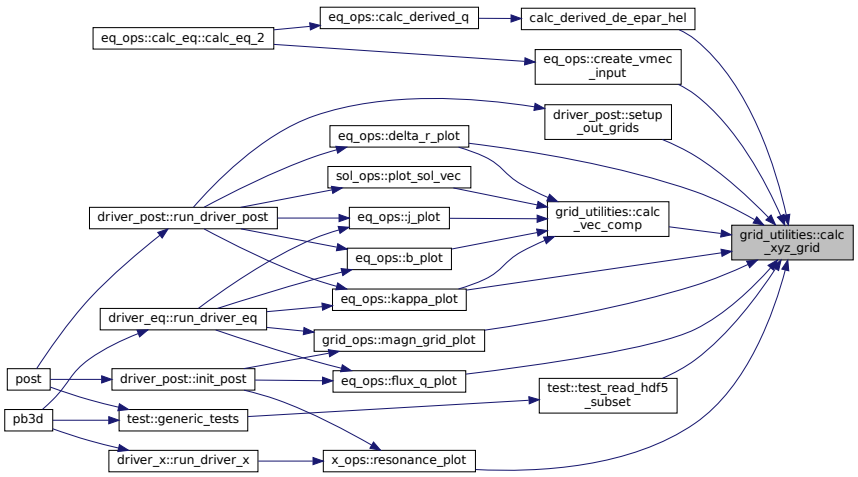
ierr

Parameters

in	*grid_eq*	equilibrium grid
in	*grid_xyz*	grid for which to calculate X, Y, Z and optionally L
in,out	*x*	X of grid
in,out	*y*	Y of grid
in,out	*z*	Z of grid
in,out	*l*	λ of grid
in,out	*r*	R of grid

Definition at line 799 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.2.6 coord_e2f_r()

```
integer function grid_utilities::coord_e2f_r (
               type(grid_type), intent(in) grid_eq,
               real(dp), dimension(:), intent(in) r_E,
               real(dp), dimension(:), intent(inout) r_F,
               real(dp), dimension(:), intent(in), optional, target r_E_array,
               real(dp), dimension(:), intent(in), optional, target r_F_array )
```

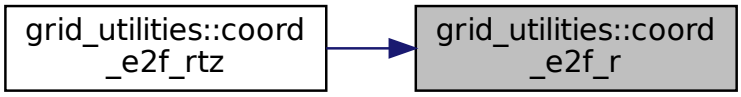
version with only r

Parameters

in	*grid_eq*	equilibrium grid (for normal local limits)
in	*r_e*	(local) r_E
in,out	*r_f*	(local) r_F
in	*r_e_array*	optional array that defines mapping between two coord. systems
in	*r_f_array*	optional array that defines mapping between two coord. systems

Definition at line 473 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.2.7 coord_e2f_rtz()

```
integer function grid_utilities::coord_e2f_rtz (
               type(grid_type), intent(in) grid_eq,
               real(dp), dimension(:), intent(in) r_E,
               real(dp), dimension(:,:,:), intent(in) theta_E,
               real(dp), dimension(:,:,:), intent(in) zeta_E,
               real(dp), dimension(:), intent(inout) r_F,
               real(dp), dimension(:,:,:), intent(inout) theta_F,
               real(dp), dimension(:,:,:), intent(inout) zeta_F,
               real(dp), dimension(:), intent(in), optional, target r_E_array,
               real(dp), dimension(:), intent(in), optional, target r_F_array )
```

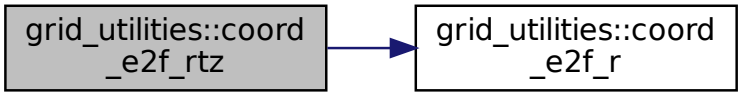
version with r, theta and zeta

Parameters

in	*grid_eq*	equilibrium grid (for normal local limits)
in	*r_e*	(local) r_E
in	*theta_e*	θ_E
in	*zeta_e*	ζ_E
in,out	*r_f*	(local) r_F
in,out	*theta_f*	θ_F
in,out	*zeta_f*	ζ_F
in	*r_e_array*	optional array that defines mapping between two coord. systems
in	*r_f_array*	optional array that defines mapping between two coord. systems

Definition at line 359 of file grid_utilities.f90.

Here is the call graph for this function:



B.12.2.8 copy_grid()

```
integer function, public grid_utilities::copy_grid (
               class(grid_type), intent(in) grid_A,
               class(grid_type), intent(inout) grid_B,
               integer, dimension(3,2), intent(in), optional lims_B,
               integer, dimension(2), intent(in), optional i_lim )
```

Copy a grid A to a new grid B, that was not yet initialized.

This new grid can contain a subsection of the previous grid in all dimensions. It can also be a divided grid, by providing the limits

Note

1. The normal limits for the divided grid should be given in terms of the normal dimension of `grid` B.
2. if the grids are not purely normal, the procedure can currently only handle the copying of a `grid_A` that is not divided.

Returns

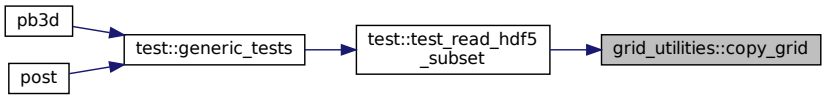
ierr

Parameters

`in`	*grid_a*	grid to be initialized
`in,out`	*grid_b*	grid to be initialized
`in`	*lims_b*	ranges for subset of grid
`in`	*i_lim*	min. and max. local normal index

Definition at line 1189 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.2.9 extend_grid_f()

```
integer function, public grid_utilities::extend_grid_f (
               type(grid_type), intent(in) grid_in,
               type(grid_type), intent(inout) grid_ext,
               type(grid_type), intent(in), optional grid_eq,
               integer, intent(in), optional n_theta_plot,
               integer, intent(in), optional n_zeta_plot,
               real(dp), dimension(2), intent(in), optional lim_theta_plot,
               real(dp), dimension(2), intent(in), optional lim_zeta_plot )
```

Extend a grid angularly.

This is done using equidistant variables of `n_theta_plot` and `n_zeta_plot` angular and own `loc_n_r` points in F coordinates.

Optionally, the grid can also be converted to E coordinates if the equilibrium grid is provided. This is required for many operations, such as the calculation of X, Y and Z through calc_XYZ_grid().

Note

For VMEC, it can be slow, as grid_utilities.coord_f2e() is used.

Returns

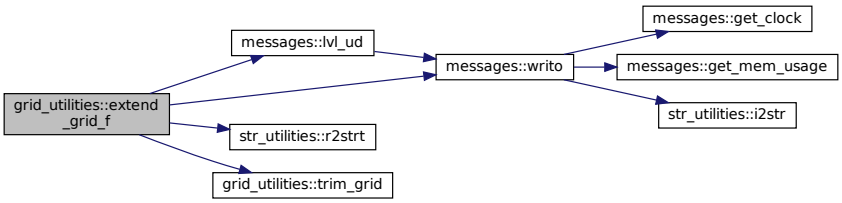
ierr

Parameters

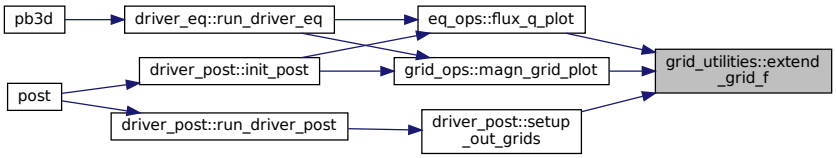
in	*grid_in*	grid to be extended
in,out	*grid_ext*	extended grid
in	*grid_eq*	equilibrium grid
in	*n_theta_plot*	number of poins in theta direction
in	*n_zeta_plot*	number of poins in zeta direction
in	*lim_theta_plot*	limits in theta
in	*lim_zeta_plot*	limits in zeta

Definition at line 1096 of file grid_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.12.2.10 find_compr_range()

```
subroutine, public grid_utilities::find_compr_range (
               real(dp), dimension(:), intent(in) r_F,
               real(dp), dimension(2), intent(in) lim_r,
               integer, dimension(2), intent(inout) lim_id )
```

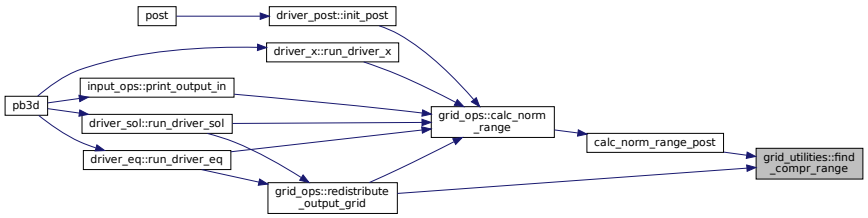
finds smallest range that comprises a minimum and maximum value.

Parameters

in	*r_f*	all values of coordinate
in	*lim↩ _r*	limiting range
in,out	*lim↩ _id*	limiting indices

Definition at line 2995 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.2.11 nufft()

```
integer function, public grid_utilities::nufft (
               real(dp), dimension(:), intent(in) x,
               real(dp), dimension(:), intent(in) f,
               real(dp), dimension(:,:), intent(inout), allocatable f_F,
               character(len=*), intent(in), optional plot_name )
```

calculates the cosine and sine mode numbers of a function defined on a non-regular grid.

If a plot name is provided, the modes are plotted.

Note

The fundamental interval is assumed to be $0 \ldots 2\pi$ but there should be no overlap between the first and last point.

Returns

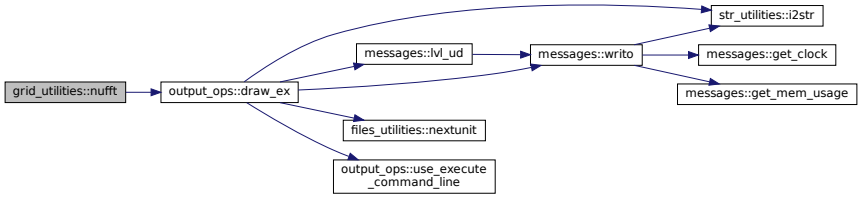
ierr

Parameters

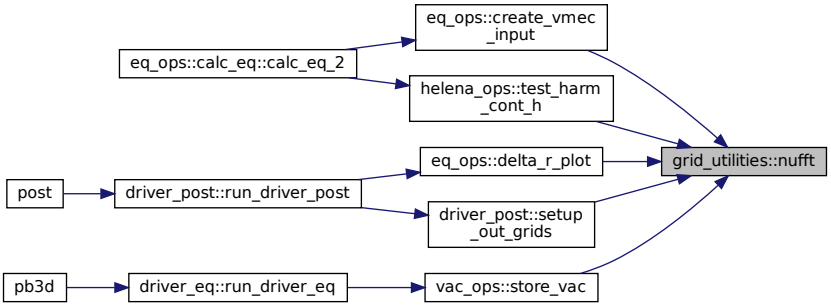
in	*x*	coordinate values
in	*f*	function values
in,out	*f_f*	Fourier modes for $\cos$ and $\sin$
in	*plot_name*	name of possible plot

Definition at line 2901 of file grid_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.12.2.12 trim_grid()

```
integer function, public grid_utilities::trim_grid (
               type(grid_type), intent(in) grid_in,
               type(grid_type), intent(inout) grid_out,
               integer, dimension(2), intent(inout), optional norm_id )
```

Trim a grid, removing any overlap between the different regions.

by default, the routine assumes a symmetric ghost region and cuts as many grid points from the end of the previous process as from the beginning of the next process, but if the number of overlapping grid points is odd, the next process looses one more point.

optionally, the trimmed indices in the normal direction can be provided in `norm_id`, i.e. the indices in the old, untrimmed grid that correspond to the start and end indices of the trimmed grid. E.g. if

- proc 0: 3 ... 25
- proc 1: 20 ... 50

then the trimmed grid will be:

- proc 0: 3 ... 22
- proc 1: 23 ... 50

which is shifted down by 2 to

- proc 0: 1 ... 20
- proc 1: 21 ... 48

in the trimmed grid. The indices of the previous step (3 & 22 and 23 & 50) are saved in `norm_id`.

Returns

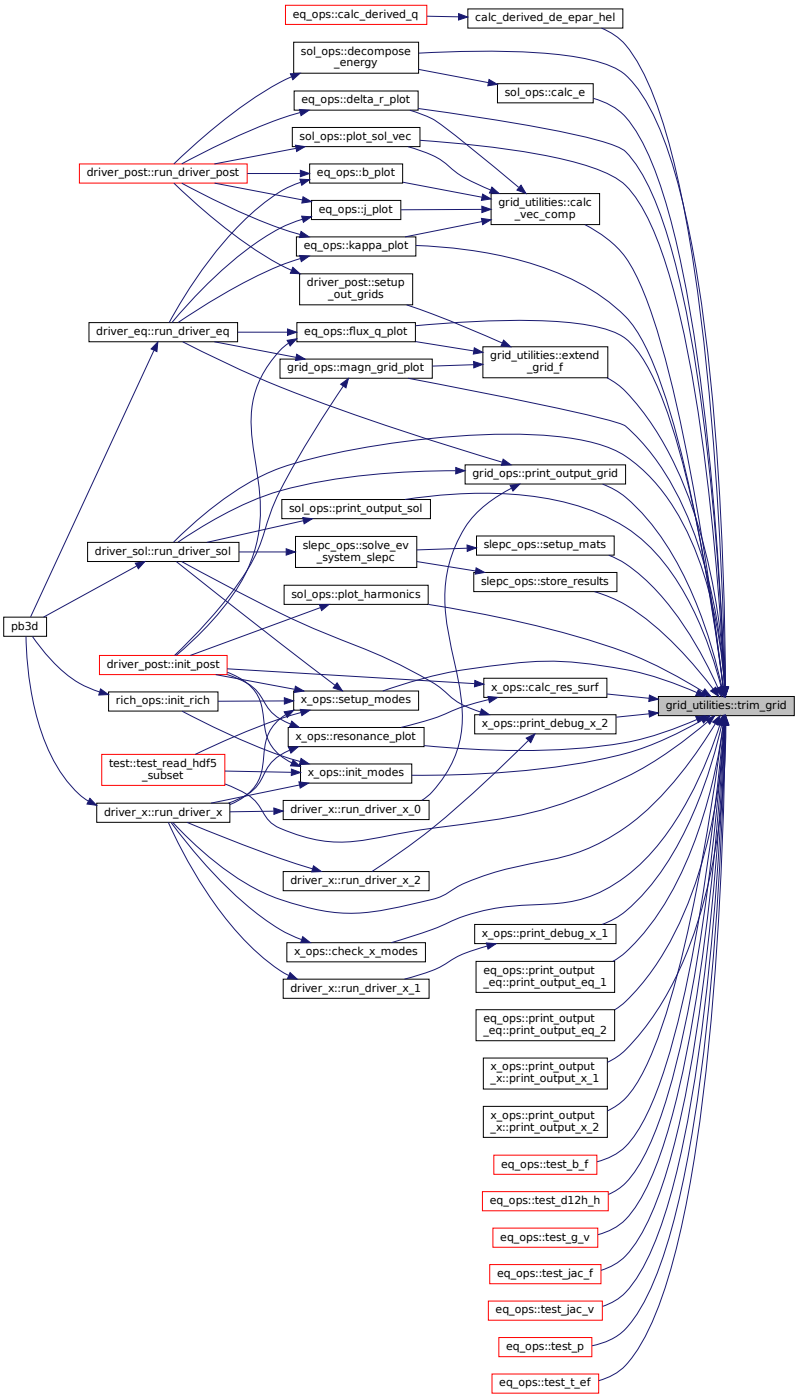
ierr

Parameters

`in`	*grid_in*	input grid
`in,out`	*grid_out*	trimmed grid
`in,out`	*norm↩ _id*	normal indices corresponding to trimmed part

Definition at line 1636 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.2.13 untrim_grid()

```
integer function, public grid_utilities::untrim_grid (
              type(grid_type), intent(in) grid_in,
```

```
type(grid_type), intent(inout) grid_out,
integer, intent(in) size_ghost )
```

Untrims a trimmed grid by introducing an asymmetric ghost regions at the right.

The width of the ghost region has to be provided.

Note

1. the ghosted grid should be deallocated (with dealloc_grid()).
2. the input grid has to be trimmed.

Returns

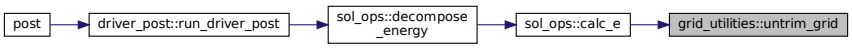
ierr

Parameters

in	*grid_in*	input grid
in,out	*grid_out*	ghosted grid
in	*size_ghost*	width of ghost region

Definition at line 1764 of file grid_utilities.f90.

Here is the caller graph for this function:



B.12.3 Variable Documentation

B.12.3.1 debug_calc_int_vol

```
logical, public grid_utilities::debug_calc_int_vol = .false.
```

plot debug information for calc_int_vol()

Note

Debug version only

Definition at line 25 of file grid_utilities.f90.

B.12.3.2 debug_calc_vec_comp

```
logical, public grid_utilities::debug_calc_vec_comp = .false.
```

plot debug information for calc_vec_comp()

Note

Debug version only

Definition at line 27 of file grid_utilities.f90.

B.13 grid_vars Module Reference

Variables pertaining to the different grids used.

Interfaces and Types

- type grid_type
  Type for grids.

Functions/Subroutines

- integer function init_grid (grid, n, i_lim, divided)
  Initializes a new grid.
- subroutine dealloc_grid (grid)
  Deallocates a grid.
- integer function copy_grid (grid_i, grid_o)
  Deep copy of a grid.

Variables

- integer, public n_r_in
  nr. of normal points in input grid
- integer, public n_r_eq
  nr. of normal points in equilibrium grid
- integer, public n_r_x
  nr. of normal points in perturbation grid
- integer, public n_r_sol
  nr. of normal points in solution grid
- integer, public n_alpha
  nr. of field-lines
- real(dp), public min_par_x
  min. of parallel coordinate [π] in field-aligned grid
- real(dp), public max_par_x

max. of parallel coordinate [π] in field-aligned grid

- real(dp), public min_alpha

  min. of field-line label [π] in field-aligned grid
- real(dp), public max_alpha

  max. of field-line label [π] in field-aligned grid
- real(dp), dimension(:), allocatable, public alpha

  field line label alpha
- integer, public n_alloc_grids

  nr. of allocated grids
- integer, public n_alloc_discs

  nr. of allocated discretizations

B.13.1 Detailed Description

Variables pertaining to the different grids used.

B.13.2 Function/Subroutine Documentation

B.13.2.1 copy_grid()

```
integer function grid_vars::copy_grid (
               class(grid_type), intent(in) grid_i,
               type(grid_type), intent(inout) grid_o )
```

Deep copy of a grid.

Note

Does not copy possible trigoniometric factors.

Returns

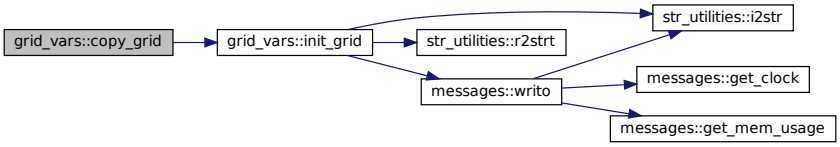
ierr

Parameters

in	*grid_i*	grid to be copied
in,out	*grid_o*	copied grid

Definition at line 267 of file grid_vars.f90.

Here is the call graph for this function:



B.13.2.2 dealloc_grid()

```
subroutine grid_vars::dealloc_grid (
               class(grid_type), intent(inout) grid )
```

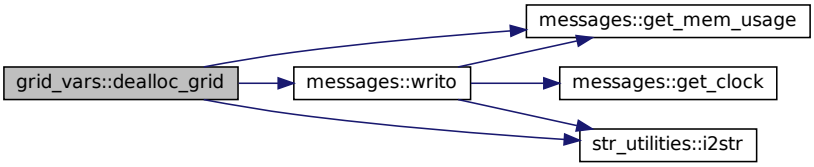
Deallocates a grid.

Parameters

in,out	*grid*	grid to be deallocated

Definition at line 203 of file grid_vars.f90.

Here is the call graph for this function:



B.13.2.3 init_grid()

```
integer function grid_vars::init_grid (
               class(grid_type), intent(inout) grid,
               integer, dimension(3), intent(in) n,
               integer, dimension(2), intent(in), optional i_lim,
               logical, intent(in), optional divided )
```

Initializes a new grid.

Optionally, the local limits can be provided for a divided grid.

Optionally, it can be set whether the grid is divided or not. A situation where this is useful is when only a subset of MPI processes is used to calculate the solution in SLEPC. In this case, the extra ranks all only contain the last grid point, and the last used process has as upper limit this same grid point. This way, all the procedures are reusable, but in this case if only one process is used, this procedure becomes confused and sets this process to undivided. In most cases, this functionality probably does not need to be used.

Returns

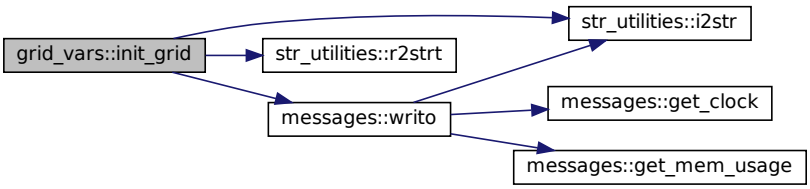
ierr

Parameters

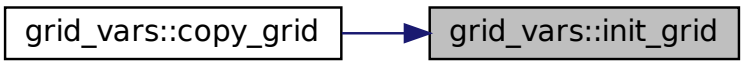
`in,out`	*grid*	grid to be initialized
`in`	*n*	tot. nr. of points (par,r,alpha)
`in`	*i_lim*	min. and max. local normal index
`in`	*divided*	divided grid or not

Definition at line 102 of file grid_vars.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.13.3 Variable Documentation

B.13.3.1 alpha

```
real(dp), dimension(:), allocatable, public grid_vars::alpha
```

field line label alpha

Definition at line 28 of file grid_vars.f90.

B.13.3.2 max_alpha

```
real(dp), public grid_vars::max_alpha
```

max. of field-line label [π] in field-aligned grid

Definition at line 27 of file grid_vars.f90.

B.13.3.3 max_par_x

```
real(dp), public grid_vars::max_par_x
```

max. of parallel coordinate [π] in field-aligned grid

Definition at line 25 of file grid_vars.f90.

B.13.3.4 min_alpha

```
real(dp), public grid_vars::min_alpha
```

min. of field-line label [π] in field-aligned grid

Definition at line 26 of file grid_vars.f90.

B.13.3.5 min_par_x

```
real(dp), public grid_vars::min_par_x
```

min. of parallel coordinate [π] in field-aligned grid

Definition at line 24 of file grid_vars.f90.

B.13.3.6 n_alloc_discs

```
integer, public grid_vars::n_alloc_discs
```

nr. of allocated discretizations

Note

Debug version only

Definition at line 31 of file grid_vars.f90.

B.13.3.7 n_alloc_grids

```
integer, public grid_vars::n_alloc_grids
```

nr. of allocated grids

Note

Debug version only

Definition at line 30 of file grid_vars.f90.

B.13.3.8 n_alpha

```
integer, public grid_vars::n_alpha
```

nr. of field-lines

Definition at line 23 of file grid_vars.f90.

B.13.3.9 n_r_eq

```
integer, public grid_vars::n_r_eq
```

nr. of normal points in equilibrium grid

Definition at line 20 of file grid_vars.f90.

B.13.3.10 n_r_in

```
integer, public grid_vars::n_r_in
```

nr. of normal points in input grid

Definition at line 19 of file grid_vars.f90.

B.13.3.11 n_r_sol

```
integer, public grid_vars::n_r_sol
```

nr. of normal points in solution grid

Definition at line 22 of file grid_vars.f90.

B.13.3.12 n_r_x

```
integer, public grid_vars::n_r_x
```

nr. of normal points in perturbation grid

Definition at line 21 of file grid_vars.f90.

B.14 hdf5_ops Module Reference

Operations on HDF5 and XDMF variables.

Interfaces and Types

- interface reset_hdf5_item

  Resets an HDF5 XDMF item.

Functions/Subroutines

- integer function, public open_hdf5_file (file_info, file_name, sym_type, descr, ind_plot, cont_plot)

  Opens an HDF5 file and accompanying xmf file and returns the handles.
- integer function, public close_hdf5_file (file_info, ind_plot, cont_plot)

  Closes an HDF5 file and writes the accompanying xmf file.
- integer function, public add_hdf5_item (file_info, XDMF_item, reset, ind_plot)

  Add an XDMF item to a HDF5 file.
- integer function, public print_hdf5_3d_data_item (dataitem_id, file_info, var_name, var, dim_tot, loc↩ _dim, loc_offset, init_val, ind_plot, cont_plot)

  Prints an HDF5 data item.
- subroutine, public merge_hdf5_3d_data_items (merged_id, dataitem_ids, var_name, dim_tot, reset, ind_plot, cont_plot)

  Joins dataitems in a vector.
- subroutine, public print_hdf5_att (att_id, att_dataitem, att_name, att_center, att_type, reset, ind_plot)

  Prints an HDF5 attribute.
- subroutine, public print_hdf5_top (top_id, top_type, top_n_elem, ind_plot)

  Prints an HDF5 topology.
- subroutine, public print_hdf5_geom (geom_id, geom_type, geom_dataitems, reset, ind_plot)

  Prints an HDF5 geometry.
- integer function, public print_hdf5_grid (grid_id, grid_name, grid_type, grid_time, grid_top, grid_geom, grid_atts, grid_grids, reset, ind_plot)

  Prints an HDF5 grid.
- integer function, public create_output_hdf5 (HDF5_name)

  Creates an HDF5 output file.
- integer function, public print_hdf5_arrs (vars, PB3D_name, head_name, rich_lvl, disp_info, ind_print, remove_previous_arrs)

  Prints a series of arrays, in the form of an array of pointers, to an HDF5 file.
- integer function, public read_hdf5_arr (var, PB3D_name, head_name, var_name, rich_lvl, disp_info, lim_loc)

  Reads a PB3D output file in HDF5 format.

Variables

- logical, public debug_hdf5_ops = .false.

  set to true to debug general information
- logical, public debug_print_hdf5_arrs = .false.

  set to true to debug print_HDF5_arrs

B.14.1 Detailed Description

Operations on HDF5 and XDMF variables.

Note

Notes about XDMF:

- Collections can be spatial or temporal. If a variable is to be evolved in time or if its domain is decomposed (e.g. the same physical variable is defined in multiple HDF5 variables), then the attribute name of this variable has to be the same for all the grids in the collection.
- If no collection is used, but just multiple grids, the attribute can be different but does not have to be, as the different grid are distinguished by their different grid names, not by the attribute of the variables they contain.
- The selection of hyperslabs is used, as described here: http://davis.lbl.gov/Manuals/HDF5-1.8.7/UG/12_Dataspaces.html
- Chunking is used for partial I/O, as described here: https://www.hdfgroup.org/HDF5/doc/Advanced/Chunking/ As no chunk cache is reused, the w0 setting should be 1. Furthermore, the number of slots is chosen to be equal to the number of chunks. And Finally, the chunk size is chosen to be the size of the previous dimensions, if it does not exceed 4GB
- Individual writes or plots are done now using the standard I/O driver, as using MPI does not allow for more than 2GB. This is important for the HELENA version of PB3D where `X_2` variables get larger than that. https://www.hdfgroup.org/HDF5/doc/UG/OldHtmlSource/UG_frame08TheFile.html
- Have a look in Documentation/XDMF_HDF5.pdf for an example.

B.14.2 Function/Subroutine Documentation

B.14.2.1 add_hdf5_item()

```
integer function, public hdf5_ops::add_hdf5_item (
               type(hdf5_file_type), intent(inout) file_info,
               type(xml_str_type), intent(inout) XDMF_item,
               logical, intent(in), optional reset,
               logical, intent(in), optional ind_plot )
```

Add an XDMF item to a HDF5 file.

Note

This should only be used with grids (or for topologies and geometries that are used throughout)

Returns

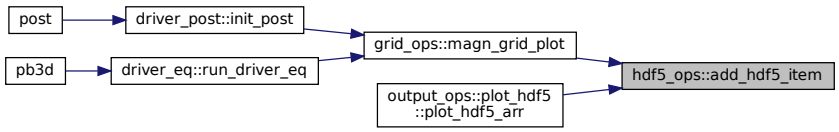
ierr

Parameters

in,out	*file_info*	info about HDF5 file
in,out	*xdmf_item*	XDMF item to add
in	*reset*	if .true., XDMF_item is reset
in	*ind_plot*	whether individual plot

Definition at line 340 of file HDF5_ops.f90.

Here is the caller graph for this function:



B.14.2.2 close_hdf5_file()

```
integer function, public hdf5_ops::close_hdf5_file (
               type(hdf5_file_type), intent(inout) file_info,
               logical, intent(in), optional ind_plot,
               logical, intent(in), optional cont_plot )
```

Closes an HDF5 file and writes the accompanying xmf file.

Returns

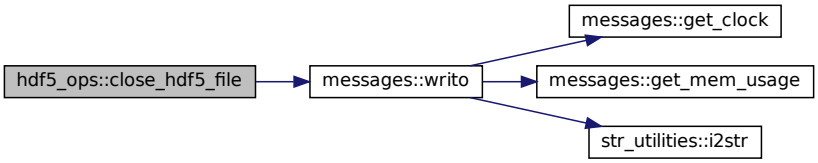
ierr

Parameters

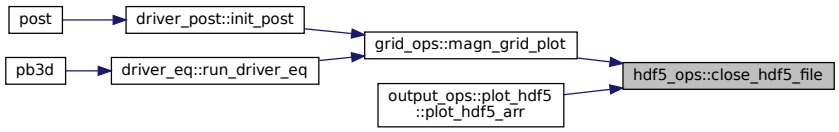
in,out	*file_info*	info about HDF5 file
in	*ind_plot*	whether individual plot
in	*cont_plot*	continued plot

Definition at line 264 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.3 create_output_hdf5()

```
integer function, public hdf5_ops::create_output_hdf5 (
               character(len=*), intent(in) HDF5_name )
```

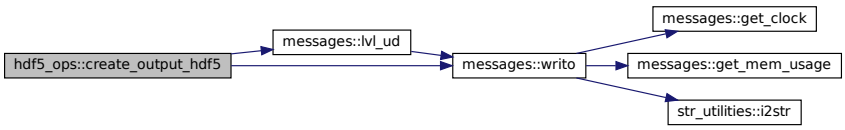
Creates an HDF5 output file.

Parameters

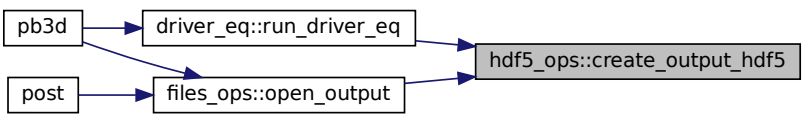
in	*hdf5_name*	name of HDF5 file

Definition at line 1075 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.4 merge_hdf5_3d_data_items()

```
subroutine, public hdf5_ops::merge_hdf5_3d_data_items (
                type(xml_str_type), intent(inout) merged_id,
                type(xml_str_type), dimension(:), intent(inout) dataitem_ids,
                character(len=*), intent(in) var_name,
                integer, dimension(3), intent(in) dim_tot,
                logical, intent(in), optional reset,
                logical, intent(in), optional ind_plot,
                logical, intent(in), optional cont_plot )
```

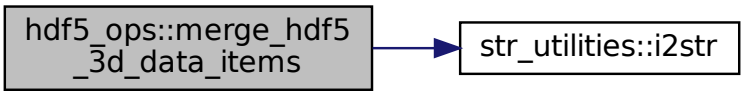
Joins dataitems in a vector.

Parameters

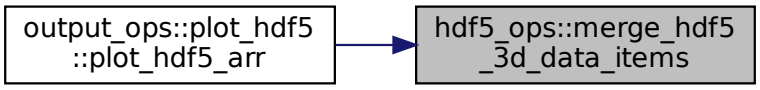
in,out	*merged_id*	ID of merged data item
in,out	*dataitem_ids*	ID of data items
in	*var_name*	name of variable
in	*dim_tot*	total dimensions of variable
in	*reset*	whether to reset dataitems
in	*ind_plot*	whether individual plot
in	*cont_plot*	continued plot

Definition at line 603 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.5 open_hdf5_file()

```
integer function, public hdf5_ops::open_hdf5_file (
               type(hdf5_file_type), intent(inout) file_info,
               character(len=*), intent(in) file_name,
               integer, intent(inout), optional sym_type,
               character(len=*), intent(in), optional descr,
               logical, intent(in), optional ind_plot,
               logical, intent(in), optional cont_plot )
```

Opens an HDF5 file and accompanying xmf file and returns the handles.

Optionally, a description of the file can be provided.

Also, the plot can be done for only one process, setting the variable `ind_plot`.

Furthermore, if the plot is a continuation, using `cont_plot`, the previous plot is opened and `sym_type` is returned.

Returns

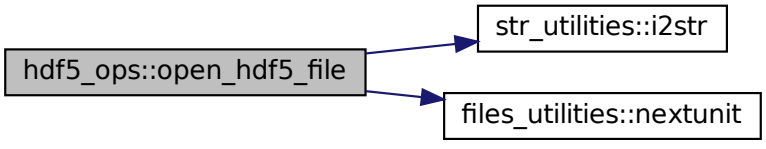
ierr

Parameters

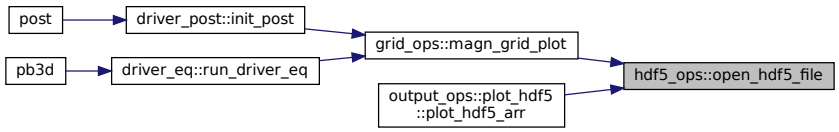
`in,out`	*file_info*	info about HDF5 file
`in`	*file_name*	name of HDF5 file
`in,out`	*sym_type*	symmetry type
`in`	*descr*	description of file
`in`	*ind_plot*	whether individual plot
`in`	*cont_plot*	continued plot

Definition at line 78 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.6 print_hdf5_3d_data_item()

```
integer function, public hdf5_ops::print_hdf5_3d_data_item (
               type(xml_str_type), intent(inout) dataitem_id,
               type(hdf5_file_type), intent(in) file_info,
               character(len=*), intent(in) var_name,
               real(dp), dimension(:,:,:), intent(in) var,
               integer, dimension(3), intent(in) dim_tot,
               integer, dimension(3), intent(in), optional loc_dim,
               integer, dimension(3), intent(in), optional loc_offset,
               real(dp), intent(in), optional init_val,
               logical, intent(in), optional ind_plot,
               logical, intent(in), optional cont_plot )
```

Prints an HDF5 data item.

If this is a parallel data item, the group dimension and offset have to be specified as well.

Returns

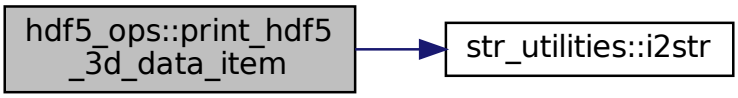
ierr

Parameters

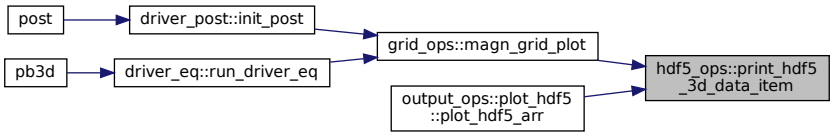
in,out	*dataitem_id*	ID of data item
in	*file_info*	info about HDF5 file
in	*var_name*	name of variable
in	*var*	variable to write
in	*dim_tot*	total dimensions of variable
in	*loc_dim*	dimensions in this group
in	*loc_offset*	offset in this group
in	*init_val*	initial fill value
in	*ind_plot*	whether individual plot
in	*cont_plot*	continued plot

Definition at line 396 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.7 print_hdf5_arrs()

```
integer function, public hdf5_ops::print_hdf5_arrs (
               type(var_1d_type), dimension(:), intent(in) vars,
               character(len=*), intent(in) PB3D_name,
               character(len=*), intent(in) head_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional disp_info,
               logical, intent(in), optional ind_print,
               logical, intent(in), optional remove_previous_arrs )
```

Prints a series of arrays, in the form of an array of pointers, to an HDF5 file.

Optionally, output can be given about the variable being written. Also, if `rich_lvl` is provided, `_R_[rich↩ _lvl]` is appended to the head name if it is > 0.

Also optionally, previously encountered arrays can be removed. This is to be used with care, as it may disturb the internal workings of PB3D. Currently (v2.15) it is used only for the specific case of jumping to solutions for `X_grid_style` 1 or 3, when writing solutions and solution grids.

Note

See `https://www.hdfgroup.org/HDF5/doc/UG/12_Dataspaces.html`, 7.4.2.3 for an explanation of the selection of the dataspaces.

Returns

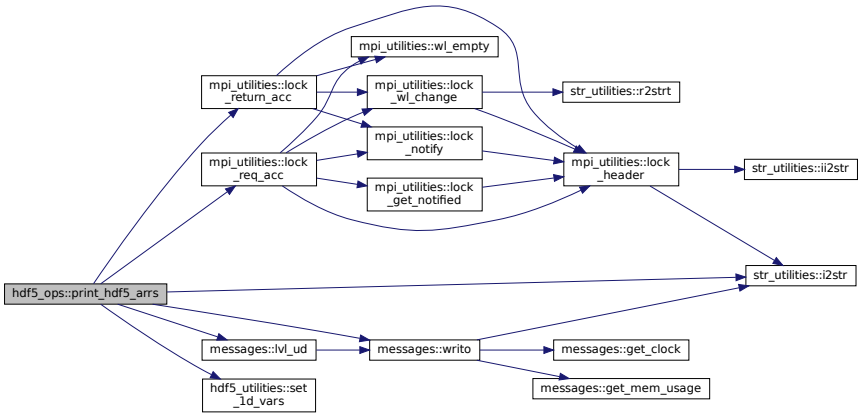
ierr

Parameters

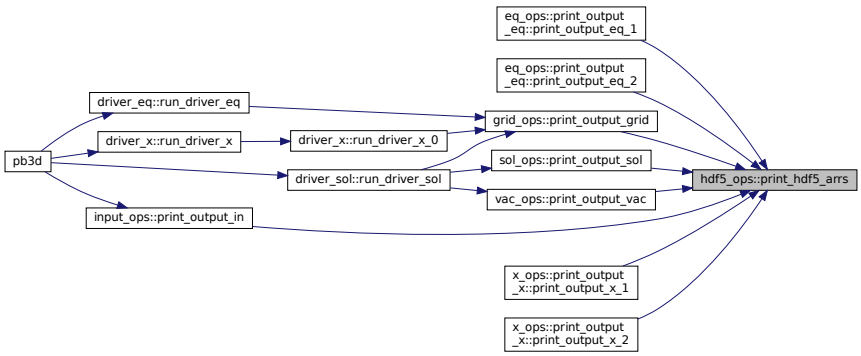
in	*vars*	variables to write
in	*pb3d_name*	name of PB3D file
in	*head_name*	head name of variables
in	*rich_lvl*	Richardson level to append to file name
in	*disp_info*	display additional information about variable being read
in	*ind_print*	individual write (possibly partial I/O)
in	*remove_previous_arrs*	remove previous variables if present

Definition at line 1132 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.8 print_hdf5_att()

```
subroutine, public hdf5_ops::print_hdf5_att (
               type(xml_str_type), intent(inout) att_id,
               type(xml_str_type), intent(inout) att_dataitem,
               character(len=*), intent(in) att_name,
               integer, intent(in) att_center,
               integer, intent(in) att_type,
               logical, intent(in), optional reset,
               logical, intent(in), optional ind_plot )
```

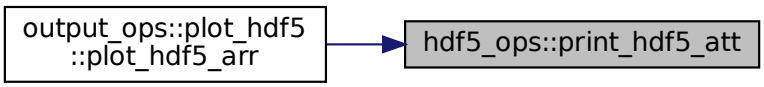
Prints an HDF5 attribute.

Parameters

in,out	*att_id*	ID of attribute
in,out	*att_dataitem*	dataitem of attribute
in	*att_name*	name of attribute
in	*att_center*	center of attribute
in	*att_type*	type of attribute
in	*reset*	whether to reset dataitems
in	*ind_plot*	whether individual plot

Definition at line 696 of file HDF5_ops.f90.

Here is the caller graph for this function:



B.14.2.9 print_hdf5_geom()

```
subroutine, public hdf5_ops::print_hdf5_geom (
               type(xml_str_type), intent(inout) geom_id,
               integer, intent(in) geom_type,
               type(xml_str_type), dimension(:), intent(inout) geom_dataitems,
               logical, intent(in), optional reset,
               logical, intent(in), optional ind_plot )
```

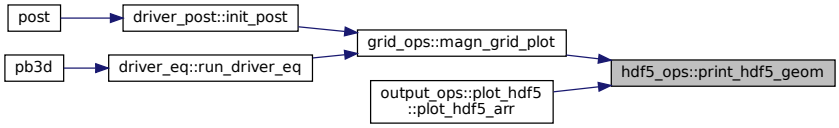
Prints an HDF5 geometry.

Parameters

in,out	*geom_id*	ID of geometry
in	*geom_type*	type of geometry
in,out	*geom_dataitems*	data items of geometry
in	*reset*	whether to reset dataitems
in	*ind_plot*	whether individual plot

Definition at line 805 of file HDF5_ops.f90.

Here is the caller graph for this function:



B.14.2.10 print_hdf5_grid()

```
integer function, public hdf5_ops::print_hdf5_grid (
               type(xml_str_type), intent(inout) grid_id,
               character(len=*), intent(in) grid_name,
               integer, intent(in) grid_type,
               real(dp), intent(in), optional grid_time,
               type(xml_str_type), optional grid_top,
               type(xml_str_type), optional grid_geom,
               type(xml_str_type), dimension(:), optional grid_atts,
               type(xml_str_type), dimension(:), optional grid_grids,
               logical, intent(in), optional reset,
               logical, intent(in), optional ind_plot )
```

Prints an HDF5 grid.

If this is is a uniform grid, the geometry and topology have to be specified, or else it will be assumed that it is already present in the XDMF domain, and reused.

If the grid is a collection grid, the grids in the collection have to be specified.

Optionally, also a time can be specified (for the grids in a collection grid).

Returns

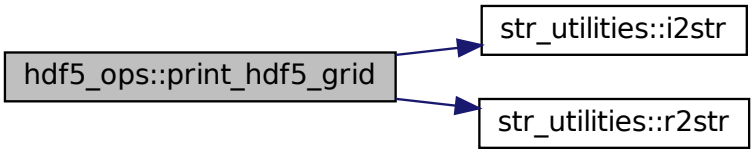
ierr

Parameters

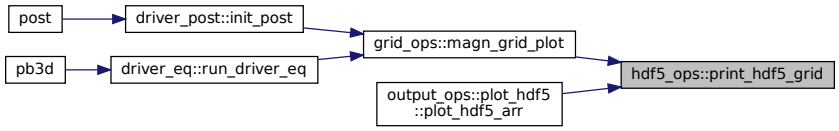
in,out	*grid_id*	ID of grid
in	*grid_name*	name
in	*grid_type*	type
in	*grid_time*	time of grid
	grid_top	topology
	grid_geom	geometry
	grid_atts	attributes
	grid_grids	grids
in	*reset*	whether top, geom and atts and grids are reset
in	*ind_plot*	whether individual plot

Definition at line 886 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.11 print_hdf5_top()

```
subroutine, public hdf5_ops::print_hdf5_top (
               type(xml_str_type), intent(inout) top_id,
               integer, intent(in) top_type,
               integer, dimension(:), intent(in) top_n_elem,
               logical, intent(in), optional ind_plot )
```

Prints an HDF5 topology.

Note

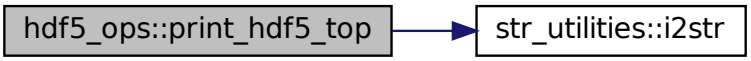
Currently only structured grids supported.

Parameters

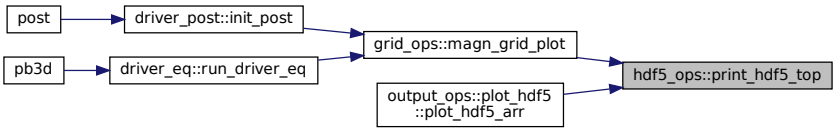
in,out	*top_id*	ID of topology
in	*top_type*	type
in	*top_n_elem*	nr. of elements
in	*ind_plot*	whether individual plot

Definition at line 755 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.2.12 read_hdf5_arr()

```
integer function, public hdf5_ops::read_hdf5_arr (
               type(var_1d_type), intent(inout) var,
               character(len=*), intent(in) PB3D_name,
               character(len=*), intent(in) head_name,
               character(len=*), intent(in) var_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional disp_info,
               integer, dimension(:,:), intent(in), optional lim_loc )
```

Reads a PB3D output file in HDF5 format.

This happens in a non-parallel way. By default, all variables are read, but an array of strings with acceptable variable names can be passed.

Optionally, output can be given about the variable being read. Also, if `rich_lvl` is provided, `_R_rich_lvl` is appended to the head name if it is > 0.

Furthermore, using `lim_loc`, a hyperslab of the variable can be read.

Note

If a limit of lim_loc is a negative value, the procedure just takes the entire range. This is necessary as sometimes the calling procuderes don't have, and don't need to have, knowledge of the underlying sizes, for example in the case of having multiple parallel jobs.

Returns

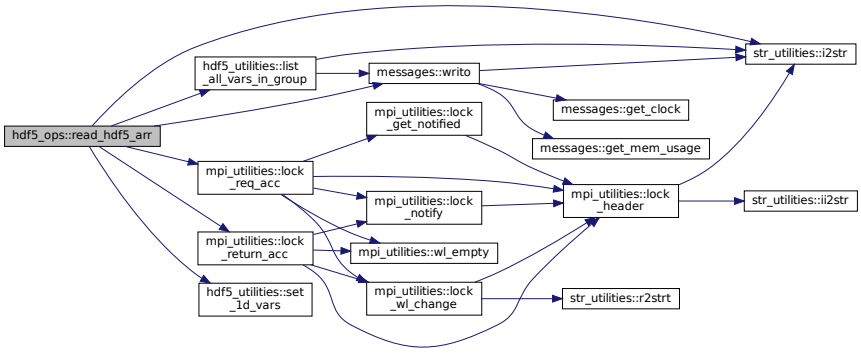
ierr

Parameters

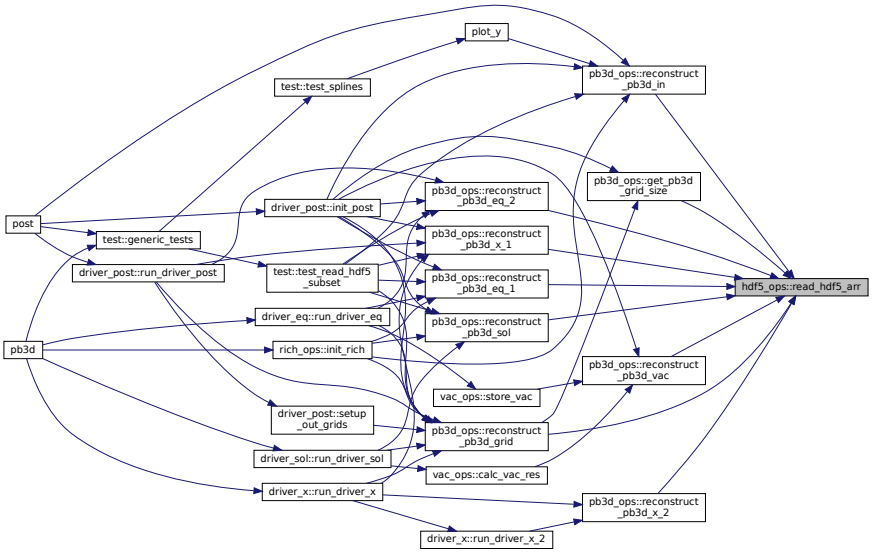
in,out	*var*	variable to read
in	*pb3d_name*	name of PB3D file
in	*head_name*	head name of variables
in	*var_name*	name of variable
in	*rich_lvl*	Richardson level to reconstruct
in	*disp_info*	display additional information about variable being read
in	*lim_loc*	local limits

Definition at line 1530 of file HDF5_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.14.3 Variable Documentation

B.14.3.1 debug_hdf5_ops

```
logical, public hdf5_ops::debug_hdf5_ops = .false.
```

set to true to debug general information

Note

Debug version only

Definition at line 47 of file HDF5_ops.f90.

B.14.3.2 debug_print_hdf5_arrs

```
logical, public hdf5_ops::debug_print_hdf5_arrs = .false.
```

set to true to debug print_HDF5_arrs

Note

Debug version only

Definition at line 48 of file HDF5_ops.f90.

B.15 hdf5_utilities Module Reference

Utilities pertaining to HDF5 and XDMF.

Functions/Subroutines

- integer function, public set_1d_vars (lim_tot, lim_loc, space_id, c_plist_id)

  Sets up the chunk property list and/or the 1D hyperslabs that correspond to a local subset of `lim_tot` given by the limits `lim_loc`.
- integer function, public probe_hdf5_group (HDF5_name, group_name, group_exists)

  Probe HDF5 file for group existence.
- integer function, public list_all_vars_in_group (group_id)

  Lists all variables in a HDF5 group.

Variables

- logical, public debug_set_1d_vars = .false.

  set to true to debug set_1D_vars

B.15.1 Detailed Description

Utilities pertaining to HDF5 and XDMF.

See also

See hdf5_ops for notes on HDF5 and XDMF.

B.15.2 Function/Subroutine Documentation

B.15.2.1 list_all_vars_in_group()

```
integer function, public hdf5_utilities::list_all_vars_in_group (
              integer(hid_t), intent(in) group_id )
```

Lists all variables in a HDF5 group.

Note

Debug version only

Returns

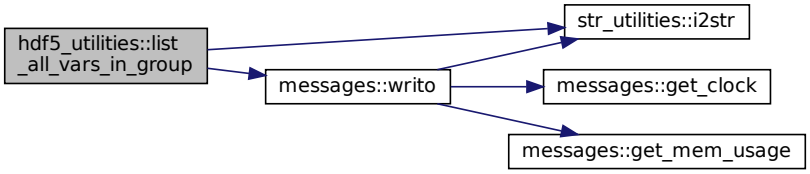
ierr

Parameters

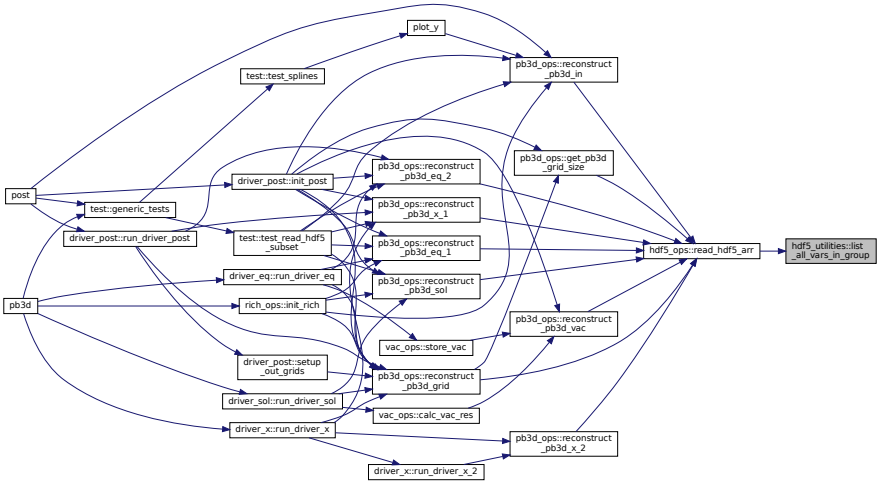
in	*group↩ _id*	group identifier

Definition at line 319 of file HDF5_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.15.2.2 probe_hdf5_group()

```
integer function, public hdf5_utilities::probe_hdf5_group (
              character(len=*), intent(in) HDF5_name,
              character(len=*), intent(in) group_name,
              logical, intent(inout) group_exists )
```

Probe HDF5 file for group existence.

Returns

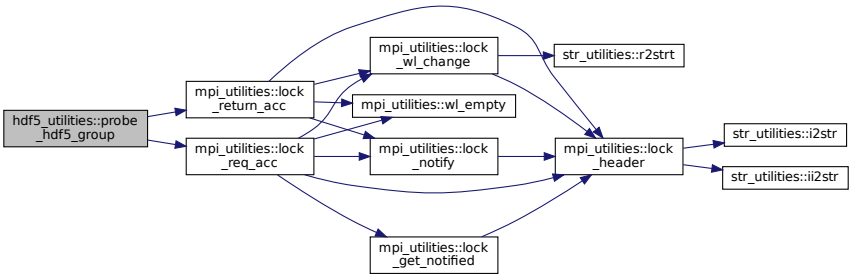
ierr

Parameters

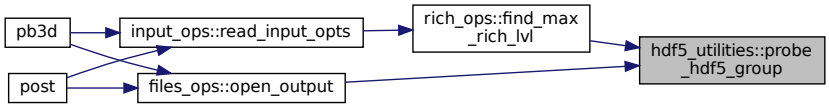
in	*hdf5_name*	name of HDF5 file
in	*group_name*	name of group to probe for
in,out	*group_exists*	whether group exists

Definition at line 251 of file HDF5_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.15.2.3 set_1d_vars()

```
integer function, public hdf5_utilities::set_1d_vars (
               integer, dimension(:,:), intent(in) lim_tot,
               integer, dimension(:,:), intent(in) lim_loc,
               integer(hid_t), intent(in), optional space_id,
               integer(hid_t), intent(inout), optional c_plist_id )
```

Sets up the chunk property list and/or the 1D hyperslabs that correspond to a local subset of `lim_tot` given by the limits `lim_loc`.

An example for the hyperslab is given in 3 dimensions with size n = 5,3,3:

- for dim = 1, limits: 2..4

- for dim = 2, limits: 3..3
- for dim = 3, limits: 2..3.

The 1D equivalent can be represented as
```
[.....][.....][.....] | [.....][.....][.....] | [.....][.....][.....]
```

Now, dimension 1 will only allow for the following elements (given by -):
```
[.---.][.---.][.---.] | [.---.][.---.][.---.] | [.---.][.---.][.---.]
```

dimension 2 will only allow for the following elements:
```
[.....][.....][-----] | [.....][.....][-----] | [.....][.....][-----]
```

dimension 3 will only allow for the following elements:
```
[.....][.....][.....] | [-----][-----][-----] | [-----][-----][-----]
```

so that the total is given by:
```
[.....][.....][.....] | [.....][.....][.....] | [.....][.....][.---.]
```

In practice, it is easiest to work with a full selection, where for every dimension the ranges that do not correspond to the range of that dimension are taken away. Clearly, if a dimension has the full range, nothing will be taken out:

- `block_i` = $n_1 n_2 \cdots n_{i-1}(b_i - a_{i+1})$,
- `stride_i` = $n_1 n_2 \cdots n_{i-1}(B_i - A_{i+1})$,
- `offset_i` = $n_1 n_2 \cdots n_{i-1}(a_i - A_i)$,
- `count_i` = $n_{i+1} n_{i+2} \cdots n_N$,

where a_i and b_i represent the local limits of dimension i, A_i and B_i the total ones and the number of dimensions is N, as can be verified.

The chunk property list for creation can be set up so that its size is equal to the largest dimensions of full range, or an integer part of that if it exceeds 4GB. Since the variables don't need to be used more than once, either `nbytes` or `nslots` could be set to 0 in an access property list, but this is currently not done.

Returns

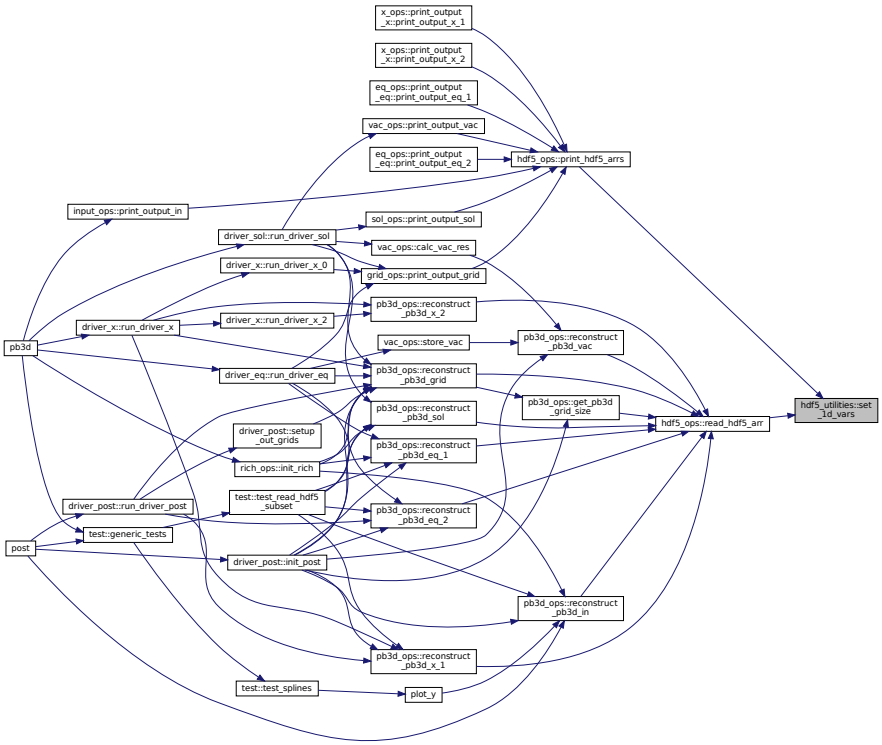
ierr

Parameters

`in`	*lim_tot*	total limits
`in`	*lim_loc*	local limits
`in`	*space_id*	dataspace identifier
`in,out`	*c_plist↩ _id*	chunk creation property list identifier

Definition at line 81 of file HDF5_utilities.f90.

Here is the caller graph for this function:



B.15.3 Variable Documentation

B.15.3.1 debug_set_1d_vars

```
logical, public hdf5_utilities::debug_set_1d_vars = .false.
```

set to true to debug set_1D_vars

Note

Debug version only

Definition at line 24 of file HDF5_utilities.f90.

B.16 hdf5_vars Module Reference

Variables pertaining to HDF5 and XDMF.

Interfaces and Types

- interface dealloc_var_1d
  Deallocates 1D variable.
- interface dealloc_xml_str
  Deallocates XML_str_type.
- type hdf5_file_type
  HDF5 data type, containing the information about HDF5 files.
- type var_1d_type
  1D equivalent of multidimensional variables, used for internal HDF5 storage.
- type xml_str_type
  XML strings used in XDMF.

Functions/Subroutines

- subroutine, public init_hdf5
  Initializes the HDF5 types.

Variables

- character(len=6), public xmf_fmt = '(999A)'
  format to write the xmf file
- integer, parameter, public max_dim_var_1d = 100000
  maximum dimension of var_1D
- character(len=max_str_ln), dimension(2), public xdmf_num_types
  possible XDMF number types
- character(len=max_str_ln), dimension(2), public xdmf_format_types
  possible XDMF format types
- character(len=max_str_ln), dimension(2), public xdmf_geom_types
  possible XDMF geometry types
- character(len=max_str_ln), dimension(2), public xdmf_top_types
  possible XDMF topology types
- character(len=max_str_ln), dimension(2), public xdmf_att_types
  possible XDMF attribute types
- character(len=max_str_ln), dimension(2), public xdmf_center_types
  possible XDMF attribute center types
- character(len=max_str_ln), dimension(3), public xdmf_grid_types
  possible XDMF grid types

B.16.1 Detailed Description

Variables pertaining to HDF5 and XDMF.

B.16.2 Function/Subroutine Documentation

B.16.2.1 init_hdf5()

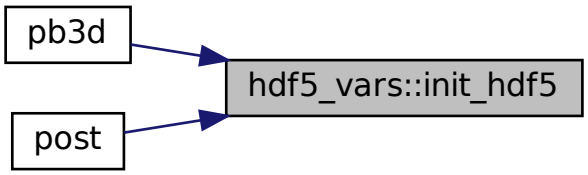
```
subroutine, public hdf5_vars::init_hdf5
```

Initializes the HDF5 types.

Has to be called once.

Definition at line 82 of file HDF5_vars.f90.

Here is the caller graph for this function:



B.16.3 Variable Documentation

B.16.3.1 max_dim_var_1d

```
integer, parameter, public hdf5_vars::max_dim_var_1d = 100000
```

maximum dimension of var_1D

Definition at line 21 of file HDF5_vars.f90.

B.16.3.2 xdmf_att_types

```
character(len=max_str_ln), dimension(2), public hdf5_vars::xdmf_att_types
```

possible XDMF attribute types

Definition at line 28 of file HDF5_vars.f90.

B.16.3.3 xdmf_center_types

```
character(len=max_str_ln), dimension(2), public hdf5_vars::xdmf_center_types
```

possible XDMF attribute center types

Definition at line 29 of file HDF5_vars.f90.

B.16.3.4 xdmf_format_types

```
character(len=max_str_ln), dimension(2), public hdf5_vars::xdmf_format_types
```

possible XDMF format types

Definition at line 25 of file HDF5_vars.f90.

B.16.3.5 xdmf_geom_types

```
character(len=max_str_ln), dimension(2), public hdf5_vars::xdmf_geom_types
```

possible XDMF geometry types

Definition at line 26 of file HDF5_vars.f90.

B.16.3.6 xdmf_grid_types

```
character(len=max_str_ln), dimension(3), public hdf5_vars::xdmf_grid_types
```

possible XDMF grid types

Definition at line 30 of file HDF5_vars.f90.

B.16.3.7 xdmf_num_types

```
character(len=max_str_ln), dimension(2), public hdf5_vars::xdmf_num_types
```

possible XDMF number types

Definition at line 24 of file HDF5_vars.f90.

B.16.3.8 xdmf_top_types

```
character(len=max_str_ln), dimension(2), public hdf5_vars::xdmf_top_types
```

possible XDMF topology types

Definition at line 27 of file HDF5_vars.f90.

B.16.3.9 xmf_fmt

```
character(len=6), public hdf5_vars::xmf_fmt = '(999A)'
```

format to write the xmf file

Definition at line 20 of file HDF5_vars.f90.

B.17 helena_ops Module Reference

Operations on HELENA variables.

Functions/Subroutines

- integer function, public read_hel (n_r_in, use_pol_flux_H)

  Reads the HELENA equilibrium data.
- integer function get_ang_interp_data_hel (grid_in, grid_out, theta_i, fund_theta_int_displ, tb_sym, use_E)

  Calculate interpolation factors for angular interpolation in `grid_out` of quantities defined on `grid_in`.
- integer function, public interp_hel_on_grid (grid_in, grid_out, eq_2, X_1, X_2, eq_2_out, X_1_out, X_↩ 2_out, eq_1, grid_name)

  Interpolate variables resulting from HELENA equilibria to another grid (angularly).
- integer function, public test_metrics_h ()

  Checks whether the metric elements provided by HELENA are consistent with a direct calculation using the coordinate transformations [15].
- integer function, public test_harm_cont_h ()

  Investaige harmonic content of the HELENA variables.

B.17.1 Detailed Description

Operations on HELENA variables.

B.17.2 Function/Subroutine Documentation

B.17.2.1 get_ang_interp_data_hel()

```
integer function helena_ops::get_ang_interp_data_hel (
                type(grid_type), intent(in) grid_in,
                type(grid_type), intent(in) grid_out,
                real(dp), dimension(:,:,:), intent(inout), allocatable theta_i,
                integer, dimension(:,:,:), intent(inout), allocatable fund_theta_int_displ,
                logical, intent(in), optional tb_sym,
                logical, intent(in), optional use_E )
```

Calculate interpolation factors for angular interpolation in `grid_out` of quantities defined on `grid_in`.

This version is specific for an input grid corresponding to axisymmetric variables with optional top-down symmetry, as is the case for variables resulting from HELENA equilibria.

The output of a 3-D array of real values for the poloidal angle θ where the floored integer of each value indicates the base index of the interpolated value in the output grid and the modulus is the the fraction towards the next integer.

The flag `tb_sym` indicates optionally that there is top-bottom symmetry as well as axisymmetry. When there is top-down symmetry, the variables in the lower half (i.e. $-\pi < \theta < 0$) are calculated from the variables in the upper half using the symmetry properties of the variables. To indicate this, the sign of the interpolation factor is inverted to a negative value.

The displacement of the theta interval towards the fundamental theta interval is also outputted. For asymmetric variables this is for example:

- 1 for $-2\pi \ldots 0$
- 0 for $0 \ldots 2\pi$
- -1 for $2\pi \ldots 4\pi$

etc.

For symmetric variables, this is for example:

- 1 for $-3\pi \ldots -2\pi$, with symmetry property
- 1 for $-2\pi \ldots -\pi$
- 0 for $-\pi \ldots 0$, with symmetry property
- 0 for $0 \ldots \pi$
- -1 for $\pi \ldots 2\pi$, with symmetry property
- -1 for $3\pi \ldots 3\pi$ etc.

By default, the variables in the Flux coord. system are used, but this can be changed optionally with the flag "`use_E`.

Returns

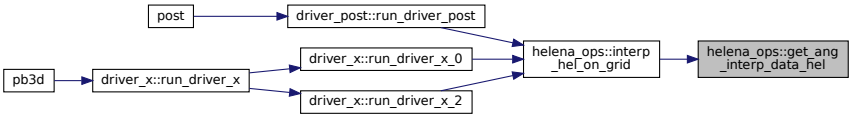
ierr

Parameters

in	*grid_in*	input grid
in	*grid_out*	output grid
in,out	*theta_i*	interpolation index
in,out	*fund_theta_int_displ*	displacement of fundamental theta interval
in	*tb_sym*	top-bottom symmetry
in	*use_e*	whether E is used instead of F

Definition at line 413 of file HELENA_ops.f90.

Here is the caller graph for this function:



B.17.2.2 interp_hel_on_grid()

```
integer function, public helena_ops::interp_hel_on_grid (
                type(grid_type), intent(in) grid_in,
                type(grid_type), intent(in) grid_out,
                type(eq_2_type), intent(inout), optional eq_2,
                type(x_1_type), intent(inout), optional X_1,
                type(x_2_type), intent(inout), optional X_2,
                type(eq_2_type), intent(inout), optional eq_2_out,
                type(x_1_type), intent(inout), optional X_1_out,
                type(x_2_type), intent(inout), optional X_2_out,
                type(eq_1_type), intent(in), optional eq_1,
                character(len=*), intent(in), optional grid_name )
```

Interpolate variables resulting from HELENA equilibria to another grid (angularly).

The input and output grid to be provided depend on the quantities to be interpolated:

- equilibrium variables: flux variables (no need to convert) and derived quantities (need equilibrium grid)
- metric variables: jac_FD (need equilibrium grid)
- vectorial perturbation variables: U_i, DU_i (need perturbation grid)
- tensorial perturbation variables: PV_i, KV_i (need perturbation grid)

Also, a message can be printed if a grid name is passed.

Note

1. The metric coefficients are interpolated and then compensated for the straight-field-line coordinates as in [15] .
2. By default the interpolated quantities overwrite the original ones, but alternative output variables can be provided.
3. as the equilibrium and perturbation grid are not generally identical, this routine has to be called separately for the variables tabulated in either grid.

Returns

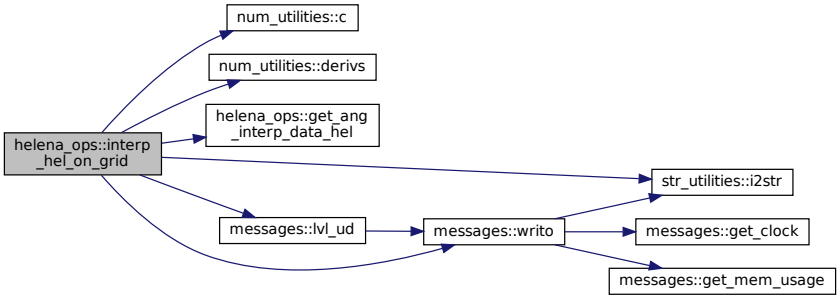
ierr

Parameters

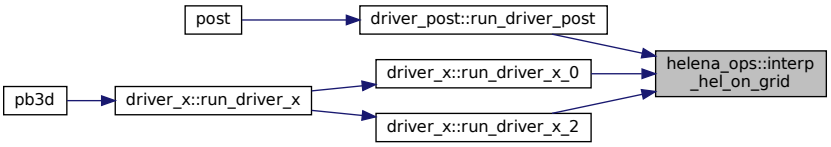
in	*grid_in*	input grid
in	*grid_out*	output grid
in,out	*eq_2*	general metric equilibrium variables
in,out	*x_1*	general vectorial perturbation variables
in,out	*x_2*	general tensorial perturbation variables
in,out	*eq_2_out*	field-aligned metric equilibrium variables
in,out	*x_1_out*	field-aligned vectorial perturbation variables
in,out	*x_2_out*	field-aligned tensorial perturbation variables
in	*eq_1*	general flux equilibrium variables for metric interpolation
in	*grid_name*	name of grid to which to adapt quantities

Definition at line 550 of file HELENA_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.17.2.3 read_hel()

```
integer function, public helena_ops::read_hel (
               integer, intent(inout) n_r_in,
               logical, intent(inout) use_pol_flux_H )
```

Reads the HELENA equilibrium data.

Adapted from HELENA routine `IODSK`.

The variales in the HELENA mapping file are globalized in two ways:

- X and Y are normalized w.r.t. vacuum geometric axis `R_vac` and toroidal field at the geometric axis `B_vac`.
- `R[m] = R_vac[m] (1 + eps X[])` ,
- `Z[m] = R_vac[m] eps Y[]` .

The covariant toroidal field `F_H`, `pres_H` and poloidal flux are normalized w.r.t magnetic axis `R_m` and total toroidal field at magnetic axis `B_m`:

- `RBphi[Tm] = F_H[] R_m[m] B_m[T]` ,
- `pres[N/m^2] = pres_H[] (B_m[T])^2/mu_0[N/A^2]` ,
- `flux_p[Tm^2] = 2pi (s[])^2 cpsurf[] B_m[T] (R_m[m])^2` .

The first normalization type is the HELENA normalization, whereas the second is the MISHKA normalization.

Everything is translated to MISHKA normalization to make comparison with MISHKA simple. This is done using the factors:

- `radius[] = a[m] / R_m[m]` ,
- `eps[] = a[m] / R_vac[m]` , so that the expressions become:
- `R[m] = radius[] (1/eps[] + X[]) R_m[m]`,

- Z[m] = radius[] Y[] R_m[m],
- RBphi[Tm] = F_H[] B_m[T] R_m[m],
- pres[N/m^2] = pres_H[] B_m[T]^2 mu_0[N/A^2]^-1,
- flux_p[Tm^2] = 2pi (s[])^2 cpsurf[] B_m[T] R_m[T]^2.

Finally, in HELENA, the total current I, the poloidal beta and the density at the geometric axis can be prescribed through:

- XIAB = mu_0 I / (a_vac B_vac) ,
- BETAP = 8 pi S <p> / (I^2 mu_0) ,
- ZN ,

where a_vac = eps R_vac and B_vac are vacuum quantities, S is the 2-D cross-sectional area and <p> is the 2-D averaged pressure.

Note

To translate this to the MISHKA normalization factors as

- R_m = (eps/radius) R_vac ,
- B_m = B_vac / B0 ,

where

- radius is in the mapping file (12), as well as in the HELENA output (20).
- eps is in the mapping file (12), as well as in the HELENA input (10) and output (20).
- B0 is in the mapping file (12), as well as in the HELENA output (20).

Furthermore, the density on axis can be specified as ZN0 from HELENA input (10).
The other variables should probably not be touched for consistency.

Finally, the variables IAS and B0 are in the mapping file (12) only in patched versions. See Getting the Equilibrium.

Returns

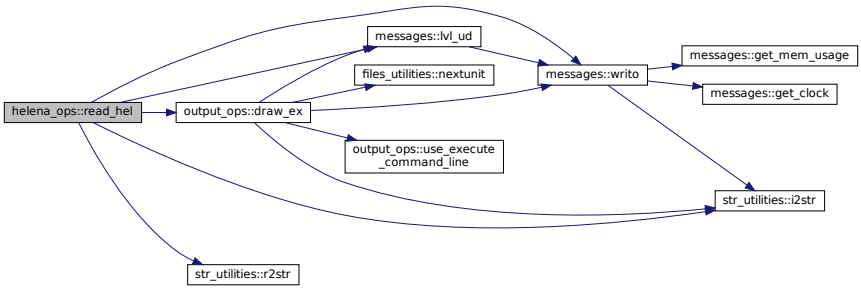
ierr

Parameters

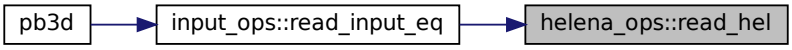
in,out	*n_r_in*	nr. of normal points in input grid
in,out	*use_pol_↩ flux_h*	.true. if HELENA equilibrium is based on pol. flux

Definition at line 84 of file HELENA_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.17.2.4 test_harm_cont_h()

```
integer function, public helena_ops::test_harm_cont_h
```

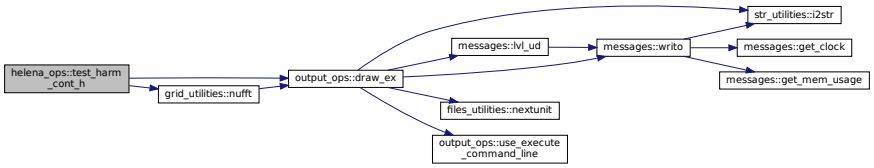
Investaige harmonic content of the HELENA variables.

Note

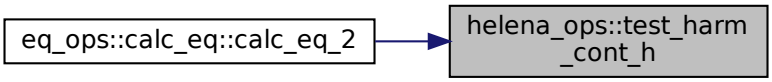
Run with one process.

Definition at line 1485 of file HELENA_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.17.2.5 test_metrics_h()

```
integer function, public helena_ops::test_metrics_h
```

Checks whether the metric elements provided by HELENA are consistent with a direct calculation using the coordinate transformations [15].

Direct calculations used:

$$|\nabla\psi|^2 = \frac{1}{\mathcal{J}^2}\left(\left(\frac{\partial Z}{\partial \chi}\right)^2 + \left(\frac{\partial R}{\partial \chi}\right)^2\right)$$
$$|\nabla\psi \cdot \nabla\chi| = \frac{1}{\mathcal{J}^2}\left(\frac{\partial Z}{\partial \chi}\frac{\partial Z}{\partial \psi} - \frac{\partial R}{\partial \chi}\frac{\partial R}{\partial \psi}\right)$$
$$|\nabla\chi|^2 = \frac{1}{\mathcal{J}^2}\left(\left(\frac{\partial Z}{\partial \psi}\right)^2 + \left(\frac{\partial R}{\partial \psi}\right)^2\right)$$
$$|\nabla\phi|^2 = \frac{1}{R^2}$$

with

$$\mathcal{J} = \frac{\partial Z}{\partial \psi}\frac{\partial R}{\partial \chi} - \frac{\partial R}{\partial \psi}\frac{\partial Z}{\partial \chi}$$

Also, test whether the pressure balance $\nabla p = \vec{J} \times \vec{B}$ is satisfied.

Note

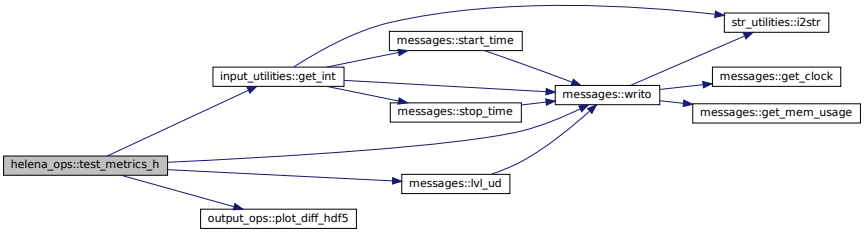
Debug version only

Returns

ierr

Definition at line 1289 of file HELENA_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.18 helena_vars Module Reference

Variables that have to do with HELENA quantities.

Functions/Subroutines

- subroutine, public dealloc_hel

  Deallocates HELENA quantities that are not used any more.

Variables

- real(dp), public rmtog_h

  R_geo/R_mag.
- real(dp), public bmtog_h

  B_geo/B_mag.
- real(dp), dimension(:), allocatable, public chi_h

  poloidal angle
- real(dp), dimension(:,:), allocatable, public flux_p_h

  poloidal flux

- real(dp), dimension(:,:), allocatable, public flux_t_h
  toroidal flux
- real(dp), dimension(:,:), allocatable, public pres_h
  pressure profile
- real(dp), dimension(:,:), allocatable, public q_saf_h
  safety factor
- real(dp), dimension(:,:), allocatable, public rot_t_h
  rotational transform
- real(dp), dimension(:,:), allocatable, public rbphi_h
  $RB_\phi (= F)$
- real(dp), dimension(:,:), allocatable, public h_h_11
  upper metric factor h_{11} *(gem11)*
- real(dp), dimension(:,:), allocatable, public h_h_12
  upper metric factor h_{12} *(gem12)*
- real(dp), dimension(:,:), allocatable, public h_h_33
  upper metric factor h_{32} *(1 / gem12)*
- real(dp), dimension(:,:), allocatable, public r_h
  major radius R *(xout)*
- real(dp), dimension(:,:), allocatable, public z_h
  height Z *(yout)*
- integer, public nchi
  nr. of poloidal points
- integer, public ias
  0 if top-bottom symmetric, 1 if not

B.18.1 Detailed Description

Variables that have to do with HELENA quantities.

B.18.2 Function/Subroutine Documentation

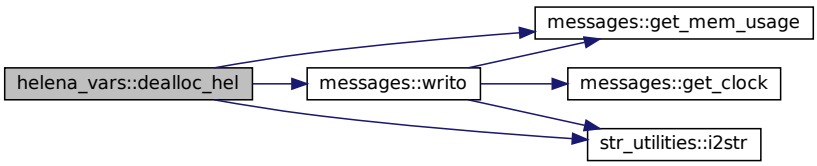
B.18.2.1 dealloc_hel()

```
subroutine, public helena_vars::dealloc_hel
```

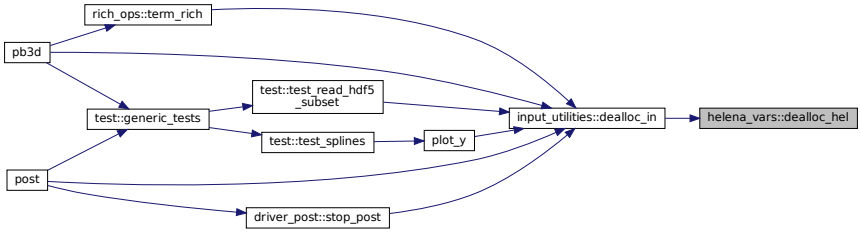
Deallocates HELENA quantities that are not used any more.

Definition at line 41 of file HELENA_vars.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.18.3 Variable Documentation

B.18.3.1 bmtog_h

```
real(dp), public helena_vars::bmtog_h
```

B_geo/B_mag.

Definition at line 22 of file HELENA_vars.f90.

B.18.3.2 chi_h

```
real(dp), dimension(:), allocatable, public helena_vars::chi_h
```

poloidal angle

Definition at line 23 of file HELENA_vars.f90.

B.18.3.3 flux_p_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::flux_p_h
```

poloidal flux

Definition at line 24 of file HELENA_vars.f90.

B.18.3.4 flux_t_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::flux_t_h
```

toroidal flux

Definition at line 25 of file HELENA_vars.f90.

B.18.3.5 h_h_11

```
real(dp), dimension(:,:), allocatable, public helena_vars::h_h_11
```

upper metric factor h_{11} (gem11)

Definition at line 30 of file HELENA_vars.f90.

B.18.3.6 h_h_12

```
real(dp), dimension(:,:), allocatable, public helena_vars::h_h_12
```

upper metric factor h_{12} (gem12)

Definition at line 31 of file HELENA_vars.f90.

B.18.3.7 h_h_33

```
real(dp), dimension(:,:), allocatable, public helena_vars::h_h_33
```

upper metric factor h_{32} (1 / gem12)

Definition at line 32 of file HELENA_vars.f90.

B.18.3.8 ias

```
integer, public helena_vars::ias
```

0 if top-bottom symmetric, 1 if not

Definition at line 36 of file HELENA_vars.f90.

B.18.3.9 nchi

```
integer, public helena_vars::nchi
```

nr. of poloidal points

Definition at line 35 of file HELENA_vars.f90.

B.18.3.10 pres_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::pres_h
```

pressure profile

Definition at line 26 of file HELENA_vars.f90.

B.18.3.11 q_saf_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::q_saf_h
```

safety factor

Definition at line 27 of file HELENA_vars.f90.

B.18.3.12 r_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::r_h
```

major radius R (xout)

Definition at line 33 of file HELENA_vars.f90.

B.18.3.13 rbphi_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::rbphi_h
```

$RB_\phi (= F)$

Definition at line 29 of file HELENA_vars.f90.

B.18.3.14 rmtog_h

```
real(dp), public helena_vars::rmtog_h
```

R_geo/R_mag.

Definition at line 21 of file HELENA_vars.f90.

B.18.3.15 rot_t_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::rot_t_h
```

rotational transform

Definition at line 28 of file HELENA_vars.f90.

B.18.3.16 z_h

```
real(dp), dimension(:,:), allocatable, public helena_vars::z_h
```

height Z (yout)

Definition at line 34 of file HELENA_vars.f90.

B.19 input_ops Module Reference

Operations concerning giving input.

Functions/Subroutines

- integer function, public read_input_opts ()
  Reads input options from user-provided input file.
- integer function, public read_input_eq ()
  Reads the equilibrium input file if no Richardson restart.
- integer function, public print_output_in (data_name, remove_previous_arrs)
  Print input quantities to an output file.

B.19.1 Detailed Description

Operations concerning giving input.

B.19.2 Function/Subroutine Documentation

B.19.2.1 print_output_in()

```
integer function, public input_ops::print_output_in (
              character(len=*), intent(in) data_name,
              logical, intent(in), optional remove_previous_arrs )
```

Print input quantities to an output file.

Variables printed:

- misc_in: prog_version, eq_style, rho_style, R_0, pres_0, B_0, psi_0 rho_0, T_0, vac_perm, use_↩ pol_flux_F, use_pol_flux_E, use_normalization, norm_disc_prec_eq, norm_disc_style_sol, n_r↩ _in, n_r_eq, n_r_sol, debug_version
- misc_in_V: is_asym_V, is_freeb_V, mnmax_V, mpol_V, ntor_V, gam_V
- flux_t_V
- flux_p_V
- pres_V
- rot_t_V
- q_saf_V
- mn_V
- R_V_c
- R_V_s
- Z_V_c
- Z_V_s
- L_V_c

- L_V_s
- B_V_sub: B_V_sub_c, B_V_sub_s
- B_V: B_V_c, B_V_s
- J_V_sup_int
- jac_V: jac_V_c, jac_V_s
- misc_in_H: ias, nchi
- RZ_H: R_H, Z_H
- chi_H
- q_saf_H
- rot_t_H
- RBphi_H
- pres_H
- flux_p_H
- flux_t_H
- misc_X: prim_X, n_mod_X, min_sec_X, max_sec_X, norm_disc_prec_X, norm_style, U_style, X_style, matrix_SLEPC_style, K_style, alpha_style
- misc_sol: min_r_sol, max_r_sol, norm_disc_prec_sol, BC_style, EV_BC, EV_BC

Returns

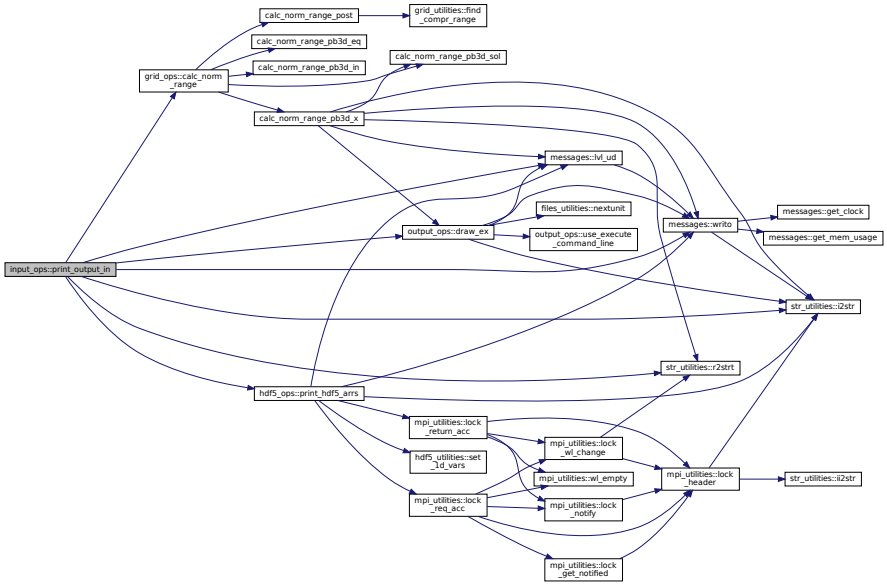
ierr

Parameters

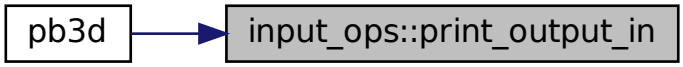
in	*data_name*	name under which to store
in	*remove_previous_arrs*	remove previous variables if present

Definition at line 1175 of file input_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.19.2.2 read_input_eq()

```
integer function, public input_ops::read_input_eq
```

Reads the equilibrium input file if no Richardson restart.

These variables will be passed on through the HDF5 data system.

See also

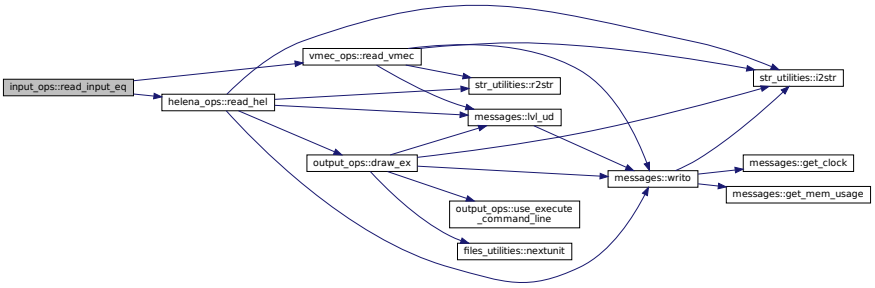
reconstruct_pb3d_in().

Returns

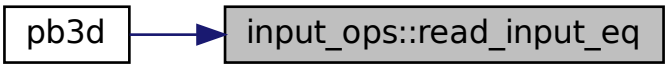
ierr

Definition at line 1104 of file input_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.19.2.3 read_input_opts()

```
integer function, public input_ops::read_input_opts
```

Reads input options from user-provided input file.

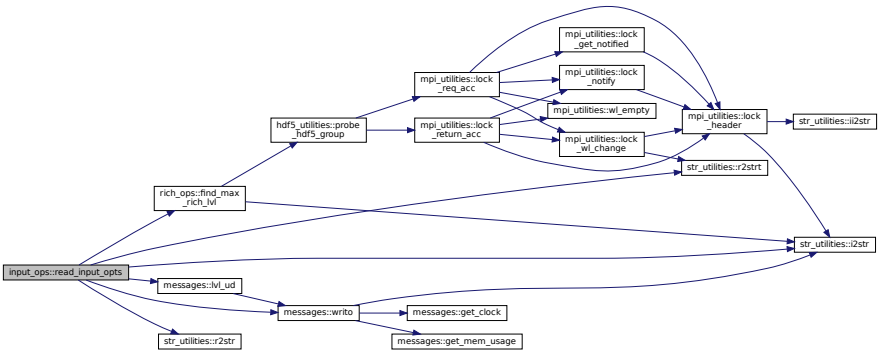
This is done by the master process and these variables will then be passed on through MPI to other processes in broadcast_input_opts().

Returns

ierr

Definition at line 23 of file input_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.20 input_utilities Module Reference

Numerical utilities related to input.

Functions/Subroutines

- logical function, public get_log (yes, ind)

  Queries for a logical value yes or no, where the default answer is also to be provided.
- real(dp) function, public get_real (lim_lo, lim_hi, ind)

  Queries for user input for a real value, where allowable range can be provided as well.
- integer function, public get_int (lim_lo, lim_hi, ind)

  Queries for user input for an integer value, where allowable range can be provided as well.
- subroutine, public pause_prog (ind)

  Pauses the running of the program.
- subroutine, public dealloc_in ()

  Cleans up input from equilibrium codes.

B.20.1 Detailed Description

Numerical utilities related to input.

B.20.2 Function/Subroutine Documentation

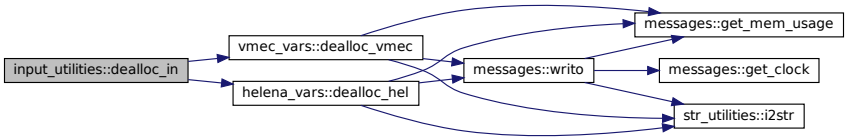
B.20.2.1 dealloc_in()

```
subroutine, public input_utilities::dealloc_in
```

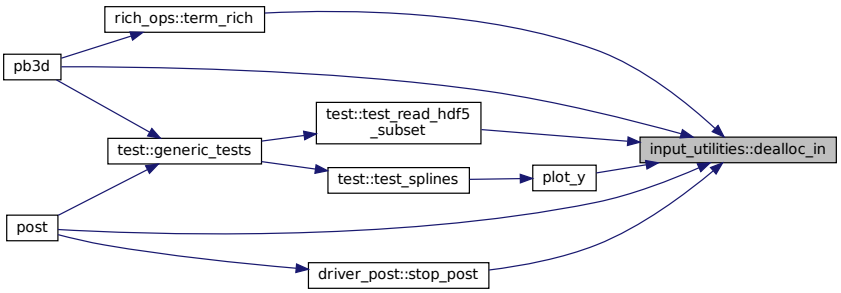
Cleans up input from equilibrium codes.

Definition at line 268 of file input_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.20.2.2 get_int()

```
integer function, public input_utilities::get_int (
               integer, intent(in), optional lim_lo,
               integer, intent(in), optional lim_hi,
               logical, intent(in), optional ind )
```

Queries for user input for an integer value, where allowable range can be provided as well.

If not called by all processes at the same time, `ind` should be set to indicate an individual operation.

Returns

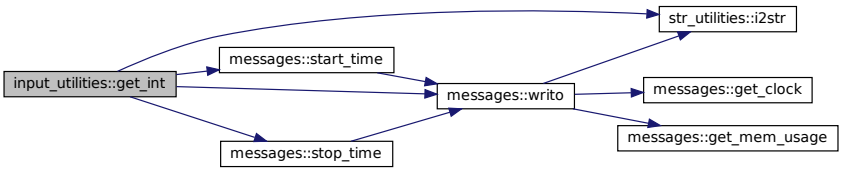
output value

Parameters

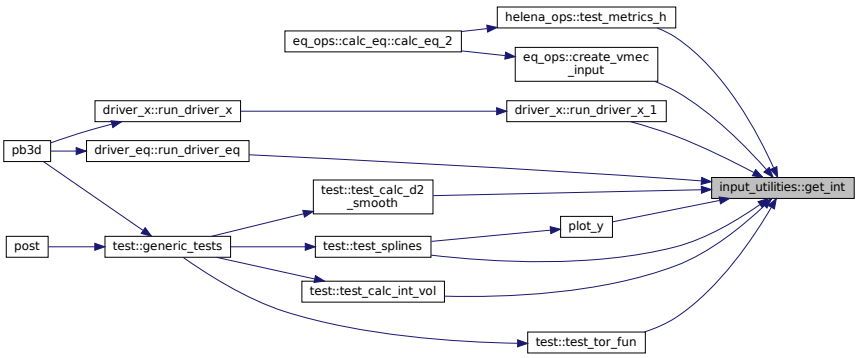
`in`	*lim_lo*	upper and lower limit
`in`	*lim_hi*	upper and lower limit
`in`	*ind*	individual pause or not

Definition at line 152 of file input_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.20.2.3 get_log()

```
logical function, public input_utilities::get_log (
               logical yes,
               logical, intent(in), optional ind )
```

Queries for a logical value yes or no, where the default answer is also to be provided.

If not called by all processes at the same time, `ind` should be set to indicate an individual operation.

Returns

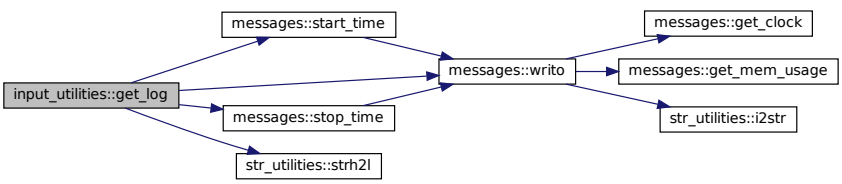
output value

Parameters

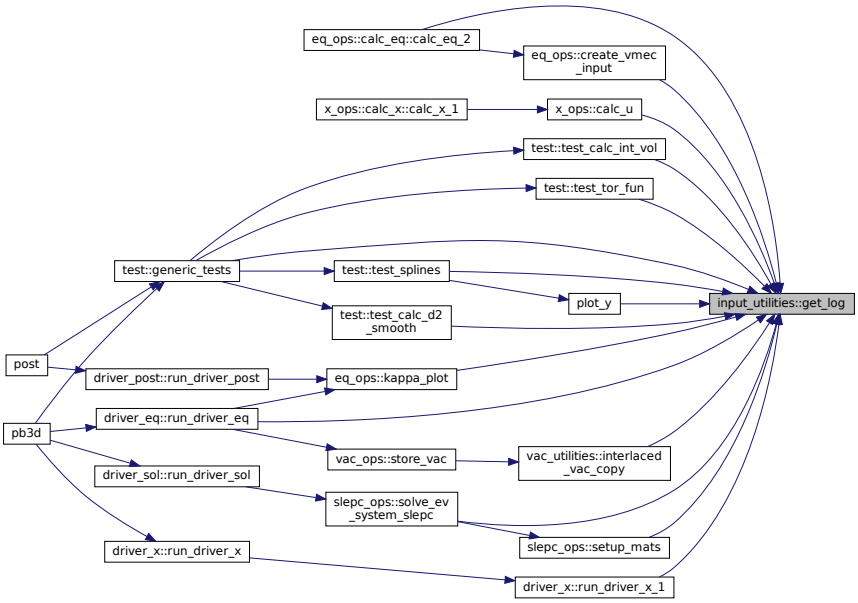
	yes	default answer
in	*ind*	individual pause or not

Definition at line 22 of file input_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.20.2.4 get_real()

```
real(dp) function, public input_utilities::get_real (
               real(dp), intent(in), optional lim_lo,
               real(dp), intent(in), optional lim_hi,
               logical, intent(in), optional ind )
```

Queries for user input for a real value, where allowable range can be provided as well.

If not called by all processes at the same time, `ind` should be set to indicate an individual operation.

Returns

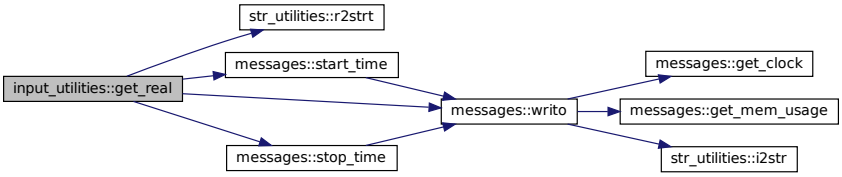
output value

Parameters

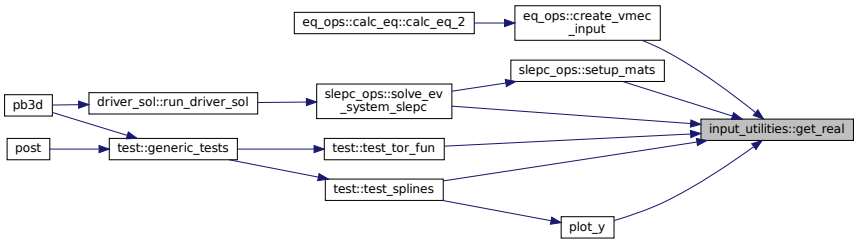
in	*lim_lo*	upper and lower limit
in	*lim_hi*	upper and lower limit
in	*ind*	individual pause or not

Definition at line 77 of file input_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.20.2.5 pause_prog()

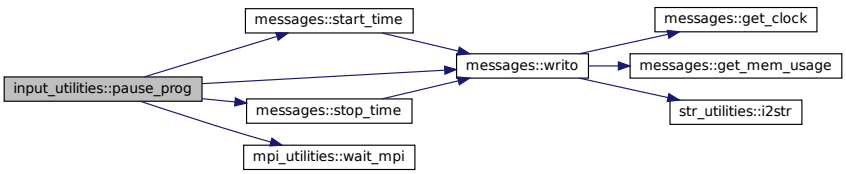
```
subroutine, public input_utilities::pause_prog (
               logical, intent(in), optional ind )
```

Pauses the running of the program.

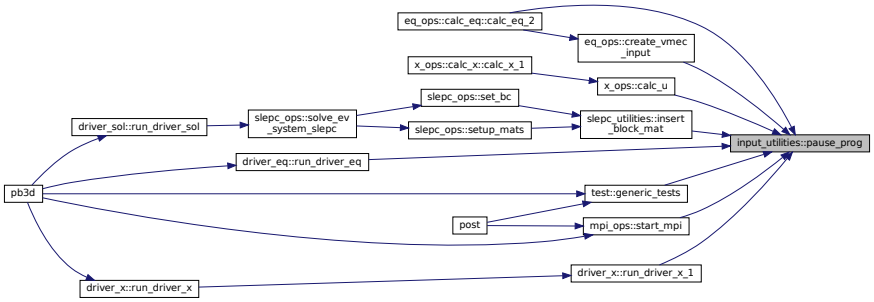
If not called by all processes at the same time, `ind` should be set to indicate an individual operation.

Definition at line 226 of file input_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.21 messages Module Reference

Numerical utilities related to giving output.

Functions/Subroutines

- subroutine, public init_output ()

  Initialize the variables for the module.
- subroutine, public print_hello ()

  Prints first message.
- subroutine, public print_goodbye ()

  Prints last messag.
- subroutine, public init_time ()

  Intialize the time passed to 0.
- subroutine, public start_time ()

  Start a timer.
- subroutine, public stop_time ()

  Stop a timer.
- subroutine, public passed_time ()

  Display the time that has passed between `t1` and `t2`.
- character(len=8) function get_clock ()

  Returns the time.
- subroutine, public print_err_msg (err_msg, routine_name)

  Prints an error message that is either user-provided, or the name of the calling routine.
- subroutine, public lvl_ud (inc)

  Increases/decreases `lvl` of output.
- subroutine, public writo (input_str, persistent, error, warning, alert)

  Write output to file identified by `output_i`.
- subroutine, public print_ar_2 (arr)

  Print an array of dimension 2 on the screen.
- subroutine, public print_ar_1 (arr)

  Print an array of dimension 1 on the screen.
- integer function, public get_mem_usage ()

  Returns the memory usage in kilobytes.

Variables

- integer, public lvl

  determines the indenting. higher `lvl` = more indenting
- character(len=2), public lvl_sep = ''

  characters that separate different levels of output
- character(len=10), public time_sep = ''

  defines the length of time part of output
- logical, public temp_output_active

  true if temporary output is to be written in `temp_output`
- character(len=max_str_ln), dimension(:), allocatable, public temp_output

  temporary output, before output file is opened

B.21.1 Detailed Description

Numerical utilities related to giving output.

B.21.2 Function/Subroutine Documentation

B.21.2.1 get_clock()

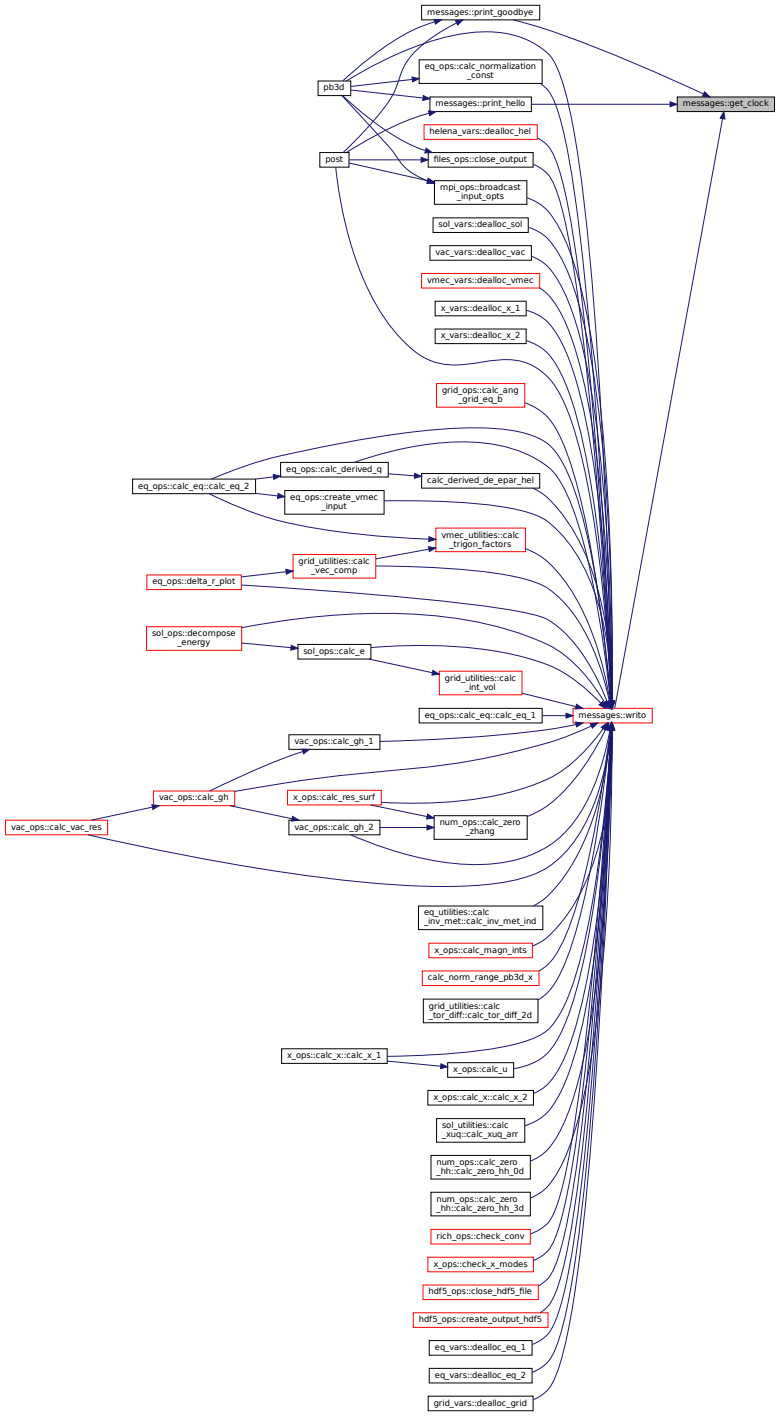
```
character(len=8) function messages::get_clock
```

Returns the time.

from http://infohost.nmt.edu/tcc/help/lang/fortran/date.html

Definition at line 218 of file messages.f90.

Here is the caller graph for this function:



B.21.2.2 get_mem_usage()

```
integer function, public messages::get_mem_usage
```

Returns the memory usage in kilobytes.

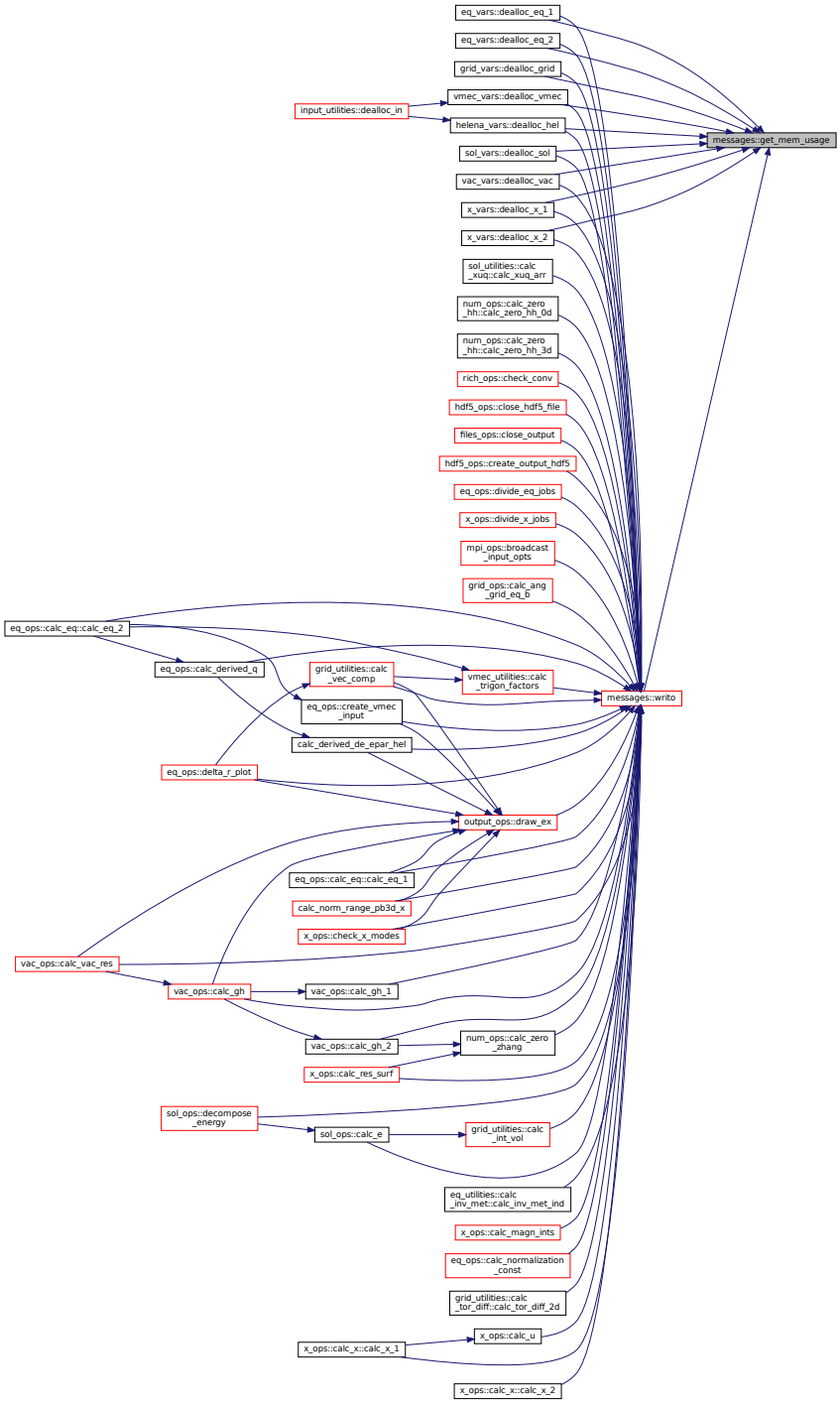
Based on http://stackoverflow.com/questions/22028571/track-memory-usage-in-fortran-90

Note

Only works under linux.
Debug version only

Definition at line 554 of file messages.f90.

Here is the caller graph for this function:



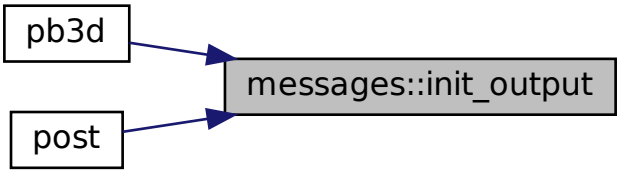
B.21.2.3 init_output()

```
subroutine, public messages::init_output
```

Initialize the variables for the module.

Definition at line 32 of file messages.f90.

Here is the caller graph for this function:



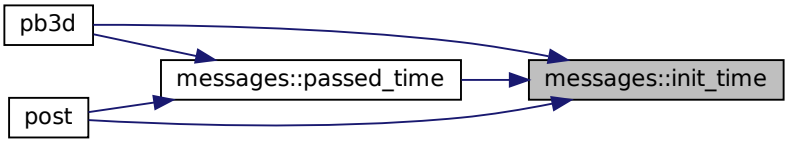
B.21.2.4 init_time()

```
subroutine, public messages::init_time
```

Intialize the time passed to 0.

Definition at line 113 of file messages.f90.

Here is the caller graph for this function:



B.21.2.5 lvl_ud()

```
subroutine, public messages::lvl_ud (
               integer inc )
```

Increases/decreases `lvl` of output.

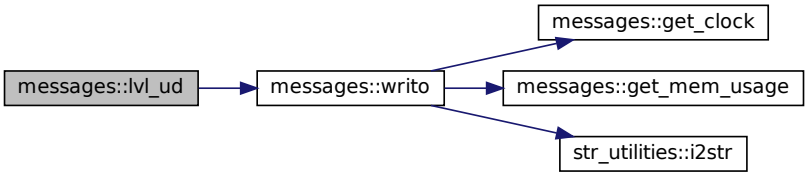
Name stands for level up or down.

Parameters

inc	increment of level

Definition at line 254 of file messages.f90.

Here is the call graph for this function:



B.21.2.6 passed_time()

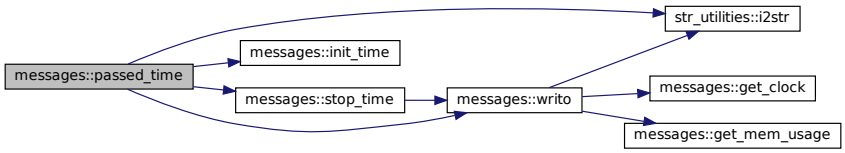
`subroutine, public messages::passed_time`

Display the time that has passed between `t1` and `t2`.

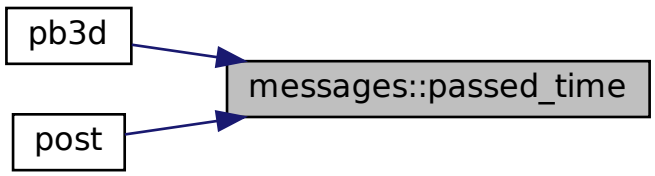
Automatically stops time and resets everything to zero.

Definition at line 152 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.21.2.7 print_ar_1()

```
subroutine, public messages::print_ar_1 (
               real(dp), dimension(:), intent(in) arr )
```

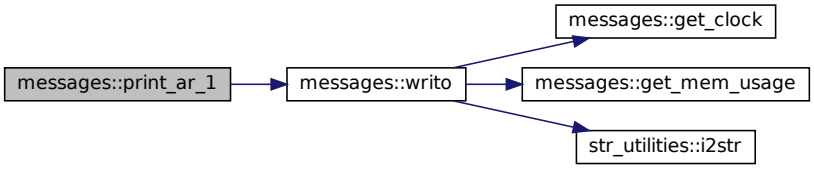
Print an array of dimension 1 on the screen.

Parameters

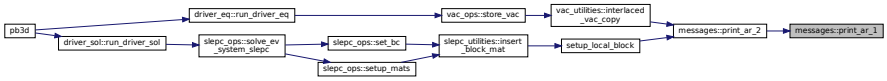
in	*arr*	array to be printed

Definition at line 487 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.21.2.8 print_ar_2()

```
subroutine, public messages::print_ar_2 (
               real(dp), dimension(:,:), intent(in) arr )
```

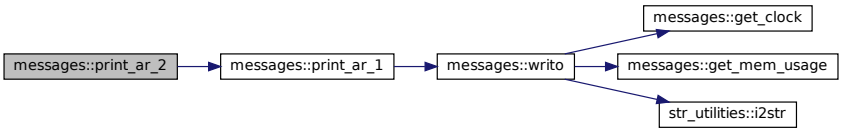
Print an array of dimension 2 on the screen.

Parameters

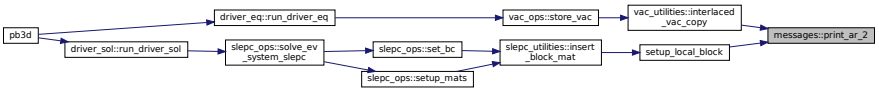
in	*arr*	array to be printed

Definition at line 475 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.21.2.9 print_err_msg()

```
subroutine, public messages::print_err_msg (
              character(len=*), intent(in) err_msg,
              character(len=*), intent(in) routine_name )
```

Prints an error message that is either user-provided, or the name of the calling routine.

Note

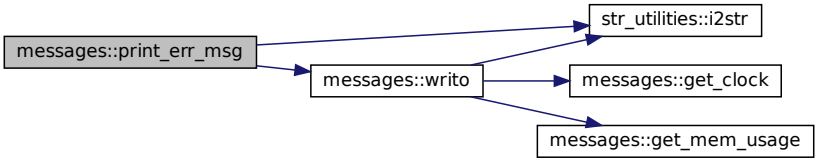
This should be used with the macro CHCKERR.

Parameters

in	*err_msg*	error message to be printed
in	*routine_name*	name of the routine

Definition at line 234 of file messages.f90.

Here is the call graph for this function:



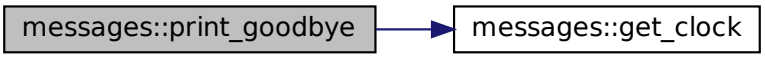
B.21.2.10 print_goodbye()

```
subroutine, public messages::print_goodbye
```

Prints last messag.

Definition at line 99 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

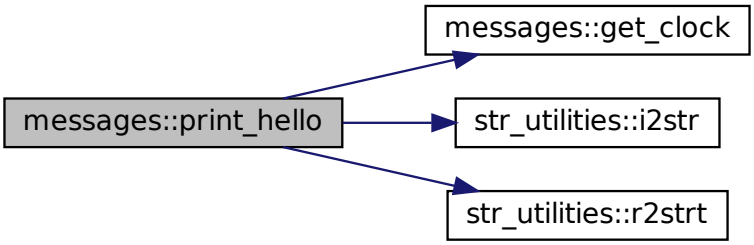
B.21.2.11 print_hello()

```
subroutine, public messages::print_hello
```

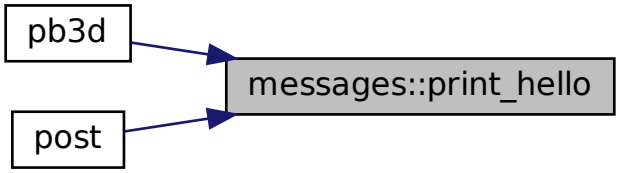
Prints first message.

Definition at line 61 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



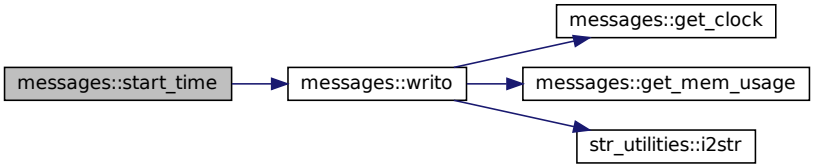
B.21.2.12 start_time()

`subroutine, public messages::start_time`

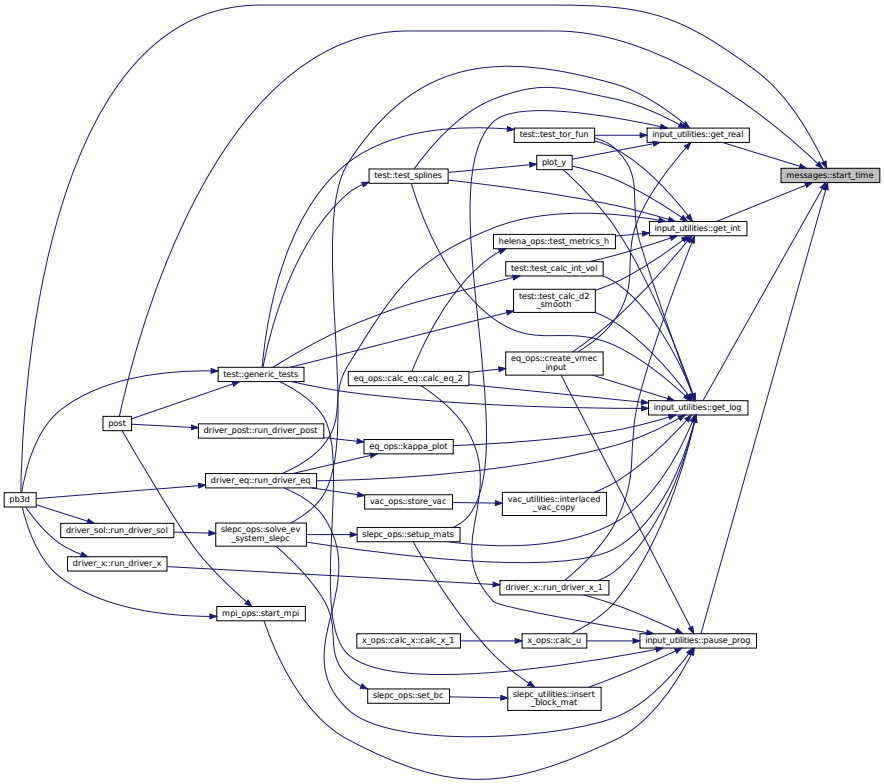
Start a timer.

Definition at line 121 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



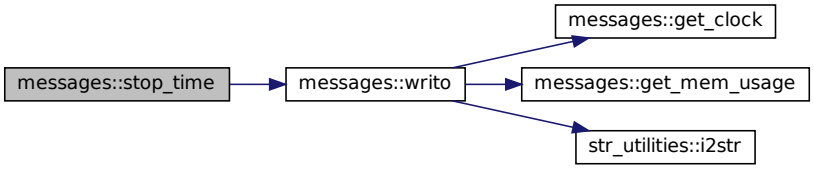
B.21.2.13 stop_time()

```
subroutine, public messages::stop_time
```

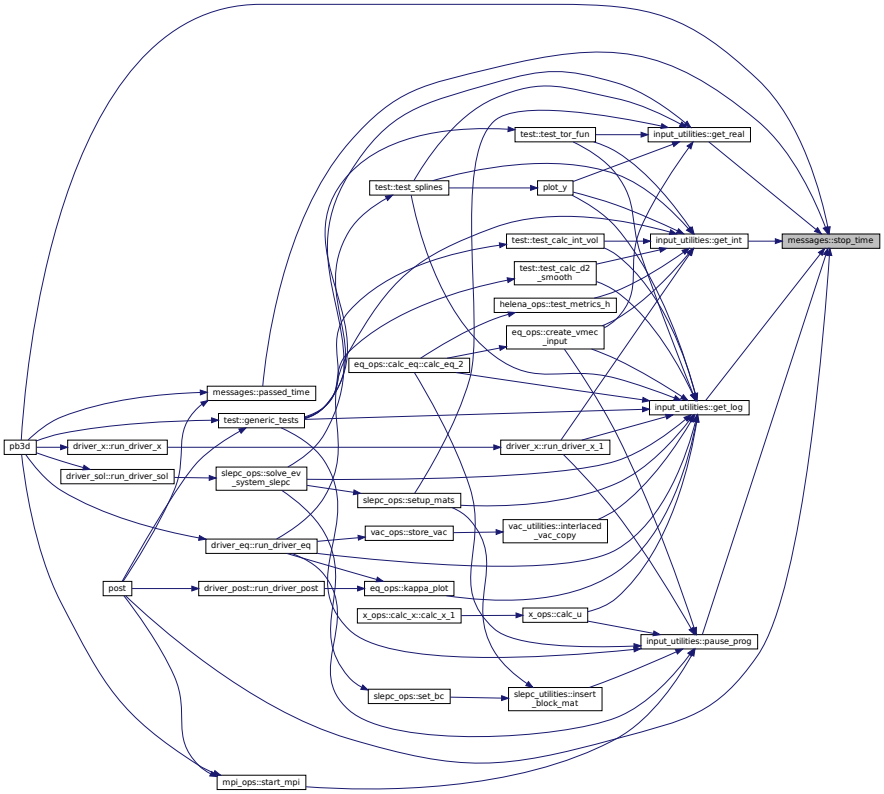
Stop a timer.

Definition at line 132 of file messages.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.21.2.14 writo()

```
subroutine, public messages::writo (
              character(len=*), intent(in) input_str,
              logical, intent(in), optional persistent,
              logical, intent(in), optional error,
              logical, intent(in), optional warning,
              logical, intent(in), optional alert )
```

Write output to file identified by `output_i`.

This is done using the correct indentation for the level (`lvl_loc`) of the output.

By default, only the master outputs, but this can be changed using `persistent`.

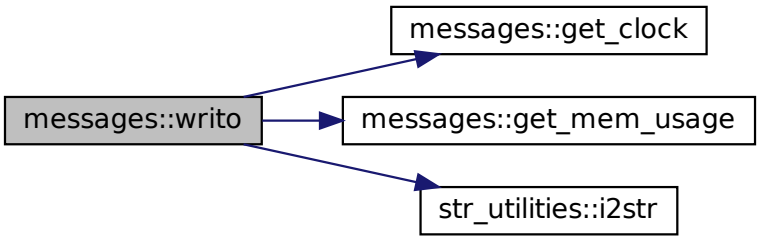
Optionally, special formatting for error, warning or alert can be chosen.

Parameters

in	*input_str*	the name that is searched for
in	*persistent*	output even if not group master
in	*error*	error message
in	*warning*	warning message
in	*alert*	alert message

Definition at line 275 of file messages.f90.

Here is the call graph for this function:



B.21.3 Variable Documentation

B.21.3.1 lvl

```
integer, public messages::lvl
```

determines the indenting. higher `lvl` = more indenting

Definition at line 20 of file messages.f90.

B.21.3.2 lvl_sep

```
character(len=2), public messages::lvl_sep = ''
```

characters that separate different levels of output

Definition at line 21 of file messages.f90.

B.21.3.3 temp_output

```
character(len=max_str_ln), dimension(:), allocatable, public messages::temp_output
```

temporary output, before output file is opened

Definition at line 27 of file messages.f90.

B.21.3.4 temp_output_active

```
logical, public messages::temp_output_active
```

true if temporary output is to be written in `temp_output`

Definition at line 26 of file messages.f90.

B.21.3.5 time_sep

```
character(len=10), public messages::time_sep = ''
```

defines the length of time part of output

Definition at line 22 of file messages.f90.

B.22 mpi_ops Module Reference

Operations related to MPI.

Functions/Subroutines

- integer function, public start_mpi ()
  Start MPI and gather information.
- integer function, public stop_mpi (grid_eq, grid_eq_B, grid_X, grid_X_B, grid_sol, eq_1, eq_2, X_1, X_2, vac, sol)
  Stop MPI.
- integer function, public abort_mpi ()
  Abort MPI suddenly.
- integer function, public broadcast_input_opts ()
  Broadcasts options (e.g. user-prescribed) that are not passed through the HDF5 output file (i.e. `ltest`, `no_plots`, ...).
- subroutine, public sudden_stop (ierr)
  Suddenly stops the computations, aborting MPI, etc.

B.22.1 Detailed Description

Operations related to MPI.

B.22.2 Function/Subroutine Documentation

B.22.2.1 abort_mpi()

```
integer function, public mpi_ops::abort_mpi
```

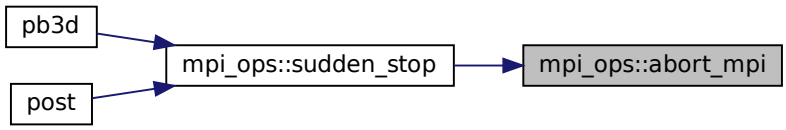
Abort MPI suddenly.

Returns

ierr

Definition at line 227 of file MPI_ops.f90.

Here is the caller graph for this function:



B.22.2.2 broadcast_input_opts()

```
integer function, public mpi_ops::broadcast_input_opts
```

Broadcasts options (e.g. user-prescribed) that are not passed through the HDF5 output file (i.e. `ltest`, `no↩_plots`, ...).

See also

read_input_opts()

Note

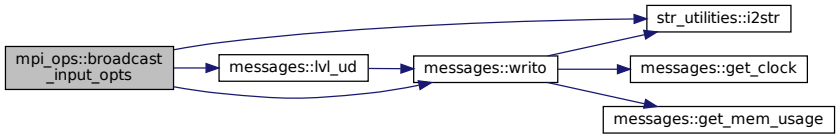
Some variables (e.g. `eq_style`, ...) are not passed over MPI. Every process should call its own reconstruct_pb3d_in() in order to obtain them.

Returns

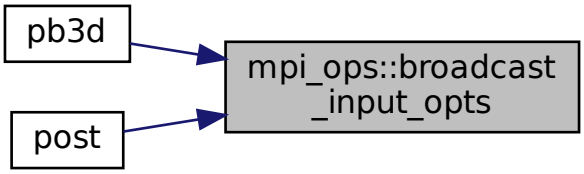
ierr

Definition at line 247 of file MPI_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.22.2.3 start_mpi()

```
integer function, public mpi_ops::start_mpi
```

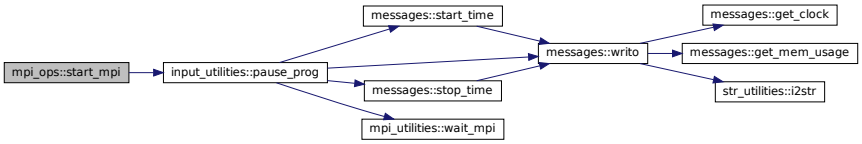
Start MPI and gather information.

Returns

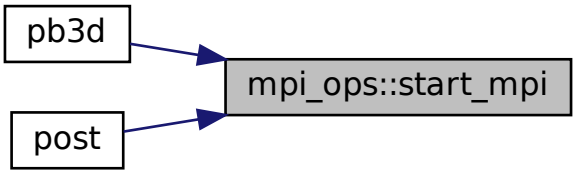
ierr

Definition at line 22 of file MPI_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.22.2.4 stop_mpi()

```
integer function, public mpi_ops::stop_mpi (
               type(grid_type), intent(inout), optional grid_eq,
               type(grid_type), intent(inout), optional, pointer grid_eq_B,
               type(grid_type), intent(inout), optional grid_X,
               type(grid_type), intent(inout), optional, pointer grid_X_B,
               type(grid_type), intent(inout), optional grid_sol,
               type(eq_1_type), intent(inout), optional eq_1,
               type(eq_2_type), intent(inout), optional eq_2,
               type(x_1_type), intent(inout), optional X_1,
               type(x_2_type), intent(inout), optional X_2,
               type(vac_type), intent(inout), optional vac,
               type(sol_type), intent(inout), optional sol )
```

Stop MPI.

Also deallocates:

- grid_eq
- grid_eq_B
- grid_X

- grid_X_B
- grid_sol
- eq_1
- eq_2
- X_1
- X_2
- sol

Returns

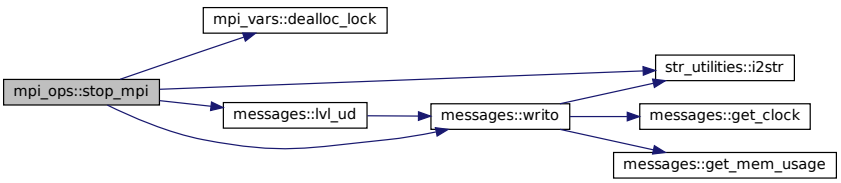
ierr

Parameters

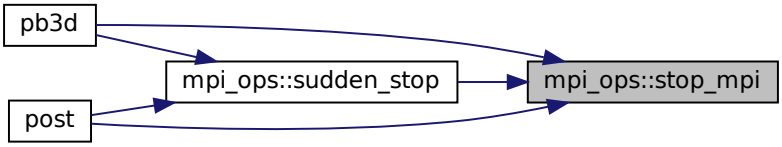
in,out	*grid_eq*	equilibrium grid
in,out	*grid_↩ eq_b*	field-aligned equilibrium grid
in,out	*grid_x*	perturbation grid
in,out	*grid_x_b*	field-aligned perturbation grid
in,out	*grid_sol*	solution grid
in,out	*eq_1*	Flux equilibrium variables
in,out	*eq_2*	metric equilibrium variables
in,out	*x_1*	vectorial perturbation variables
in,out	*x_2*	integrated tensorial perturbation variables
in,out	*vac*	vacuum variables
in,out	*sol*	solution variables

Definition at line 81 of file MPI_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.22.2.5 sudden_stop()

```
subroutine, public mpi_ops::sudden_stop (
               integer, intent(in) ierr )
```

Suddenly stops the computations, aborting MPI, etc.

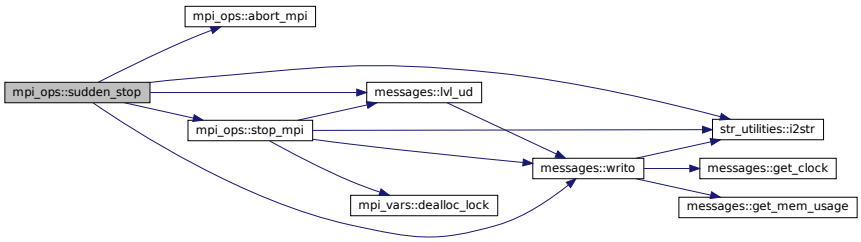
As a special case, if `ierr` = 66, no error message is printed.

Returns

ierr

Definition at line 483 of file MPI_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.23 mpi_utilities Module Reference

Numerical utilities related to MPI.

Interfaces and Types

- interface broadcast_var

  Wrapper function to broadcast a single variable using MPI.
- interface get_ghost_arr

  Fill the ghost regions in an array.
- interface get_ser_var

  Gather parallel variable in serial version on group master.

Functions/Subroutines

- integer function, public redistribute_var (var, dis_var, lims, lims_dis)

  Redistribute variables according to new limits.
- integer function, public wait_mpi ()

  Wait for all processes, wrapper to MPI barrier.
- integer function, public lock_req_acc (lock, blocking)

  Request access to lock of a BL (blocking) or optionally a NB (non-blocking) type.
- integer function, public lock_return_acc (lock)

  Returns access to a lock.
- logical function wl_empty (wl, proc_type, next_procs)

  Decides whether a waiting list is empty.
- integer function lock_notify (lock_loc, rec_rank)

  Notifies a rank that they can get the lock.
- integer function lock_get_notified (lock_loc)

  Get notified that the rank can get the lock.
- integer function, public lock_wl_change (wl_action, blocking, lock_loc, wl, ranks)

  Adds, removes or sets to active a rank from the waiting list for a lock and returns the lock waiting list:
- character(len=max_str_ln) function, public lock_header (lock_loc)

  Returns the header for lock debug messages.

Variables

- logical, public debug_lock = .false.

  print debug information about lock operations
- integer, public n_waits = 0

  number of waits

B.23.1 Detailed Description

Numerical utilities related to MPI.

This includes a lock system, which can be both BL (blocking) or NB (non-blocking). It is based on the implementation of an MPI-IO atomic mode without file support, described in [13].

See also

See mpi_vars.

The reason for this was the fact that using a simple lock file can lead to crashes.

Note

A downside of this method is that in some rare cases a deadlock may occur as the master process, which contains the shared variable with a window that other processes may use, is idle and waiting, whereas the others are still performing lock options. As the master is idle and waiting, its MPI asynchronous communication is not performed. To remedy this, just call wait_MPI() after procedures where lock operations are performed.

B.23.2 Function/Subroutine Documentation

B.23.2.1 lock_get_notified()

```
integer function mpi_utilities::lock_get_notified (
               type(lock_type), intent(in) lock_loc )
```

Get notified that the rank can get the lock.

Note

Based on [13].

Returns

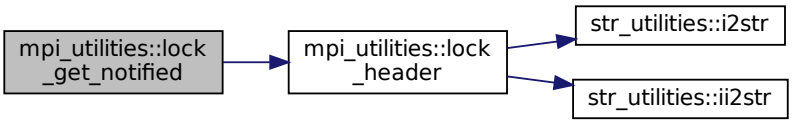
ierr

Parameters

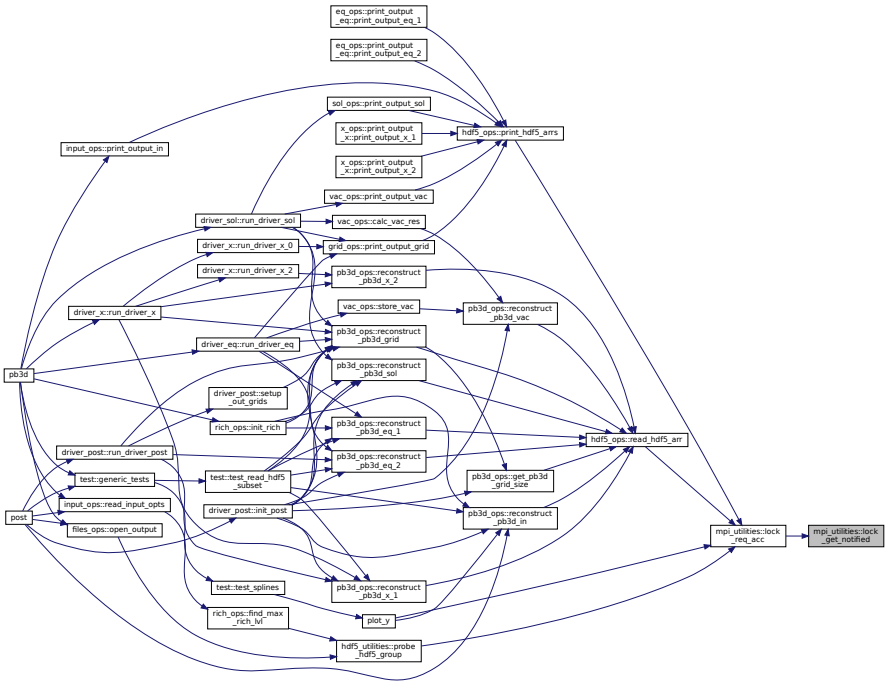
in	*lock_loc*	lock

Definition at line 1063 of file MPI_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.23.2.2 lock_header()

```
character(len=max_str_ln) function, public mpi_utilities::lock_header (
              type(lock_type), intent(in) lock_loc )
```

Returns the header for lock debug messages.

Note

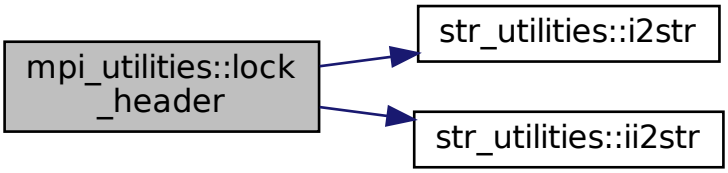
Debug version only

Parameters

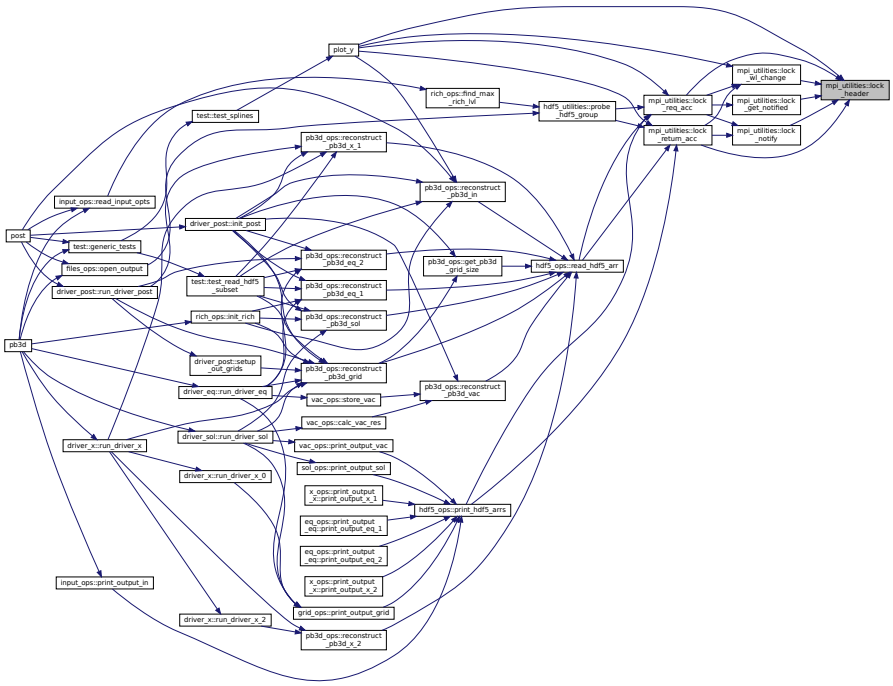
in	*lock_loc*	lock

Definition at line 1217 of file MPI_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.23.2.3 lock_notify()

```
integer function mpi_utilities::lock_notify (
               type(lock_type), intent(in) lock_loc,
               integer, intent(in) rec_rank )
```

Notifies a rank that they can get the lock.

The signal sent is the rank + 1.

Note

Based on [13].

Returns

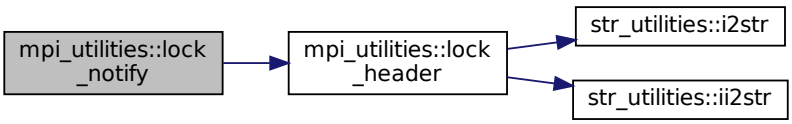
ierr

Parameters

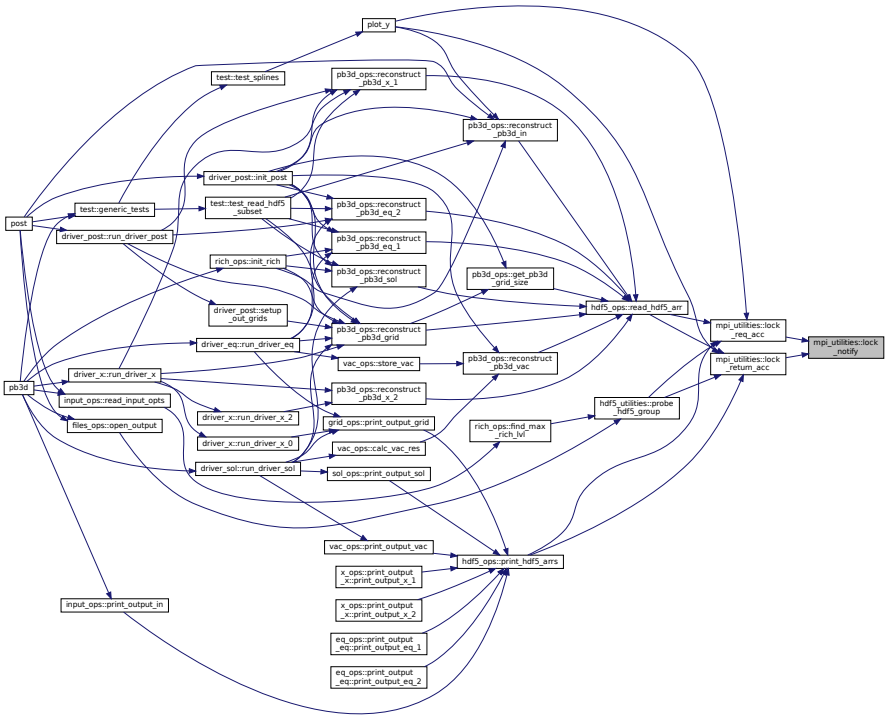
in	*lock_loc*	lock
in	*rec_rank*	receiving rank

Definition at line 1031 of file MPI_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.23.2.4 lock_req_acc()

```
integer function, public mpi_utilities::lock_req_acc (
               type(lock_type), intent(inout) lock,
               logical, intent(in), optional blocking )
```

Request access to lock of a BL (blocking) or optionally a NB (non-blocking) type.

Note

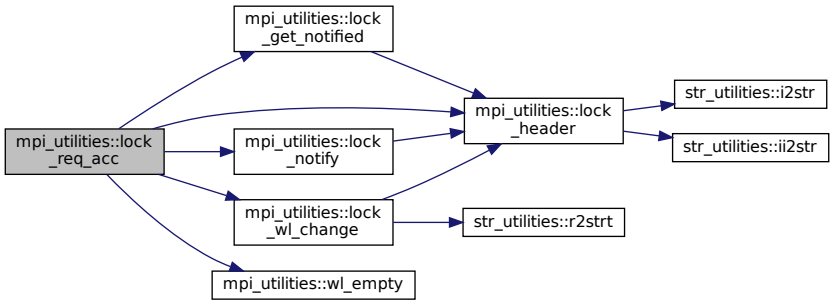
Based on [13].

Returns

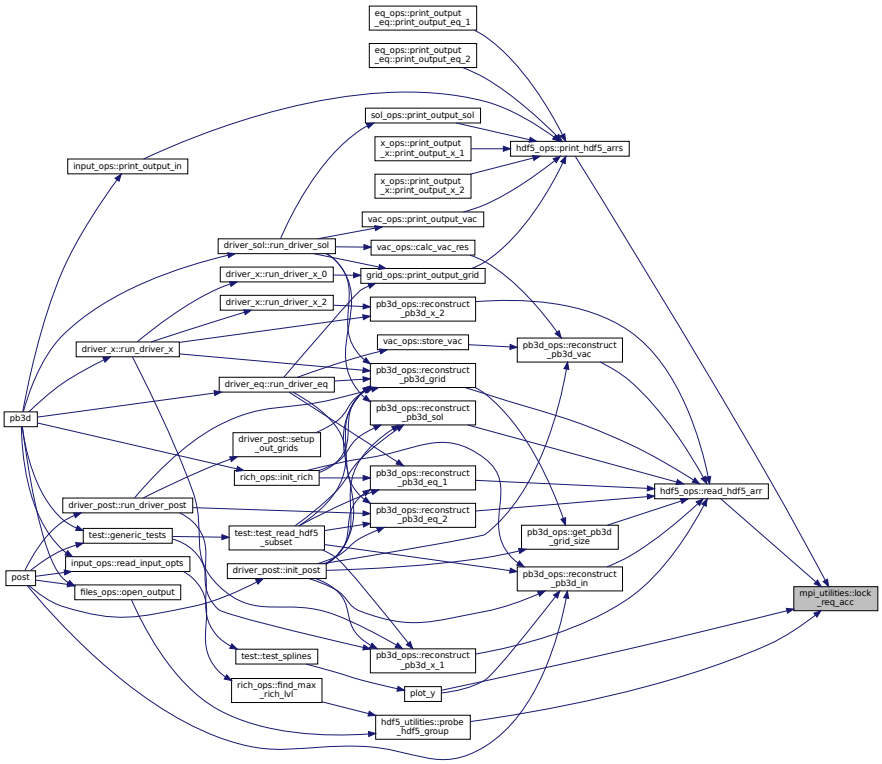
ierr

Definition at line 765 of file MPI_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.23.2.5 lock_return_acc()

```
integer function, public mpi_utilities::lock_return_acc (
              type(lock_type), intent(inout) lock )
```

Returns access to a lock.

The blocking property has been set when requesting the lock.

Note

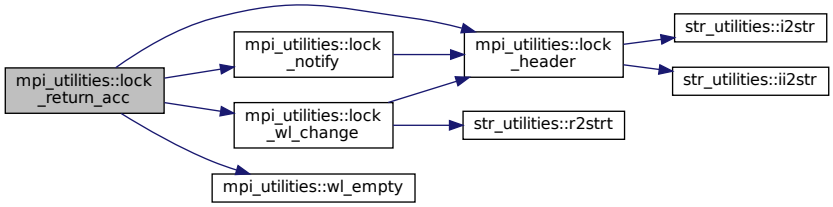
Based on [13].

Returns

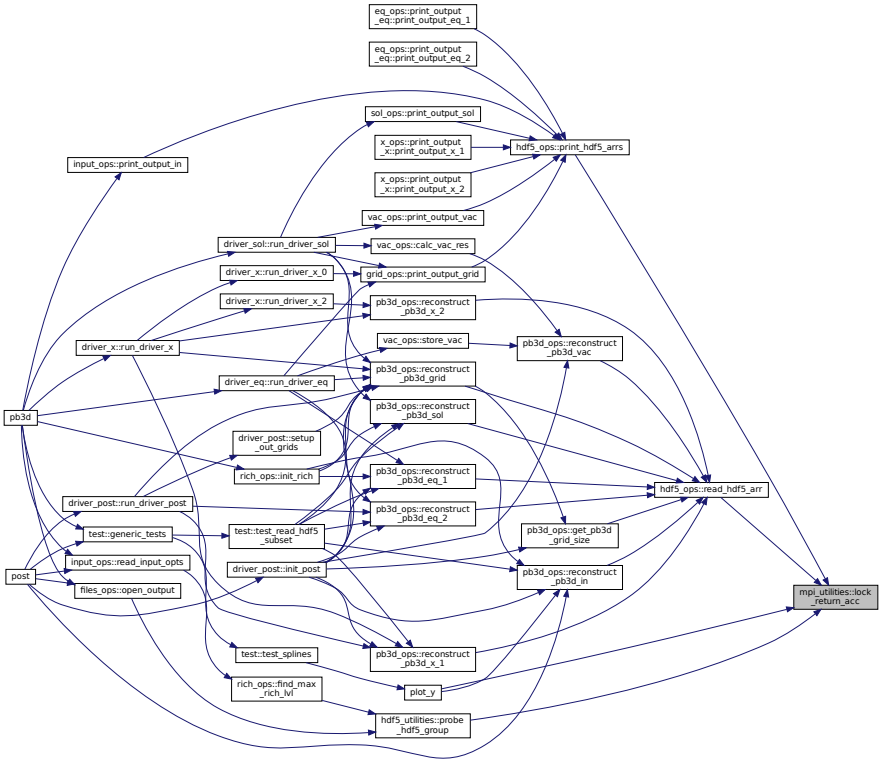
ierr

Definition at line 872 of file MPI_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.23.2.6 lock_wl_change()

```
integer function, public mpi_utilities::lock_wl_change (
               integer, intent(in) wl_action,
               logical, intent(in) blocking,
               type(lock_type), intent(inout) lock_loc,
               integer, dimension(:), intent(inout), allocatable wl,
               integer, dimension(:), intent(in), optional ranks )
```

Adds, removes or sets to active a rank from the waiting list for a lock and returns the lock waiting list:

Actions:

- `wl_action` = 0: remove
- `wl_action` = 1: add
- `wl_action` = 2: active

Or negative equivalents for non-blocking (NB) procs.

Optionally, the rank(s) of the process for which to perform this action can be passed. This is useful for doing the same action on multiple processes.

Note

Based on [13].

Debug version only

Returns

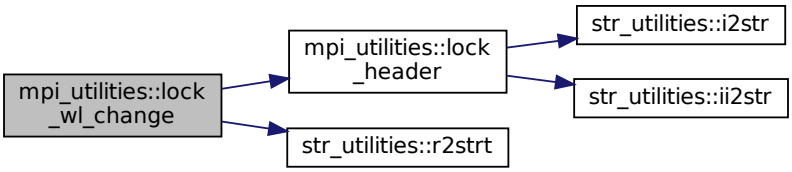
ierr

Parameters

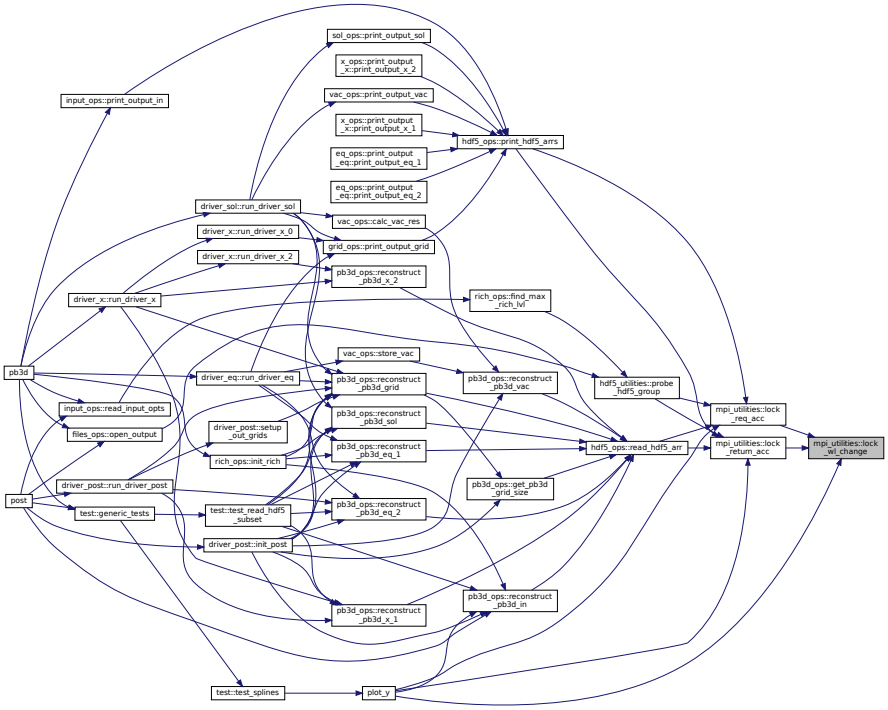
`in`	*wl_action*	action to perform
`in`	*blocking*	the ranks to be changed are blocking
`in,out`	*lock_loc*	lock
`in,out`	*wl*	waiting list
`in`	*ranks*	rank(s) for which to perform option

Definition at line 1111 of file MPI_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.23.2.7 redistribute_var()

```
integer function, public mpi_utilities::redistribute_var (
               real(dp), dimension(:), intent(in) var,
               real(dp), dimension(:), intent(inout) dis_var,
               integer, dimension(2), intent(in) lims,
               integer, dimension(2), intent(in) lims_dis )
```

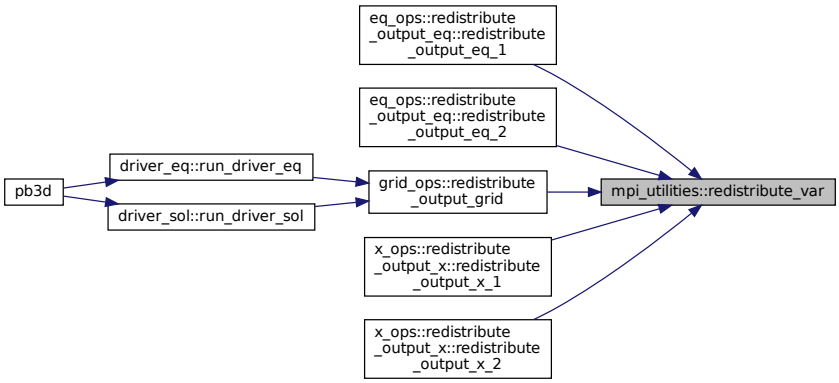
Redistribute variables according to new limits.

Parameters

in	*var*	parallel vector
in,out	*dis_var*	redistributed vector
in	*lims*	indices of parallel vector
in	*lims_dis*	indices of redistributed parallel vector

Definition at line 330 of file MPI_utilities.f90.

Here is the caller graph for this function:



B.23.2.8 wait_mpi()

`integer function, public mpi_utilities::wait_mpi`

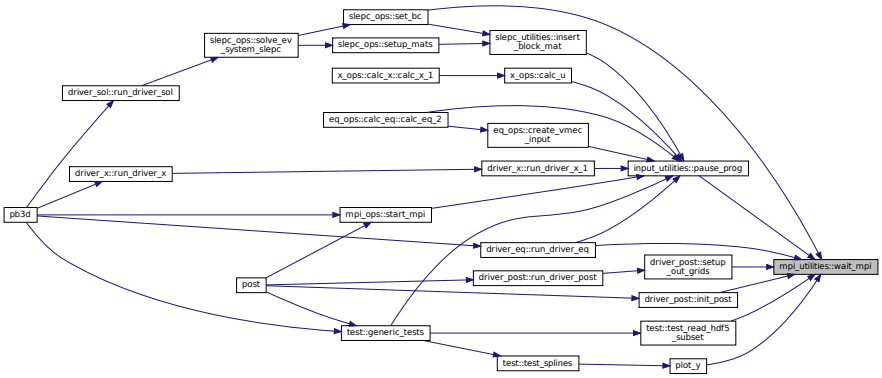
Wait for all processes, wrapper to MPI barrier.

Returns

ierr

Definition at line 744 of file MPI_utilities.f90.

Here is the caller graph for this function:



B.23.2.9 wl_empty()

```
logical function mpi_utilities::wl_empty (
               integer, dimension(:), intent(in) wl,
               integer, dimension(:), intent(in) proc_type,
               integer, dimension(:), intent(inout), optional, allocatable next_procs )
```

Decides whether a waiting list is empty.

The type of process to find is indicated by an array of possible values.

See also

See lock_wl_change() for an explanation of the process type.

Additionally, for NB processes, the negative inverse of these values are used.

If the waiting list is not empty, the next process(es) can optionally be returned.

Note

Based on [13].

Returns

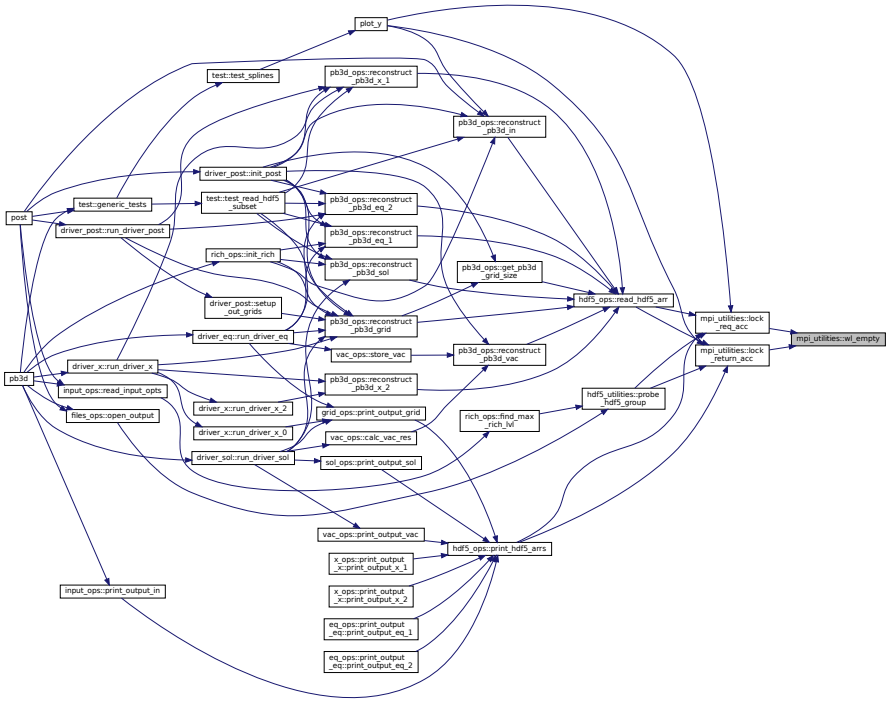
ierr

Parameters

in	*wl*	waiting list
in	*proc_type*	types of processes accepted
in,out	*next_procs*	next process(es) if not empty

Definition at line 985 of file MPI_utilities.f90.

Here is the caller graph for this function:



B.23.3 Variable Documentation

B.23.3.1 debug_lock

```
logical, public mpi_utilities::debug_lock = .false.
```

print debug information about lock operations

Note

Debug version only

Definition at line 40 of file MPI_utilities.f90.

B.23.3.2 n_waits

```
integer, public mpi_utilities::n_waits = 0
```

number of waits

Note

Debug version only

Definition at line 41 of file MPI_utilities.f90.

B.24 mpi_vars Module Reference

Variables pertaining to MPI.

Interfaces and Types

- type lock_type
  lock type

Functions/Subroutines

- integer function, public init_lock (lock_loc, wu_tag)
  Initializes a lock.
- integer function, public dealloc_lock (lock_loc)
  Deallocates a lock.

Variables

- type(lock_type), public hdf5_lock
  HDF5 lock.

B.24.1 Detailed Description

Variables pertaining to MPI.

B.24.2 Function/Subroutine Documentation

B.24.2.1 dealloc_lock()

```
integer function, public mpi_vars::dealloc_lock (
              class(lock_type), intent(inout) lock_loc )
```

Deallocates a lock.

Note

Should be called collectively.

Returns

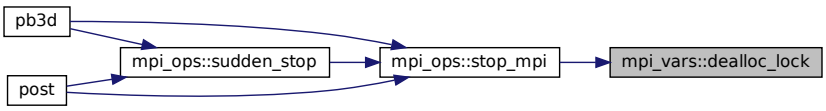
ierr

Parameters

in,out	*lock_loc*	lock

Definition at line 133 of file MPI_vars.f90.

Here is the caller graph for this function:



B.24.2.2 init_lock()

```
integer function, public mpi_vars::init_lock (
               class(lock_type), intent(inout) lock_loc,
               integer, intent(in) wu_tag )
```

Initializes a lock.

Note

1. Should be called collectively.
2. Every lock should have a unique wakeup tag.

Returns

ierr

Parameters

in,out	*lock_loc*	lock
in	*wu_tag*	wakeup tag

Definition at line 87 of file MPI_vars.f90.

B.24.3 Variable Documentation

B.24.3.1 hdf5_lock

```
type(lock_type), public mpi_vars::hdf5_lock
```

HDF5 lock.

Definition at line 76 of file MPI_vars.f90.

B.25 num_ops Module Reference

Numerical operations.

Interfaces and Types

- interface calc_zero_hh
  Finds the zero of a function using Householder iteration.

Functions/Subroutines

- character(len=max_str_ln) function, public calc_zero_zhang (zero, fun, x_int_in)
  Finds the zero of a function using Zhang's method, which is simpler than Brent's method.

Variables

- logical, public debug_calc_zero = .false.
  plot debug information for calc_zero

B.25.1 Detailed Description

Numerical operations.

B.25.2 Function/Subroutine Documentation

B.25.2.1 calc_zero_zhang()

```
character(len=max_str_ln) function, public num_ops::calc_zero_zhang (
               real(dp), intent(inout) zero,
                fun,
               real(dp), dimension(2), intent(in) x_int_in )
```

Finds the zero of a function using Zhang's method, which is simpler than Brent's method.

Taken from from Steven Stage's correction of Zhang's paper [19].

Unlike Householder, Zhang's method needs an interval `x_int_in` to work in, not a guess. Also, it does not require the derivative of the function.

The routine returns an error message if no zero is found, and which is empty otherwise.

Note

The interval `x_int_in` needs to be so that the function values at either end are of different value.

Parameters

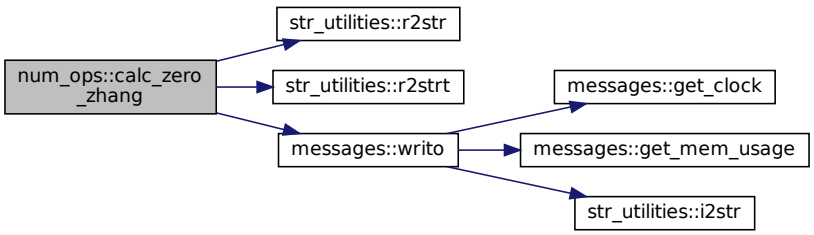
in,out	*fun*	fun(x) with • x abscissa • fun ordinate
in,out	*zero*	output
in	*x_int_in*	interval

Returns

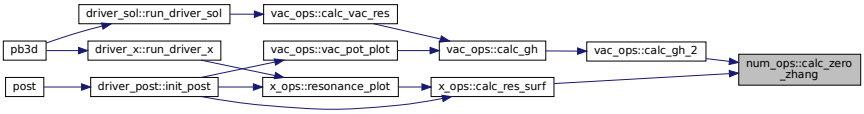
possible error message

Definition at line 462 of file num_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.25.3 Variable Documentation

B.25.3.1 debug_calc_zero

```
logical, public num_ops::debug_calc_zero = .false.
```

plot debug information for calc_zero

Note

Debug version only

Definition at line 22 of file num_ops.f90.

B.26 num_utilities Module Reference

Numerical utilities.

Interfaces and Types

- interface add_arr_mult

  Add to an array (3) the product of arrays (1) and (2).
- interface bubble_sort

  Sorting with the bubble sort routine.
- interface calc_det

  Calculate determinant of a matrix.
- interface calc_int

  Integrates a function using the trapezoidal rule.
- interface calc_inv

  Calculate inverse of square matrix A.
- interface calc_mult

  Calculate matrix multiplication of two square matrices $\overline{AB} = \overline{A}\,\overline{B}$.
- interface con

  Either takes the complex conjugate of a square matrix element A, defined on a 3-D grid, or not.
- interface con2dis

  Convert between points from a continuous grid to a discrete grid.
- interface conv_mat

  Converts a (symmetric) matrix A with the storage convention described in eq_vars.eq_2_type.
- interface dis2con

  Convert between points from a discrete grid to a continuous grid.
- interface order_per_fun

  Order a periodic function to include $0 \ldots 2\pi$ *and an overlap.*
- interface round_with_tol

  Rounds an arry of values to limits, with a tolerance 10^{-5} *that can optionally be modified.*
- interface spline

  Wrapper to the pspline library, making it easier to use for 1-D applications where speed is not the main priority. If spline representations are to be reused, manually use the library.

Functions/Subroutines

- integer function calc_inv_0d (inv_0D, A)

  private constant version
- subroutine, public calc_aux_utilities (n_mod)

  Initialize utilities for fast future reference, depending on program style.
- integer function calc_derivs_1d_id (deriv, dims)

  Returns the 1D indices for derivatives of a certain order in certain dimensions.
- integer function, dimension(:,:), allocatable, public derivs (order, dims)

  Returns derivatives of certain order.
- integer function, public check_deriv (deriv, max_deriv, sr_name)

  checks whether the derivatives requested for a certain subroutine are valid
- integer function, public calc_ext_var (ext_var, var, var_points, ext_point, deriv_in)

  Extrapolates a function.

- integer function, public calc_coeff_fin_diff (deriv, nr, ind, coeff)

  Calculate the coefficients for finite differences.
- integer function, public c (ij, sym, n, lim_n)

  Convert 2-D coordinates (i,j) to the storage convention used in matrices.
- recursive integer function, public fac (n)

  Calculate factorial.
- integer function, public is_sym (n, nn, sym)

  Determines whether a matrix making use of the storage convention in eq_vars.eq_2_type is symmetric or not.
- recursive integer function, public lcm (u, v)

  Returns common multiple using the Euclid's algorithm.
- recursive integer function, public gcd (u, v)

  Returns least denominator using the GCD.
- subroutine, public shift_f (Al, Bl, Cl, A, B, C)

  Calculate multiplication through shifting of fourier modes A and B into C.
- subroutine, public solve_vand (n, a, b, x, transp)

  Solve a Vandermonde system $\overline{A}\,\vec{X} = \vec{B}$.
- integer function, public calc_d2_smooth (fil_N, x, y, D2y, style)

  Calculate second derivative with smoothing formula by Holoborodko, [9].

Variables

- integer, dimension(:,:,:), allocatable, public d

  1-D array indices of derivatives
- integer, dimension(:,:), allocatable, public m

  1-D array indices of metric indices
- integer, dimension(:,:), allocatable, public f

  1-D array indices of Fourier mode combination indices
- logical, public debug_con2dis_reg = .false.

  plot debug information for con2dis_reg()
- logical, public debug_calc_coeff_fin_diff = .false.

  plot debug information for calc_coeff_fin_diff()

B.26.1 Detailed Description

Numerical utilities.

B.26.2 Function/Subroutine Documentation

B.26.2.1 c()

```
integer function, public num_utilities::c (
               integer, dimension(2), intent(in) ij,
               logical, intent(in) sym,
               integer, intent(in), optional n,
               integer, dimension(2,2), intent(in), optional lim_n )
```

Convert 2-D coordinates (i,j) to the storage convention used in matrices.

Their size is by default taken to be 3:

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \text{ or } \begin{pmatrix} 1 & & \\ 2 & 4 & \\ 3 & 5 & 6 \end{pmatrix} \text{ for symmetic matrices .}$$

Optionally, this can be changed using n.

The value of c is then given by

$$c = (j-1)n + i$$

for non-symmetric matrices, and by

$$c = (j-1)n + i - (j-1)\frac{j}{2} \quad \text{if } i > j$$
$$c = (i-1)n + j - (i-1)\frac{i}{2} \quad \text{if } j > i$$

since $\sum_{k=1}^{j-1} k = \frac{(j-1)j}{2}$.

For local indices in submatrices, the limits in both dimensions have to be passed. The results for the full matrix are then subtracted by amount corresponding to the left, above and below parts with respect to the submatrix:

- left:
$$\sum_{i=1}^{m(2)-1} n - i + 1 = (m(2)-1)\left(n + 1 - \frac{m(2)}{2}\right)$$
- above (if positive):
$$\sum_{i=1}^{j} (m(1) - m(2) + 1 - i) = j\left(m(1) - m(2) + \frac{1}{2} - \frac{j^*}{2}\right) \text{ , with } j^* = \min(0, j, m(1) - m(2) + 1)$$
- below:
$$\sum_{i=1}^{j-1} n - M(1) = (n - M(1))(j-1)$$

where m = min and M = max.

Note

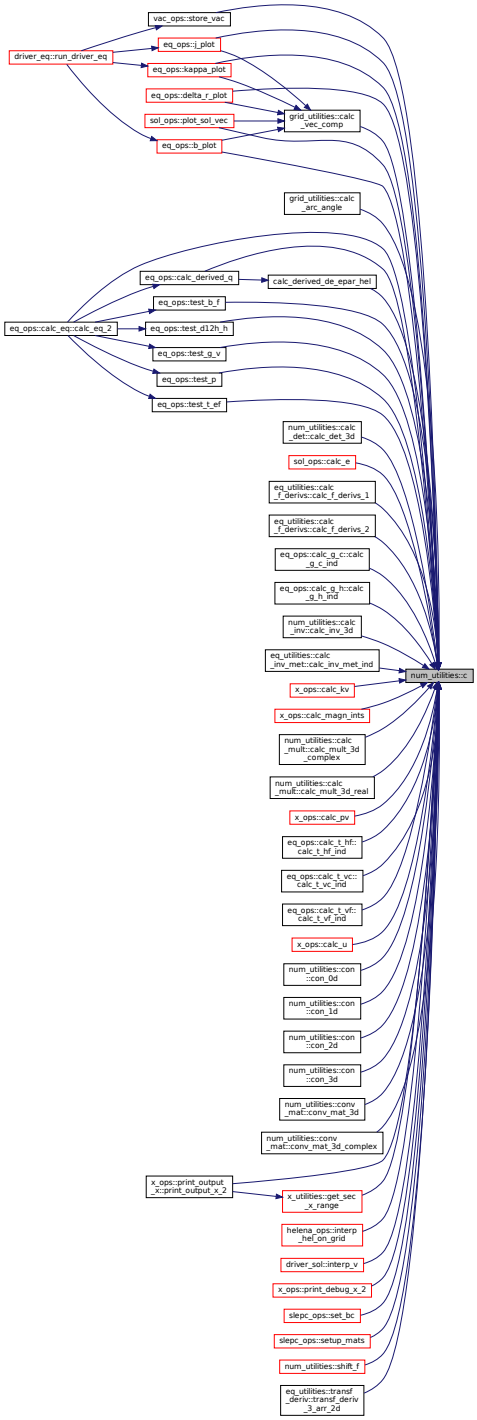
1. The submatrix version is not fast, so results should be saved and reused.
2. No checks are done whether the indices make sense.

Parameters

in	*ij*	2D coords. (i,j)
in	*sym*	whether symmetric
in	*n*	max of 2-D coords.
in	*lim_n*	min. and max. of 2-D coords.

Definition at line 2556 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.2 calc_aux_utilities()

```
subroutine, public num_utilities::calc_aux_utilities (
               integer, intent(in), optional n_mod )
```

Initialize utilities for fast future reference, depending on program style.

Utilities initialized:

- derivatives
- metrics
- Fourier modes (optionally)

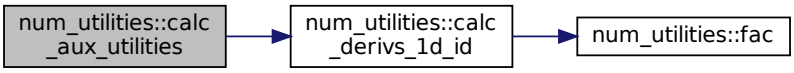
If Fourier modes are also initialized, the quantity `n_mod` has to be provided as well.

Parameters

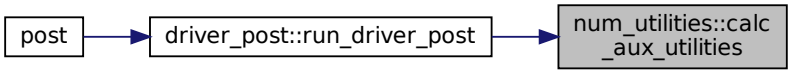
`in`	*n_mod*	n_mod for Fourier modes

Definition at line 2063 of file num_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.26.2.3 calc_coeff_fin_diff()

```
integer function, public num_utilities::calc_coeff_fin_diff (
               integer, intent(in) deriv,
               integer, intent(in) nr,
               integer, intent(in) ind,
               real(dp), dimension(:), intent(inout), allocatable coeff )
```

Calculate the coefficients for finite differences.

These represent a derivative of degree `deriv` at an index `ind`, by the weighted sum of a number `nr` of points with `1<=ind<=nr`.

They are found by solving a Vandermonde system, multiplied by the faculty of the derivative.

The result returned in `coeff(1:nr)` are used in

$$\frac{df}{dx} = \sum_{j=1}^{n} c(j) f(i+j).$$

where i = `ind`, n = `nr` and c = `coeff`

Note

They need to be divided by the step size before usage.

Examples:

- symmetric finite differences for derivative `deriv` of order `ord`:
  - $m =$ ceiling$(\frac{p+d}{2})$ to guarantee the order
  - $n = 1 + 2m$
  - $i = 1 + m$
- left finite differences for derivative `deriv` of order `ord`:
  - $n = 1 + d + p$
  - $i = m$ where p = `ord`, d = `deriv` and m = `nr_2`.

Returns

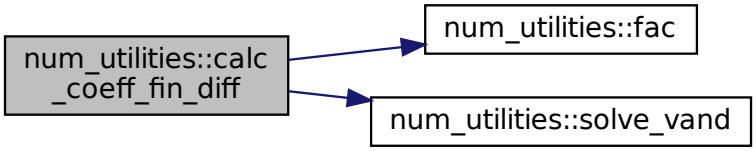
ierr

Parameters

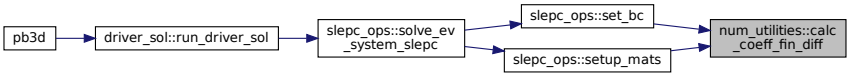
`in`	*deriv*	degree of derivative
`in`	*nr*	number of points
`in`	*ind*	position of derivative
`in,out`	*coeff*	output coefficients

Definition at line 2449 of file num_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.26.2.4 calc_d2_smooth()

```
integer function, public num_utilities::calc_d2_smooth (
               integer, intent(in) fil_N,
               real(dp), dimension(:), intent(in) x,
               real(dp), dimension(:), intent(in) y,
               real(dp), dimension(:), intent(inout) D2y,
               integer, intent(in) style )
```

Calculate second derivative with smoothing formula by Holoborodko, [9].

The formula used is

$s\left(k\right) = \left[\left(2N-10\right)s\left(k+1\right) - \left(N+2k+3\right)s\left(k+2\right)\right] / \left(N-2k-1\right)$

with $s\left(k>M\right)=0$ and $\left(k=N\right)=1$.

for a given N.

For `style` 1, central differences are used:

$\frac{\partial^2 y}{\partial x^2} \approx \frac{1}{2^{N-3}} \left(\sum_{k=1}^{M} \alpha_k \left(y_k + y_{-k} \right) - 2 y_0 \sum_{k=1}^{M} \alpha_k \right)$

and for `style` 2, backward differences:

$\frac{\partial^2 y}{\partial x^2} \approx \frac{1}{2^{N-3}} \left(\sum_{k=1}^{M} \beta_k \left(y_{-M+k} + y_{-M-k} \right) - 2 y_{-M} \sum_{k=1}^{M} \beta_k \right)$

with α_k and β_k given by

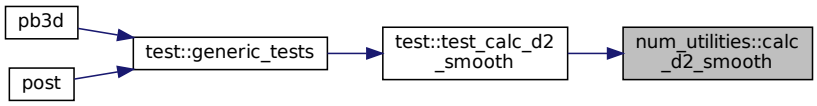
$$\alpha_k = \frac{4k^2 s_k}{\left(x_k - x_{-k}\right)^2}$$
$$\beta_k = \frac{4k^2 s_k}{\left(x_{-M+k} - x_{-M-k}\right)^2}.$$

Parameters

in	*fil↩ _n*	filter length (must be odd)
in	*x*	input abscissa
in	*y*	input ordinate
in,out	*d2y*	second derivative

Definition at line 2829 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.5 calc_derivs_1d_id()

```
integer function num_utilities::calc_derivs_1d_id (
               integer, dimension(:), intent(in) deriv,
               integer, intent(in) dims )
```

Returns the 1D indices for derivatives of a certain order in certain dimensions.

The algorithm works by considering a structure such as the following example in three dimensions:

1. (0,0,0)
2. (1,0,0)
3. (0,1,0)
4. (0,0,1)
5. (2,0,0)
6. (1,1,0)
7. (1,0,1)
8. (0,2,0)
9. (0,1,1)
10. (0,0,2)
11. (3,0,0)
12. (2,1,0)

etc...

By then defining d as the vector of derivatives, n as the size of d, $D = \sum_i^n d(i)$ the total degree of the derivative, and I as the index of the last nonzero element in d, and extending d to the left by considering $d(0)$ = 0, the following formula can be can be deduced for the displacements in this table with respect to index 1:

displacements in this table with respect to index 1:

$$\sum_{j=0}^{I-1} \sum_{i=0}^{D-(d(0)+\ldots+d(j))-1} \left(\begin{array}{c} n-j+i-1 \\ i \end{array} \right),$$

making use of the binomial coefficients

$$\left(\begin{array}{c} a \\ b \end{array} \right) = \frac{a!}{b!(a-b)!}$$

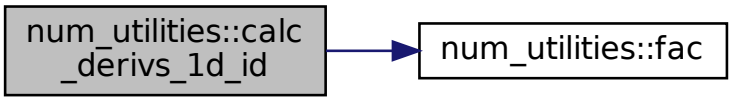
It can be seen that each of the terms in the summation in j corresponds to the displacement in dimension j and the binomial coefficient comes from the classic stars and bars problem.

Parameters

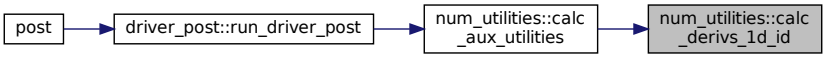
in	*deriv*	derivatives
in	*dims*	nr. of dimensions

Definition at line 2146 of file num_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.26.2.6 calc_ext_var()

```
integer function, public num_utilities::calc_ext_var (
               real(dp), intent(inout) ext_var,
               real(dp), dimension(:), intent(in) var,
               real(dp), dimension(:), intent(in) var_points,
               real(dp), intent(in) ext_point,
               integer, intent(in), optional deriv_in )
```

Extrapolates a function.

This is done using linear or quadratic interpolation, depending on the number of points and values given.

The data should be given sorted, in ascending order, without duplicate points in var_points.

It uses the following solution:

$$\overline{\mathsf{A}}\,\vec{b} = \vec{c}$$

which is written out to

$$\begin{pmatrix} x_1^0 & x_1^1 & x_1^2 & \dots \\ x_2^0 & x_2^1 & x_2^2 & \dots \\ x_3^0 & x_3^1 & x_3^2 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \end{pmatrix}$$

where v_{i-1} = var_i.

This is solved for the coefficients of the interpolating polynomial

```
ext_var = a_0 + a_1 x + a_2 x^2 + ...
```

There is an optional flag to look for the k^{th} derivative of the function instead of the function itself, where k should be lower than the degree of the polynomial.

Returns

ierr

Parameters

in,out	*ext_var*	output
in	*var*	abscissa
in	*var_points*	ordinate
in	*ext_point*	point to which extrapolate
in	*deriv_in*	specifies an optional derivative

Definition at line 2325 of file num_utilities.f90.

B.26.2.7 calc_inv_0d()

```
integer function num_utilities::calc_inv_0d (
```

```
real(dp), dimension(:,:), intent(inout) inv_0D,
real(dp), dimension(:,:), intent(in) A )
```

private constant version

Parameters

in,out	*inv_0d*	output
in	*a*	input

Definition at line 772 of file num_utilities.f90.

B.26.2.8 check_deriv()

```
integer function, public num_utilities::check_deriv (
               integer, dimension(3), intent(in) deriv,
               integer, intent(in) max_deriv,
               character(len=*), intent(in) sr_name )
```

checks whether the derivatives requested for a certain subroutine are valid

Returns

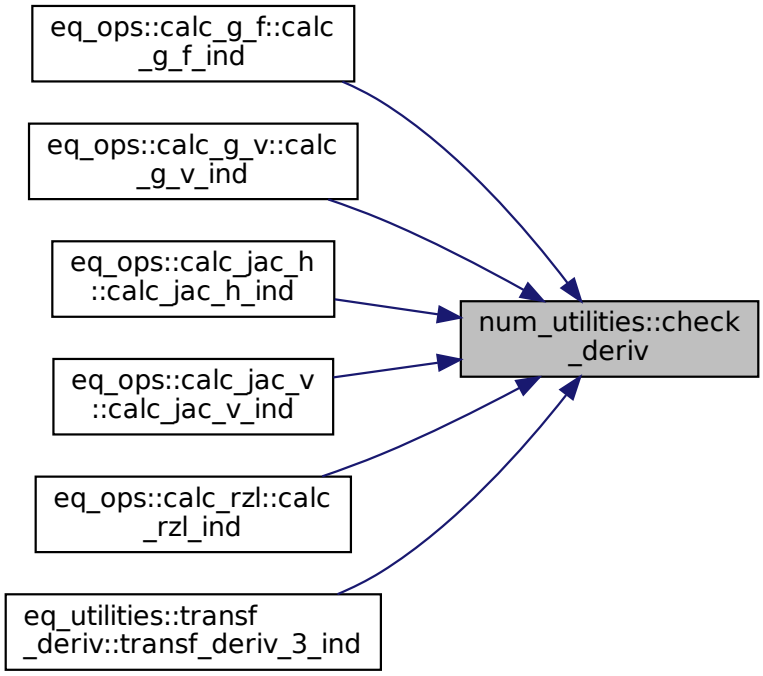
ierr

Parameters

in	*deriv*	derivative to be checked
in	*max_deriv*	maximum derivative allowed
in	*sr_name*	name of subroutine

Definition at line 2260 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.9 derivs()

```
integer function, dimension(:,:), allocatable, public num_utilities::derivs (
               integer, intent(in) order,
               integer, intent(in), optional dims )
```

Returns derivatives of certain order.

Parameters

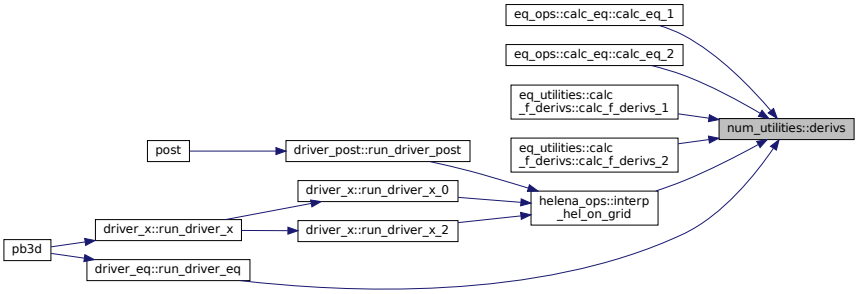
in	*order*	order of derivative
in	*dims*	nr. of dimensions

Returns

array of all unique derivatives

Definition at line 2246 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.10 fac()

```
recursive integer function, public num_utilities::fac (
              integer, intent(in) n )
```

Calculate factorial.

Parameters

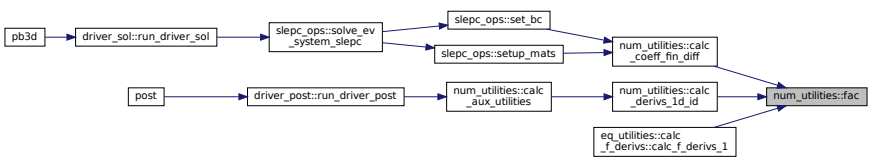
in	*n*	input

Returns

output

Definition at line 2609 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.11 gcd()

```
recursive integer function, public num_utilities::gcd (
               integer, intent(in) u,
               integer, intent(in) v )
```

Returns least denominator using the GCD.

See also

From https://rosettacode.org/wiki/Least_common_multiple#Fortran

Parameters

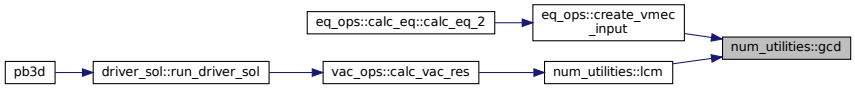
in	*u*	input
in	*v*	input

Returns

result

Definition at line 2669 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.12 is_sym()

```
integer function, public num_utilities::is_sym (
               integer, intent(in) n,
               integer, intent(in) nn,
               logical, intent(inout) sym )
```

Determines whether a matrix making use of the storage convention in eq_vars.eq_2_type is symmetric or not.

Returns

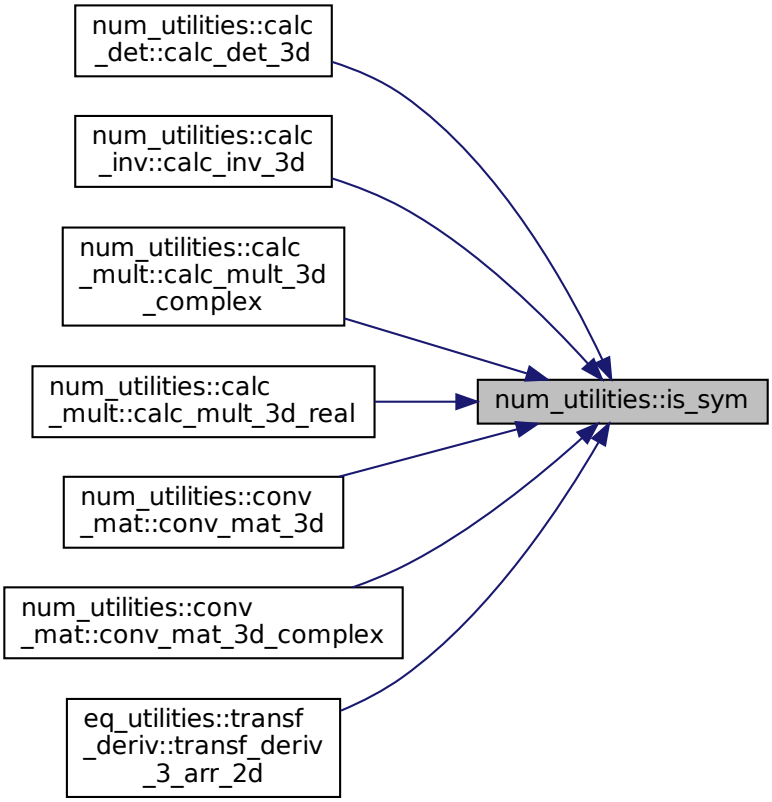
ierr

Parameters

in	*n*	size of matrix
in	*nn*	number of elements in matrix
in,out	*sym*	output

Definition at line 2626 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.2.13 lcm()

```
recursive integer function, public num_utilities::lcm (
               integer, intent(in) u,
               integer, intent(in) v )
```

Returns common multiple using the Euclid's algorithm.

See also

From https://rosettacode.org/wiki/Greatest_common_divisor#Recursive_Euclid_algorithm_3

Parameters

in	*u*	input
in	*v*	input

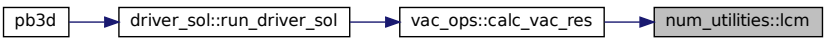
Returns

result

Definition at line 2657 of file num_utilities.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



B.26.2.14 shift_f()

```
subroutine, public num_utilities::shift_f (
               integer, dimension(2), intent(in) Al,
               integer, dimension(2), intent(in) Bl,
               integer, dimension(2), intent(in) Cl,
               real(dp), dimension(al(1):al(2),2), intent(in) A,
               real(dp), dimension(bl(1):bl(2),2), intent(in) B,
               real(dp), dimension(cl(1):cl(2),2), intent(inout) C )
```

Calculate multiplication through shifting of fourier modes A and B into C.

This works by calculating $[\sum_A (\alpha_A \cos m_A\theta + \beta_A \sin m_A\theta)]\,[\sum_B (\alpha_B \cos m_B\theta + \beta_B \sin m_B\theta)] = [\sum_C (\alpha_C \cos m_C\theta + \beta_C \sin$

where the α and β factors are provide in A, B and C.

This then boils down to finding the four combinations of cosines and sines.

Note

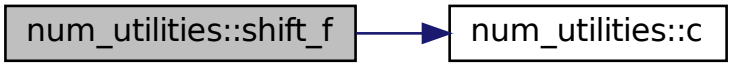
Modes that are larger than what C can hold are thrown away.

Parameters

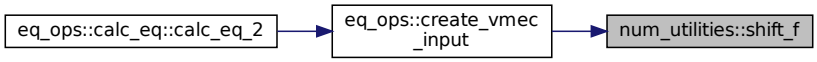
in	*al*	limits on A mode numbers
in	*bl*	limits on B mode numbers
in	*cl*	limits on C mode numbers
in	*a*	input
in	*b*	input
in,out	*c*	result

Definition at line 2696 of file num_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.26.2.15 solve_vand()

```
subroutine, public num_utilities::solve_vand (
               integer, intent(in) n,
               real(dp), dimension(n), intent(in) a,
               real(dp), dimension(n), intent(in) b,
               real(dp), dimension(n), intent(inout) x,
               logical, intent(in), optional transp )
```

Solve a Vandermonde system $\overline{\mathrm{A}}\,\vec{X} = \vec{B}$.

The Vandermonde matrix has the form

$$A = \begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^n \\ 1 & a_2 & a_2^2 & \cdots & a_2^n \\ 1 & a_3 & a_3^2 & \cdots & a_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \cdots & a_n^n \end{pmatrix}$$

See also

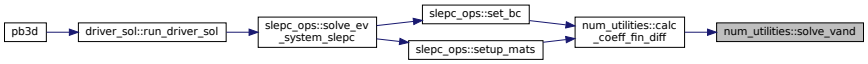
Adapted from two routines dvand and pvand in https://people.sc.fsu.edu/~jburkardt/f_src/vandermonde/vandermonde.html based on the Björk-Pereyra Algorithm [2].

Parameters

in	*n*	matrix size
in	*a*	parameters of Vandermonde matrix
in	*b*	right-hand side
in,out	*x*	solution
in	*transp*	transposed Vandermonde matrix

Definition at line 2751 of file num_utilities.f90.

Here is the caller graph for this function:



B.26.3 Variable Documentation

B.26.3.1 d

```
integer, dimension(:,:,:), allocatable, public num_utilities::d
```

1-D array indices of derivatives

Definition at line 23 of file num_utilities.f90.

B.26.3.2 debug_calc_coeff_fin_diff

```
logical, public num_utilities::debug_calc_coeff_fin_diff = .false.
```

plot debug information for calc_coeff_fin_diff()

Note

Debug version only

Definition at line 28 of file num_utilities.f90.

B.26.3.3 debug_con2dis_reg

```
logical, public num_utilities::debug_con2dis_reg = .false.
```

plot debug information for con2dis_reg()

Note

Debug version only

Definition at line 27 of file num_utilities.f90.

B.26.3.4 f

```
integer, dimension(:,:), allocatable, public num_utilities::f
```

1-D array indices of Fourier mode combination indices

Definition at line 25 of file num_utilities.f90.

B.26.3.5 m

```
integer, dimension(:,:), allocatable, public num_utilities::m
```

1-D array indices of metric indices

Definition at line 24 of file num_utilities.f90.

B.27 num_vars Module Reference

Numerical variables used by most other modules.

Variables

- integer, parameter, public dp = REAL64
  double precision
- integer, parameter, public dpi = INT64
  double precision
- real(dp), parameter, public weight_dp = 0.008
  size of double precision in kB
- integer, parameter, public max_str_ln = 120
  maximum length of strings
- integer, parameter, public max_name_ln = 30
  maximum length of filenames
- integer, parameter, public max_deriv = 2
  highest derivatives for metric factors in Flux coords.
- integer, public prog_style
  program style (1: PB3D, 2: PB3D_POST)
- character(len=4), public prog_name
  name of program, used for info
- character(len=3), parameter, public output_name = 'out'
  name of output file
- character(len=14), parameter, public shell_commands_name = 'shell_commands'
  name of shell commands file
- character(len=9), parameter, public mem_usage_name = 'mem_usage'
  name of memory usage file
- integer, public mem_usage_count
  counter for memory usage output
- real(dp), parameter, public prog_version = 2.45_dp
  version number
- real(dp), parameter, public min_pb3d_version = 2.43_dp
  minimum PB3D version for POST
- logical, public debug_version = .true.
  debug version used
- integer, public rank
  MPI rank.
- integer, public n_procs
  nr. of MPI processes
- integer, public sol_n_procs
  nr. of MPI processes for solution with SLEPC
- integer(kind=8), public time_start
  start time of simulation
- real(dp), public max_tot_mem
  maximum total memory for all processes [MB]
- real(dp), public max_x_mem
  maximum memory for perturbation calculations for all processes [MB]
- integer, dimension(:,:), allocatable, public x_jobs_lims
  data about X jobs: [$\min_k$, $\max_k$, $\min_m$, $\max_m$] for all jobs
- integer, dimension(:,:), allocatable, public eq_jobs_lims
  data about eq jobs: [$\min_\theta$, $\max_\theta$] for all jobs
- integer, public x_job_nr
  nr. of X job
- integer, public eq_job_nr

nr. of eq job

- real(dp), parameter, public mem_scale_fac = 6.0

  empirical scale factor of memory to calculate eq compared to just storing it
- real(dp), parameter, public pi =4_dp∗datan(1.0_dp)

  π
- real(dp), parameter, public mu_0_original = 4E-7_dp∗pi

  permeability of free space
- complex(dp), parameter, public iu = (0, 1)

  complex unit
- integer, public ev_style

  determines the method used for solving an EV problem
- integer, public eq_style

  either 1 (VMEC) or 2 (HELENA)
- integer, public rho_style

  style for equilibrium density profile
- integer, public u_style

  style for calculation of U (1: ord.2, 2: ord.1, 1: ord.0)
- integer, public norm_style

  style for normalization
- integer, public k_style

  style for kinetic energy
- integer, dimension(2), public bc_style

  style for BC left and right
- integer, public x_style

  style for secondary mode numbers (1: prescribed, 2: fast)
- integer, public matrix_slepc_style

  style for matrix storage (1: sparse, 2: shell)
- integer, public solver_slepc_style

  style for solver (1: Krylov-Schur, 2: GD)
- integer, public post_style

  style for POST (1: extended grid, 2: B-aligned grid)
- integer, public x_grid_style

  style for normal component of X grid (1: eq, 2: sol, 3: enriched)
- integer, public alpha_style

  style for alpha (1: one field line, many turns, 2: many field lines, one turn)
- integer, public max_it_slepc

  maximum nr. of iterations for SLEPC
- logical, public plot_resonance

  whether to plot the q-profile or iota-profile with resonances
- logical, public plot_magn_grid

  whether to plot the grid in real coordinates
- logical, public plot_b

  whether to plot the magnetic field in real coordinates
- logical, public plot_j

  whether to plot the current in real coordinates
- logical, public plot_flux_q

  whether to plot flux quantities in real coordinates
- logical, public plot_kappa

  whether to plot curvature
- logical, public plot_sol_xi

  whether to plot plasma perturbation of solution in POST

- logical, public plot_sol_q
  whether to plot magnetic perturbation of solution in POST
- logical, public plot_vac_pot
  whether to plot vacuum potential in POST
- logical, public plot_e_rec
  whether to plot energy reconstruction in POST
- logical, public ltest
  whether or not to call the testing routines
- logical, public use_pol_flux_e
  whether poloidal flux is used in E coords.
- logical, public use_pol_flux_f
  whether poloidal flux is used in F coords.
- logical, public use_normalization
  whether to use normalization or not
- real(dp), public ev_bc
  value of artificial Eigenvalue for boundary condition
- real(dp), public ev_guess
  first guess for eigenvalue
- real(dp), dimension(:), allocatable, public tol_slepc
  tolerance for SLEPC for different Richardson levels
- real(dp), public max_njq_change
  maximum change of prim. mode number times saf. fac. / rot. transf. when using X_style 2 (fast)
- integer, public norm_disc_prec_eq
  precision for normal discretization for equilibrium
- integer, public norm_disc_prec_x
  precision for normal discretization for perturbation
- integer, public norm_disc_prec_sol
  precision for normal discretization for solution
- integer, public norm_disc_style_sol
  style for normal discretization for solution (1: central fin. diff., 2: left fin. diff.)
- integer, public magn_int_style
  style for magnetic integrals (1: trapezoidal, 2: Simpson 3/8)
- integer, public max_it_rich
  number of levels for Richardson extrapolation
- real(dp), public tol_rich
  tolerance for Richardson extrapolation
- integer, public max_it_zero
  maximum number of iterations to find zeros
- integer, public max_nr_backtracks_hh
  maximum number of backtracks for Householder, relax. factors
- real(dp), public tol_zero
  tolerance for zeros
- real(dp), public tol_norm
  tolerance for normal range (normalized to 0..1)
- integer, parameter, public ex_max_size = 10000
  maximum size of matrices for external plot
- character(len=max_str_ln), public eq_name
  name of equilibrium file from VMEC or HELENA
- character(len=max_str_ln), public pb3d_name
  name of PB3D output file
- logical, public no_plots = .false.

no plots made

- logical, public jump_to_sol = .false.
  jump to solution
- logical, public export_hel = .false.
  export HELENA
- logical, public plot_vmec_modes = .false.
  plot VMEC modes
- logical, public invert_top_bottom_h = .false.
  invert top and bottom for HELENA equilibria
- logical, public no_output = .false.
  no output shown
- logical, public post_output_full = .false.
  POST has output on full grids.
- logical, public post_output_sol = .false.
  POST has outputs of solution.
- logical, public do_execute_command_line = .false.
  call "execute_command_line" inside program
- logical, public print_mem_usage = .false.
  print memory usage is printed
- logical, public swap_angles = .false.
  swap angles theta and zeta in plots (only for POST)
- logical, public compare_tor_pos = .false.
  compare quantities at toroidal positions (only for POST)
- logical, public retain_all_sol
  retain also faulty solutions
- character(len=5), public plot_dir = 'Plots'
  directory where to save plots
- character(len=7), public script_dir = 'Scripts'
  directory where to save scripts for plots
- character(len=4), public data_dir = 'Data'
  directory where to save data for plots
- integer, public n_theta_plot
  nr. of poloidal points in plot
- integer, public n_zeta_plot
  nr. of toroidal points in plot
- integer, dimension(2), public n_vac_plot
  nr. of points in R and Z in vacuum
- real(dp), public min_theta_plot
  min. of `theta_plot`
- real(dp), public max_theta_plot
  max. of `theta_plot`
- real(dp), public min_zeta_plot
  min. of `zeta_plot`
- real(dp), public max_zeta_plot
  max. of `zeta_plot`
- real(dp), public min_r_plot
  min. of `r_plot`
- real(dp), public max_r_plot
  max. of `r_plot`
- real(dp), public min_rvac_plot
  min. of `R` in which to plot vacuum

- real(dp), public max_rvac_plot
  max. of R in which to plot vacuum
- real(dp), public min_zvac_plot
  min. of Z in which to plot vacuum
- real(dp), public max_zvac_plot
  max. of Z in which to plot vacuum
- integer, public plot_grid_style
  style for POST plot grid (0: 3-D plots, 1: slab plots, 2: slab plots with folding, 3: straight cylinder))
- integer, public n_sol_requested
  how many solutions requested
- integer, dimension(4), public n_sol_plotted
  how many solutions to be plot (first unstable, last unstable, first stable, last stable)
- integer, dimension(2), public plot_size
  size of plot in inches
- integer, public rich_restart_lvl
  starting Richardson level (0: none [default])
- character(len=max_str_ln), public input_name
  will hold the full name of the input file
- integer, public ex_plot_style
  external plot style (1: GNUPlot, 2: Bokeh for 2D, Mayavi for 3D)
- real(dp), public pert_mult_factor_post
  factor with which to multiply perturbation strength for POST
- real(dp), dimension(2), public rz_0
  origin of geometrical poloidal coordinate
- integer, parameter, public input_i = 50
  file number of input file
- integer, parameter, public eq_i = 51
  file number of equilibrium file from VMEC or HELENA
- integer, parameter, public pb3d_i = 52
  file number of PB3D output file
- integer, parameter, public output_i = 53
  file number of output file
- integer, parameter, public shell_commands_i = 54
  file number of shell commands file
- integer, parameter, public mem_usage_i = 55
  file number of memory usage file
- integer, parameter, public output_ev_i = 56
  file number of output of EV
- integer, parameter, public decomp_i = 57
  file number of output of EV decomposition
- integer, parameter, public hel_export_i = 59
  file number of output of HELENA equilibrium export file
- integer, parameter, public hel_pert_i = 58
  file number of HELENA equilibrium perturbation file
- integer, parameter, public prop_b_tor_i = 60
  file number of B_ϕ proportionality factor file

B.27.1 Detailed Description

Numerical variables used by most other modules.

B.27.2 Variable Documentation

B.27.2.1 alpha_style

```
integer, public num_vars::alpha_style
```

style for alpha (1: one field line, many turns, 2: many field lines, one turn)

Definition at line 100 of file num_vars.f90.

B.27.2.2 bc_style

```
integer, dimension(2), public num_vars::bc_style
```

style for BC left and right

Definition at line 94 of file num_vars.f90.

B.27.2.3 compare_tor_pos

```
logical, public num_vars::compare_tor_pos = .false.
```

compare quantities at toroidal positions (only for POST)

Definition at line 151 of file num_vars.f90.

B.27.2.4 data_dir

```
character(len=4), public num_vars::data_dir = 'Data'
```

directory where to save data for plots

Definition at line 155 of file num_vars.f90.

B.27.2.5 debug_version

```
logical public num_vars::debug_version = .true.
```

debug version used

not debug version

Definition at line 62 of file num_vars.f90.

B.27.2.6 decomp_i

```
integer, parameter, public num_vars::decomp_i = 57
```

file number of output of EV decomposition

Definition at line 189 of file num_vars.f90.

B.27.2.7 do_execute_command_line

```
logical, public num_vars::do_execute_command_line = .false.
```

call ”execute_command_line” inside program

Definition at line 148 of file num_vars.f90.

B.27.2.8 dp

```
integer, parameter, public num_vars::dp = REAL64
```

double precision

Definition at line 46 of file num_vars.f90.

B.27.2.9 dpi

```
integer, parameter, public num_vars::dpi = INT64
```

double precision

Definition at line 48 of file num_vars.f90.

B.27.2.10 eq_i

```
integer, parameter, public num_vars::eq_i = 51
```

file number of equilibrium file from VMEC or HELENA

Definition at line 183 of file num_vars.f90.

B.27.2.11 eq_job_nr

```
integer, public num_vars::eq_job_nr
```

nr. of eq job

Definition at line 79 of file num_vars.f90.

B.27.2.12 eq_jobs_lims

```
integer, dimension(:,:), allocatable, public num_vars::eq_jobs_lims
```

data about eq jobs: $[\min_\theta, \max_\theta]$ for all jobs

Definition at line 77 of file num_vars.f90.

B.27.2.13 eq_name

```
character(len=max_str_ln), public num_vars::eq_name
```

name of equilibrium file from VMEC or HELENA

Definition at line 138 of file num_vars.f90.

B.27.2.14 eq_style

```
integer, public num_vars::eq_style
```

either 1 (VMEC) or 2 (HELENA)

Definition at line 89 of file num_vars.f90.

B.27.2.15 ev_bc

```
real(dp), public num_vars::ev_bc
```

value of artificial Eigenvalue for boundary condition

Definition at line 116 of file num_vars.f90.

B.27.2.16 ev_guess

```
real(dp), public num_vars::ev_guess
```

first guess for eigenvalue

Definition at line 117 of file num_vars.f90.

B.27.2.17 ev_style

```
integer, public num_vars::ev_style
```

determines the method used for solving an EV problem

Definition at line 88 of file num_vars.f90.

B.27.2.18 ex_max_size

```
integer, parameter, public num_vars::ex_max_size = 10000
```

maximum size of matrices for external plot

Definition at line 137 of file num_vars.f90.

B.27.2.19 ex_plot_style

```
integer, public num_vars::ex_plot_style
```

external plot style (1: GNUPlot, 2: Bokeh for 2D, Mayavi for 3D)

Definition at line 175 of file num_vars.f90.

B.27.2.20 export_hel

```
logical, public num_vars::export_hel = .false.
```

export HELENA

Definition at line 142 of file num_vars.f90.

B.27.2.21 hel_export_i

```
integer, parameter, public num_vars::hel_export_i = 59
```

file number of output of HELENA equilibrium export file

Definition at line 190 of file num_vars.f90.

B.27.2.22 hel_pert_i

```
integer, parameter, public num_vars::hel_pert_i = 58
```

file number of HELENA equilibrium perturbation file

Definition at line 191 of file num_vars.f90.

B.27.2.23 input_i

```
integer, parameter, public num_vars::input_i = 50
```

file number of input file

Definition at line 182 of file num_vars.f90.

B.27.2.24 input_name

```
character(len=max_str_ln), public num_vars::input_name
```

will hold the full name of the input file

Definition at line 174 of file num_vars.f90.

B.27.2.25 invert_top_bottom_h

```
logical, public num_vars::invert_top_bottom_h = .false.
```

invert top and bottom for HELENA equilibria

Definition at line 144 of file num_vars.f90.

B.27.2.26 iu

```
complex(dp), parameter, public num_vars::iu = (0, 1)
```

complex unit

Definition at line 85 of file num_vars.f90.

B.27.2.27 jump_to_sol

```
logical, public num_vars::jump_to_sol = .false.
```

jump to solution

Definition at line 141 of file num_vars.f90.

B.27.2.28 k_style

```
integer, public num_vars::k_style
```

style for kinetic energy

Definition at line 93 of file num_vars.f90.

B.27.2.29 ltest

```
logical, public num_vars::ltest
```

whether or not to call the testing routines

Definition at line 112 of file num_vars.f90.

B.27.2.30 magn_int_style

```
integer, public num_vars::magn_int_style
```

style for magnetic integrals (1: trapezoidal, 2: Simpson 3/8)

Definition at line 124 of file num_vars.f90.

B.27.2.31 matrix_slepc_style

```
integer, public num_vars::matrix_slepc_style
```

style for matrix storage (1: sparse, 2: shell)

Definition at line 96 of file num_vars.f90.

B.27.2.32 max_deriv

```
integer, parameter, public num_vars::max_deriv = 2
```

highest derivatives for metric factors in Flux coords.

Definition at line 52 of file num_vars.f90.

B.27.2.33 max_it_rich

```
integer, public num_vars::max_it_rich
```

number of levels for Richardson extrapolation

Definition at line 127 of file num_vars.f90.

B.27.2.34 max_it_slepc

```
integer, public num_vars::max_it_slepc
```

maximum nr. of iterations for SLEPC

Definition at line 101 of file num_vars.f90.

B.27.2.35 max_it_zero

```
integer, public num_vars::max_it_zero
```

maximum number of iterations to find zeros

Definition at line 131 of file num_vars.f90.

B.27.2.36 max_name_ln

```
integer, parameter, public num_vars::max_name_ln = 30
```

maximum length of filenames

Definition at line 51 of file num_vars.f90.

B.27.2.37 max_njq_change

```
real(dp), public num_vars::max_njq_change
```

maximum change of prim. mode number times saf. fac. / rot. transf. when using X_style 2 (fast)

Definition at line 119 of file num_vars.f90.

B.27.2.38 max_nr_backtracks_hh

```
integer, public num_vars::max_nr_backtracks_hh
```

maximum number of backtracks for Householder, relax. factors

Definition at line 132 of file num_vars.f90.

B.27.2.39 max_r_plot

```
real(dp), public num_vars::max_r_plot
```

max. of `r_plot`

Definition at line 164 of file num_vars.f90.

B.27.2.40 max_rvac_plot

```
real(dp), public num_vars::max_rvac_plot
```

max. of `R` in which to plot vacuum

Definition at line 166 of file num_vars.f90.

B.27.2.41 max_str_ln

```
integer, parameter, public num_vars::max_str_ln = 120
```

maximum length of strings

Definition at line 50 of file num_vars.f90.

B.27.2.42 max_theta_plot

```
real(dp), public num_vars::max_theta_plot
```

max. of `theta_plot`

Definition at line 160 of file num_vars.f90.

B.27.2.43 max_tot_mem

```
real(dp), public num_vars::max_tot_mem
```

maximum total memory for all processes [MB]

Definition at line 74 of file num_vars.f90.

B.27.2.44 max_x_mem

```
real(dp), public num_vars::max_x_mem
```

maximum memory for perturbation calculations for all processes [MB]

Definition at line 75 of file num_vars.f90.

B.27.2.45 max_zeta_plot

```
real(dp), public num_vars::max_zeta_plot
```

max. of `zeta_plot`

Definition at line 162 of file num_vars.f90.

B.27.2.46 max_zvac_plot

```
real(dp), public num_vars::max_zvac_plot
```

max. of `Z` in which to plot vacuum

Definition at line 168 of file num_vars.f90.

B.27.2.47 mem_scale_fac

```
real(dp), parameter, public num_vars::mem_scale_fac = 6.0
```

empirical scale factor of memory to calculate eq compared to just storing it

Definition at line 80 of file num_vars.f90.

B.27.2.48 mem_usage_count

```
integer, public num_vars::mem_usage_count
```

counter for memory usage output

Definition at line 58 of file num_vars.f90.

B.27.2.49 mem_usage_i

```
integer, parameter, public num_vars::mem_usage_i = 55
```

file number of memory usage file

Definition at line 187 of file num_vars.f90.

B.27.2.50 mem_usage_name

```
character(len=9), parameter, public num_vars::mem_usage_name = 'mem_usage'
```

name of memory usage file

Definition at line 57 of file num_vars.f90.

B.27.2.51 min_pb3d_version

```
real(dp), parameter, public num_vars::min_pb3d_version = 2.43_dp
```

minimum PB3D version for POST

Definition at line 60 of file num_vars.f90.

B.27.2.52 min_r_plot

```
real(dp), public num_vars::min_r_plot
```

min. of `r_plot`

Definition at line 163 of file num_vars.f90.

B.27.2.53 min_rvac_plot

```
real(dp), public num_vars::min_rvac_plot
```

min. of `R` in which to plot vacuum

Definition at line 165 of file num_vars.f90.

B.27.2.54 min_theta_plot

```
real(dp), public num_vars::min_theta_plot
```

min. of `theta_plot`

Definition at line 159 of file num_vars.f90.

B.27.2.55 min_zeta_plot

```
real(dp), public num_vars::min_zeta_plot
```

min. of `zeta_plot`

Definition at line 161 of file num_vars.f90.

B.27.2.56 min_zvac_plot

```
real(dp), public num_vars::min_zvac_plot
```

min. of `Z` in which to plot vacuum

Definition at line 167 of file num_vars.f90.

B.27.2.57 mu_0_original

```
real(dp), parameter, public num_vars::mu_0_original = 4E-7_dp*pi
```

permeability of free space

Definition at line 84 of file num_vars.f90.

B.27.2.58 n_procs

```
integer, public num_vars::n_procs
```

nr. of MPI processes

Definition at line 69 of file num_vars.f90.

B.27.2.59 n_sol_plotted

```
integer, dimension(4), public num_vars::n_sol_plotted
```

how many solutions to be plot (first unstable, last unstable, first stable, last stable)

Definition at line 171 of file num_vars.f90.

B.27.2.60 n_sol_requested

```
integer, public num_vars::n_sol_requested
```

how many solutions requested

Definition at line 170 of file num_vars.f90.

B.27.2.61 n_theta_plot

```
integer, public num_vars::n_theta_plot
```

nr. of poloidal points in plot

Definition at line 156 of file num_vars.f90.

B.27.2.62 n_vac_plot

```
integer, dimension(2), public num_vars::n_vac_plot
```

nr. of points in R and Z in vacuum

Definition at line 158 of file num_vars.f90.

B.27.2.63 n_zeta_plot

```
integer, public num_vars::n_zeta_plot
```

nr. of toroidal points in plot

Definition at line 157 of file num_vars.f90.

B.27.2.64 no_output

```
logical, public num_vars::no_output = .false.
```

no output shown

Definition at line 145 of file num_vars.f90.

B.27.2.65 no_plots

```
logical, public num_vars::no_plots = .false.
```

no plots made

Definition at line 140 of file num_vars.f90.

B.27.2.66 norm_disc_prec_eq

```
integer, public num_vars::norm_disc_prec_eq
```

precision for normal discretization for equilibrium

Definition at line 120 of file num_vars.f90.

B.27.2.67 norm_disc_prec_sol

```
integer, public num_vars::norm_disc_prec_sol
```

precision for normal discretization for solution

Definition at line 122 of file num_vars.f90.

B.27.2.68 norm_disc_prec_x

```
integer, public num_vars::norm_disc_prec_x
```

precision for normal discretization for perturbation

Definition at line 121 of file num_vars.f90.

B.27.2.69 norm_disc_style_sol

```
integer, public num_vars::norm_disc_style_sol
```

style for normal discretization for solution (1: central fin. diff., 2: left fin. diff.)

Definition at line 123 of file num_vars.f90.

B.27.2.70 norm_style

```
integer, public num_vars::norm_style
```

style for normalization

Definition at line 92 of file num_vars.f90.

B.27.2.71 output_ev_i

```
integer, parameter, public num_vars::output_ev_i = 56
```

file number of output of EV

Definition at line 188 of file num_vars.f90.

B.27.2.72 output_i

```
integer, parameter, public num_vars::output_i = 53
```

file number of output file

Definition at line 185 of file num_vars.f90.

B.27.2.73 output_name

```
character(len=3), parameter, public num_vars::output_name = 'out'
```

name of output file

Definition at line 55 of file num_vars.f90.

B.27.2.74 pb3d_i

```
integer, parameter, public num_vars::pb3d_i = 52
```

file number of PB3D output file

Definition at line 184 of file num_vars.f90.

B.27.2.75 pb3d_name

```
character(len=max_str_ln), public num_vars::pb3d_name
```

name of PB3D output file

Definition at line 139 of file num_vars.f90.

B.27.2.76 pert_mult_factor_post

```
real(dp), public num_vars::pert_mult_factor_post
```

factor with which to multiply perturbation strength for POST

Definition at line 176 of file num_vars.f90.

B.27.2.77 pi

```
real(dp), parameter, public num_vars::pi =4_dp*datan(1.0_dp)
```

π

Definition at line 83 of file num_vars.f90.

B.27.2.78 plot_b

```
logical, public num_vars::plot_b
```

whether to plot the magnetic field in real coordinates

Definition at line 104 of file num_vars.f90.

B.27.2.79 plot_dir

```
character(len=5), public num_vars::plot_dir = 'Plots'
```

directory where to save plots

Definition at line 153 of file num_vars.f90.

B.27.2.80 plot_e_rec

```
logical, public num_vars::plot_e_rec
```

whether to plot energy reconstruction in POST

Definition at line 111 of file num_vars.f90.

B.27.2.81 plot_flux_q

```
logical, public num_vars::plot_flux_q
```

whether to plot flux quantities in real coordinates

Definition at line 106 of file num_vars.f90.

B.27.2.82 plot_grid_style

```
integer, public num_vars::plot_grid_style
```

style for POST plot grid (0: 3-D plots, 1: slab plots, 2: slab plots with folding, 3: straight cylinder))

Definition at line 169 of file num_vars.f90.

B.27.2.83 plot_j

```
logical, public num_vars::plot_j
```

whether to plot the current in real coordinates

Definition at line 105 of file num_vars.f90.

B.27.2.84 plot_kappa

```
logical, public num_vars::plot_kappa
```

whether to plot curvature

Definition at line 107 of file num_vars.f90.

B.27.2.85 plot_magn_grid

```
logical, public num_vars::plot_magn_grid
```

whether to plot the grid in real coordinates

Definition at line 103 of file num_vars.f90.

B.27.2.86 plot_resonance

```
logical, public num_vars::plot_resonance
```

whether to plot the q-profile or iota-profile with resonances

Definition at line 102 of file num_vars.f90.

B.27.2.87 plot_size

```
integer, dimension(2), public num_vars::plot_size
```

size of plot in inches

Definition at line 172 of file num_vars.f90.

B.27.2.88 plot_sol_q

```
logical, public num_vars::plot_sol_q
```

whether to plot magnetic perturbation of solution in POST

Definition at line 109 of file num_vars.f90.

B.27.2.89 plot_sol_xi

```
logical, public num_vars::plot_sol_xi
```

whether to plot plasma perturbation of solution in POST

Definition at line 108 of file num_vars.f90.

B.27.2.90 plot_vac_pot

```
logical, public num_vars::plot_vac_pot
```

whether to plot vacuum potential in POST

Definition at line 110 of file num_vars.f90.

B.27.2.91 plot_vmec_modes

```
logical, public num_vars::plot_vmec_modes = .false.
```

plot VMEC modes

Definition at line 143 of file num_vars.f90.

B.27.2.92 post_output_full

```
logical, public num_vars::post_output_full = .false.
```

POST has output on full grids.

Definition at line 146 of file num_vars.f90.

B.27.2.93 post_output_sol

```
logical, public num_vars::post_output_sol = .false.
```

POST has outputs of solution.

Definition at line 147 of file num_vars.f90.

B.27.2.94 post_style

```
integer, public num_vars::post_style
```

style for POST (1: extended grid, 2: B-aligned grid)

Definition at line 98 of file num_vars.f90.

B.27.2.95 print_mem_usage

```
logical, public num_vars::print_mem_usage = .false.
```

print memory usage is printed

Definition at line 149 of file num_vars.f90.

B.27.2.96 prog_name

```
character(len=4), public num_vars::prog_name
```

name of program, used for info

Definition at line 54 of file num_vars.f90.

B.27.2.97 prog_style

```
integer, public num_vars::prog_style
```

program style (1: PB3D, 2: PB3D_POST)

Definition at line 53 of file num_vars.f90.

B.27.2.98 prog_version

```
real(dp), parameter, public num_vars::prog_version = 2.45_dp
```

version number

Definition at line 59 of file num_vars.f90.

B.27.2.99 prop_b_tor_i

```
integer, parameter, public num_vars::prop_b_tor_i = 60
```

file number of B_ϕ proportionality factor file

Definition at line 192 of file num_vars.f90.

B.27.2.100 rank

```
integer, public num_vars::rank
```

MPI rank.

Definition at line 68 of file num_vars.f90.

B.27.2.101 retain_all_sol

```
logical, public num_vars::retain_all_sol
```

retain also faulty solutions

Definition at line 152 of file num_vars.f90.

B.27.2.102 rho_style

```
integer, public num_vars::rho_style
```

style for equilibrium density profile

Definition at line 90 of file num_vars.f90.

B.27.2.103 rich_restart_lvl

```
integer, public num_vars::rich_restart_lvl
```

starting Richardson level (0: none [default])

Definition at line 173 of file num_vars.f90.

B.27.2.104 rz_0

```
real(dp), dimension(2), public num_vars::rz_0
```

origin of geometrical poloidal coordinate

Definition at line 179 of file num_vars.f90.

B.27.2.105 script_dir

```
character(len=7), public num_vars::script_dir = 'Scripts'
```

directory where to save scripts for plots

Definition at line 154 of file num_vars.f90.

B.27.2.106 shell_commands_i

```
integer, parameter, public num_vars::shell_commands_i = 54
```

file number of shell commands file

Definition at line 186 of file num_vars.f90.

B.27.2.107 shell_commands_name

```
character(len=14), parameter, public num_vars::shell_commands_name = 'shell_commands'
```

name of shell commands file

Definition at line 56 of file num_vars.f90.

B.27.2.108 sol_n_procs

```
integer, public num_vars::sol_n_procs
```

nr. of MPI processes for solution with SLEPC

Definition at line 70 of file num_vars.f90.

B.27.2.109 solver_slepc_style

```
integer, public num_vars::solver_slepc_style
```

style for solver (1: Krylov-Schur, 2: GD)

Definition at line 97 of file num_vars.f90.

B.27.2.110 swap_angles

```
logical, public num_vars::swap_angles = .false.
```

swap angles theta and zeta in plots (only for POST)

Definition at line 150 of file num_vars.f90.

B.27.2.111 time_start

```
integer(kind=8), public num_vars::time_start
```

start time of simulation

Definition at line 71 of file num_vars.f90.

B.27.2.112 tol_norm

```
real(dp), public num_vars::tol_norm
```

tolerance for normal range (normalized to 0..1)

Definition at line 134 of file num_vars.f90.

B.27.2.113 tol_rich

```
real(dp), public num_vars::tol_rich
```

tolerance for Richardson extrapolation

Definition at line 128 of file num_vars.f90.

B.27.2.114 tol_slepc

```
real(dp), dimension(:), allocatable, public num_vars::tol_slepc
```

tolerance for SLEPC for different Richardson levels

Definition at line 118 of file num_vars.f90.

B.27.2.115 tol_zero

```
real(dp), public num_vars::tol_zero
```

tolerance for zeros

Definition at line 133 of file num_vars.f90.

B.27.2.116 u_style

```
integer, public num_vars::u_style
```

style for calculation of U (1: ord.2, 2: ord.1, 1: ord.0)

Definition at line 91 of file num_vars.f90.

B.27.2.117 use_normalization

```
logical, public num_vars::use_normalization
```

whether to use normalization or not

Definition at line 115 of file num_vars.f90.

B.27.2.118 use_pol_flux_e

```
logical, public num_vars::use_pol_flux_e
```

whether poloidal flux is used in E coords.

Definition at line 113 of file num_vars.f90.

B.27.2.119 use_pol_flux_f

```
logical, public num_vars::use_pol_flux_f
```

whether poloidal flux is used in F coords.

Definition at line 114 of file num_vars.f90.

B.27.2.120 weight_dp

```
real(dp), parameter, public num_vars::weight_dp = 0.008
```

size of double precision in kB

Definition at line 49 of file num_vars.f90.

B.27.2.121 x_grid_style

```
integer, public num_vars::x_grid_style
```

style for normal component of X grid (1: eq, 2: sol, 3: enriched)

Definition at line 99 of file num_vars.f90.

B.27.2.122 x_job_nr

```
integer, public num_vars::x_job_nr
```

nr. of X job

Definition at line 78 of file num_vars.f90.

B.27.2.123 x_jobs_lims

```
integer, dimension(:,:), allocatable, public num_vars::x_jobs_lims
```

data about X jobs: $[\min_k, \max_k, \min_m, \max_m]$ for all jobs

Definition at line 76 of file num_vars.f90.

B.27.2.124 x_style

```
integer, public num_vars::x_style
```

style for secondary mode numbers (1: prescribed, 2: fast)

Definition at line 95 of file num_vars.f90.

B.28 output_ops Module Reference

Operations concerning giving output, on the screen as well as in output files.

Interfaces and Types

- interface plot_hdf5

  Prints variables `vars` with names `var_names` in an HDF5 file with name c file_name and accompanying XDMF file.
- interface print_ex_2d

  Print 2-D output on a file.
- interface print_ex_3d

  Print 3-D output on a file.

Functions/Subroutines

- subroutine, public draw_ex (var_names, draw_name, nplt, draw_dim, plot_on_screen, ex_plot_style, data_name, draw_ops, extra_ops, is_animated, ranges, delay, persistent)

  Use external program to draw a plot.
- subroutine, public plot_diff_hdf5 (A, B, file_name, tot_dim, loc_offset, descr, output_message)

  Takes two input vectors and plots these as well as the relative and absolute difference in a HDF5 file.
- subroutine use_execute_command_line (command, exitstat, cmdstat, cmdmsg)

  Executes command line, or displays a message if disabled.

B.28.1 Detailed Description

Operations concerning giving output, on the screen as well as in output files.

B.28.2 Function/Subroutine Documentation

B.28.2.1 draw_ex()

```
subroutine, public output_ops::draw_ex (
               character(len=*), dimension(:), intent(in) var_names,
               character(len=*), intent(in) draw_name,
               integer, intent(in) nplt,
               integer, intent(in) draw_dim,
               logical, intent(in) plot_on_screen,
               integer, intent(in), optional ex_plot_style,
               character(len=*), intent(in), optional data_name,
               character(len=*), dimension(:), intent(in), optional draw_ops,
               character(len=*), intent(in), optional extra_ops,
               logical, intent(in), optional is_animated,
               real(dp), dimension(:,:), intent(in), optional ranges,
               integer, intent(in), optional delay,
               logical, intent(in), optional persistent )
```

Use external program to draw a plot.

The external programs used are determined by `ex_plot_style`:

- `ex_plot_style` = 1: Use GnuPlot for 2-D and 3-D
- `ex_plot_style` = 1: Use Bokeh in 2-D and Mayavi in 3-D

The output is saved in a file `[draw_name].[ext]` where the extension depends on the plotting style and the dimension, or on the screen, depending on `plot_on_screen`.

If not specified otherwise through data_name, it is assumed that the datafile to be plot is situated in `[draw_name].dat`.

The title(s) of the plot(s) are provided through `var_names` and the number of plots through `nplt`. If there are less names than the number of plots, the first one is taken and appended by the plot number.

Furthermore, `draw_dim` determines whether the plot is to be 2-D, 3-D or 2-D slices in 3-D.

Also, an optional command `extra_ops` can be provided, to provide overal options, as well as draw_ops that specifies the line style for the plots from the file. If `less` draw_ops are provided than plots, they will be cycled.

Finally, there is an option to provide animated plots, but is only available in 2-D. Also, for external plot style 1, no on-screen view is possible. Animations can take optional ranges arguments as well as delay.

Note

`About` draw_dim:

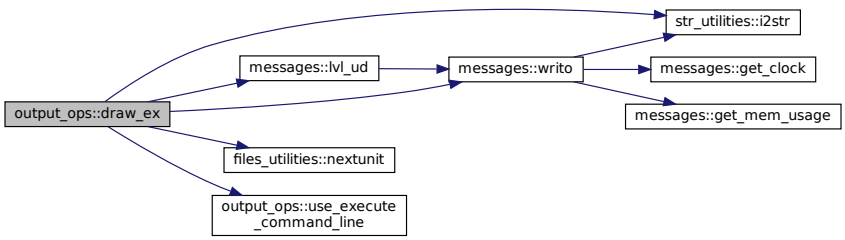
- `draw_dim` = 1: 2-D plot; should be called with the output of output_ops.print_ex_2d(), `nplt` should be correctly set.
- `draw_dim` = 2: 3-D plot; should be called with the output of print_ex_3D. For GNUPlot, `nplt` is ignored, but not for Mayavi, as it needs this information to be able to reconstruct 2D arrays for X, Y and Z.
- `draw_dim` = 3: 2-D plot in 3-D slices; should be called with the output of output_ops.print_ex_2d(), `nplt` should be correctly set.

Parameters

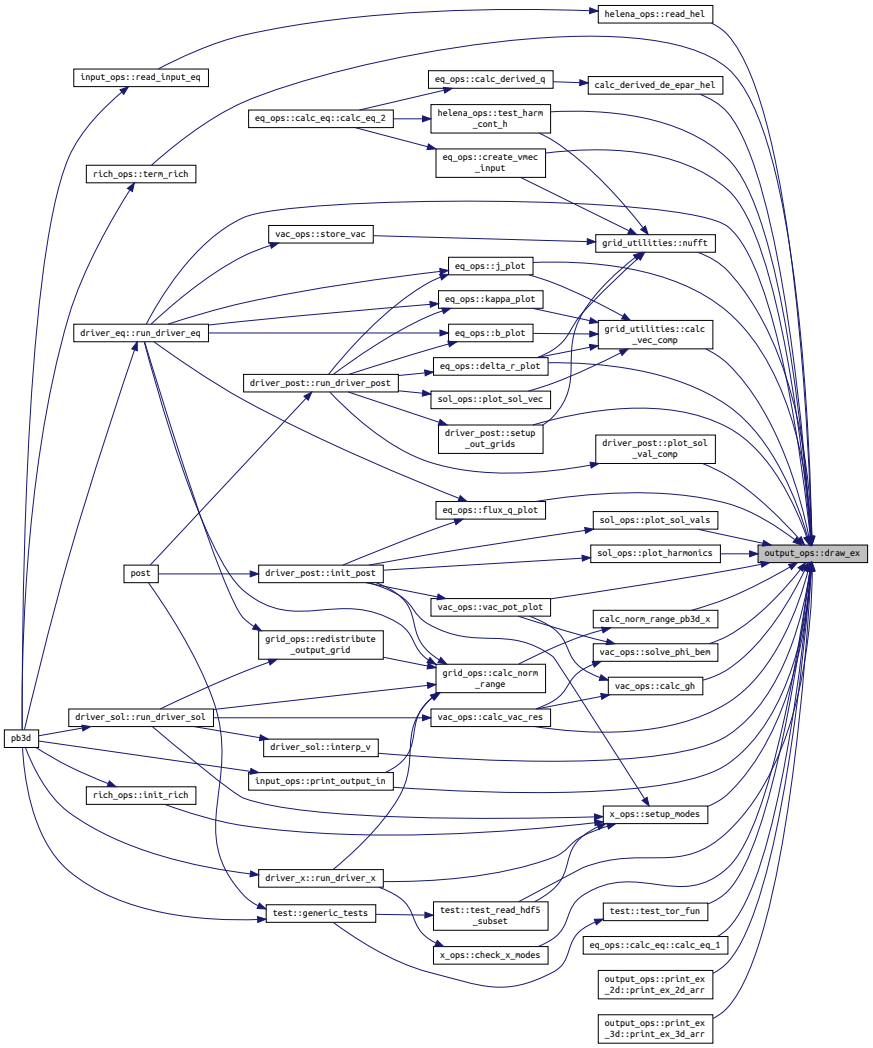
in	*var_names*	name of variables
in	*draw_name*	name of drawing
in	*nplt*	number of plots
in	*draw_dim*	1: 2-D, 2: 3-D, 3: decoupled 3-D
in	*plot_on_screen*	True if on screen, false if in file
in	*ex_plot_style*	alternative external plot style
in	*data_name*	name of data file
in	*draw_ops*	drawing options
in	*extra_ops*	extra options
in	*is_animated*	plot is animated
in	*ranges*	x and y range of animated plot
in	*delay*	time delay between animated plot frames
in	*persistent*	keep on-screen plot open

Definition at line 1079 of file output_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.28.2.2 plot_diff_hdf5()

```
subroutine, public output_ops::plot_diff_hdf5 (
              real(dp), dimension(:,:,:), intent(in) A,
              real(dp), dimension(:,:,:), intent(in) B,
              character(len=*), intent(in) file_name,
              integer, dimension(3), intent(in), optional tot_dim,
              integer, dimension(3), intent(in), optional loc_offset,
              character(len=*), intent(in), optional descr,
              logical, intent(in), optional output_message )
```

Takes two input vectors and plots these as well as the relative and absolute difference in a HDF5 file.

This is similar to a basic version of output_ops.plot_hdf5().

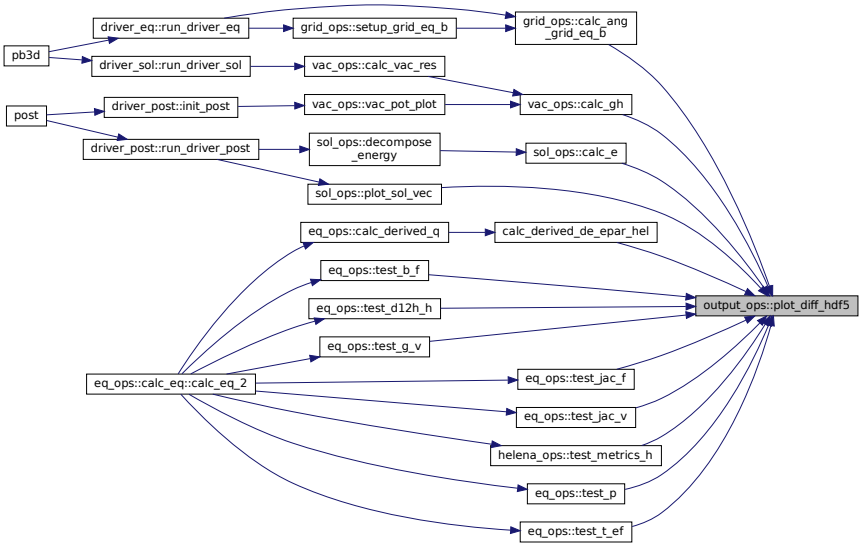
Optionally, an output message can be displayed on screen with the maximum relative and absolute error.

Parameters

in	*a*	vector A
in	*b*	vector B
in	*file_name*	name of plot
in	*tot_dim*	total dimensions of the arrays
in	*loc_offset*	offset of local dimensions
in	*descr*	description
in	*output_message*	whether to display a message or not

Definition at line 1765 of file output_ops.f90.

Here is the caller graph for this function:



B.28.2.3 use_execute_command_line()

```
subroutine output_ops::use_execute_command_line (
               character(len=*), intent(in) command,
               integer, intent(inout), optional exitstat,
               integer, intent(inout), optional cmdstat,
               character(len=*), intent(inout), optional cmdmsg )
```

Executes command line, or displays a message if disabled.

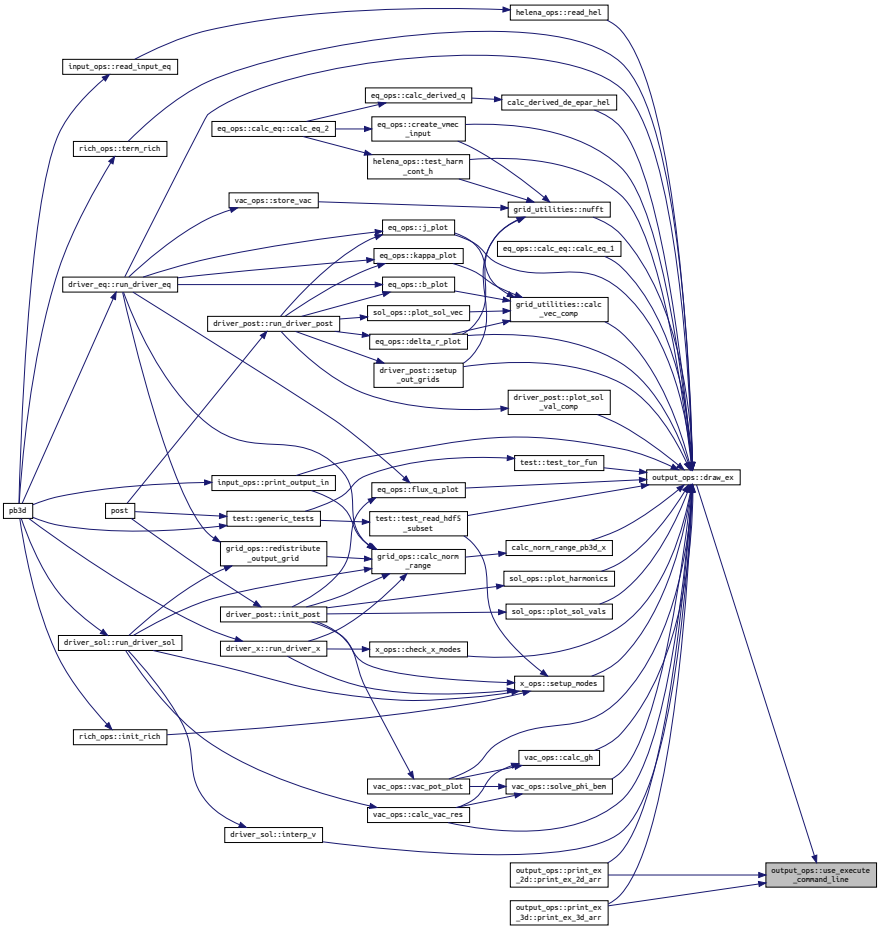
It also keeps a log of all shell commands executed.

Parameters

in	*command*	command to execute
in,out	*exitstat*	exit status
in,out	*cmdstat*	command status
in,out	*cmdmsg*	command message

Definition at line 1914 of file output_ops.f90.

Here is the caller graph for this function:



B.29 pb3d_ops Module Reference

Operations on PB3D output.

Functions/Subroutines

- integer function, public reconstruct_pb3d_in (data_name)
  Reconstructs the input variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_grid (grid, data_name, rich_lvl, tot_rich, lim_pos, grid_↩ limits)
  Reconstructs grid variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_eq_1 (grid_eq, eq, data_name, lim_pos)
  Reconstructs the equilibrium variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_eq_2 (grid_eq, eq, data_name, rich_lvl, tot_rich, lim_pos)
  Reconstructs the equilibrium variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_x_1 (mds, grid_X, X, data_name, rich_lvl, tot_rich, lim_sec↩ _X, lim_pos)
  Reconstructs the vectorial perturbation variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_x_2 (mds, grid_X, X, data_name, rich_lvl, tot_rich, lim_sec↩ _X, lim_pos, is_field_averaged)
  Reconstructs the tensorial perturbation variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_vac (vac, data_name, rich_lvl)
  Reconstructs the vacuum variables from PB3D HDF5 output.
- integer function, public reconstruct_pb3d_sol (mds, grid_sol, sol, data_name, rich_lvl, lim_sec_sol, lim_pos)
  Reconstructs the solution variables from PB3D HDF5 output.
- integer function, public get_pb3d_grid_size (n, grid_name, rich_lvl, tot_rich)
  get grid size

B.29.1 Detailed Description

Operations on PB3D output.

Note

If you have parallel jobs, if the reconstruction routines are called for multiple equilibrium jobs, this can only be done after the last equilibrium job is finished. There are no checks for this.

B.29.2 Function/Subroutine Documentation

B.29.2.1 get_pb3d_grid_size()

```
integer function, public pb3d_ops::get_pb3d_grid_size (
               integer, dimension(3), intent(inout) n,
               character(len=*), intent(in) grid_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional tot_rich )
```

get grid size

Note

`grid_` is added in front the grid_name.

Returns

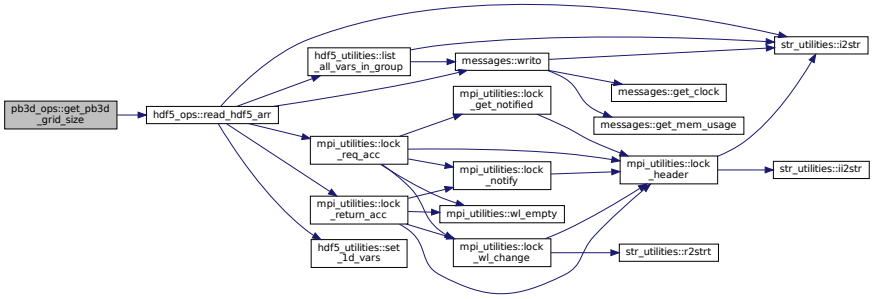
ierr

Parameters

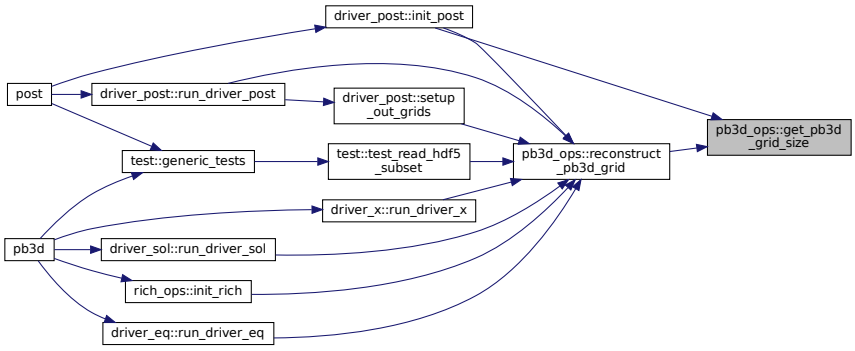
in,out	*n*	n of grid
in	*grid_name*	name of grid
in	*rich_lvl*	Richardson level to reconstruct
in	*tot_rich*	whether to combine with previous Richardson levels

Definition at line 1454 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.2 reconstruct_pb3d_eq_1()

```
integer function, public pb3d_ops::reconstruct_pb3d_eq_1 (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(inout), optional eq,
               character(len=*), intent(in) data_name,
               integer, dimension(1,2), intent(in), optional lim_pos )
```

Reconstructs the equilibrium variables from PB3D HDF5 output.

Furthermore, using `lim_pos`, you can obtain a subset of the data by directly passing its limits to the underlying HDF5 routine. This refers to the position dimensions only. If provided, the normal limits of a divided grid refer to the subset, as in copy_grid().

Returns

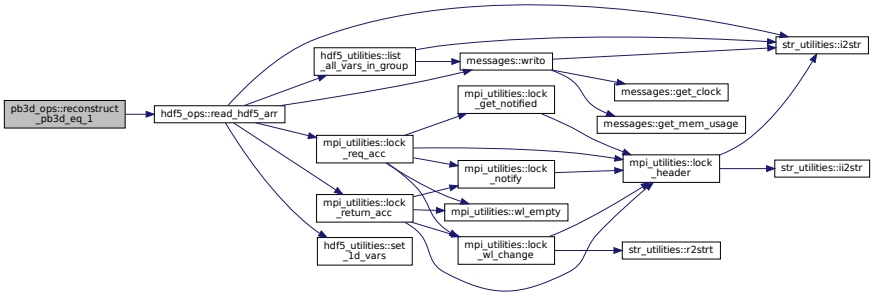
ierr

Parameters

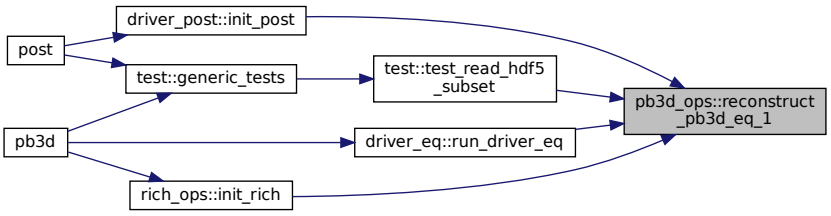
in	*grid_eq*	equilibrium grid
in,out	*eq*	flux equilibrium
in	*data_name*	name to reconstruct
in	*lim_pos*	position limits of subset of data

Definition at line 597 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.3 reconstruct_pb3d_eq_2()

```
integer function, public pb3d_ops::reconstruct_pb3d_eq_2 (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(inout) eq,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
```

```
logical, intent(in), optional tot_rich,
integer, dimension(3,2), intent(in), optional lim_pos )
```

Reconstructs the equilibrium variables from PB3D HDF5 output.

Also, if `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

With `tot_rich` the information from previous Richardson levels can be combined.

Furthermore, using `lim_pos`, you can obtain a subset of the data by directly passing its limits to the underlying HDF5 routine. This refers to the position dimensions only. If provided, the normal limits of a divided grid refer to the subset, as in copy_grid().

Returns

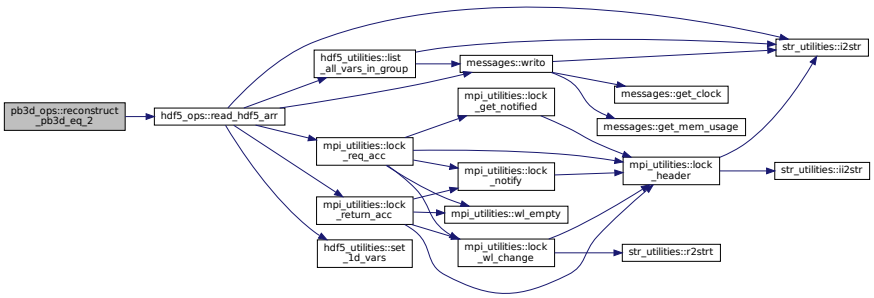
ierr

Parameters

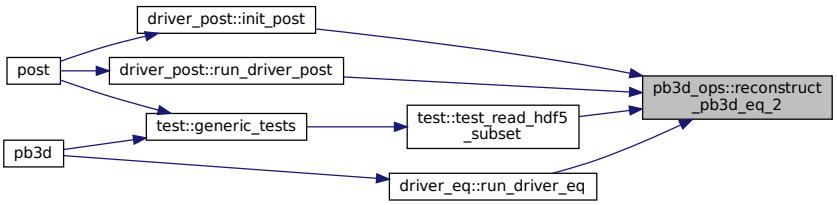
in	*grid_eq*	equilibrium grid
in,out	*eq*	metric equilibrium
in	*data_name*	name to reconstruct
in	*rich_lvl*	Richardson level to reconstruct
in	*tot_rich*	whether to combine with previous Richardson levels
in	*lim_pos*	position limits of subset of data

Definition at line 695 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.4 reconstruct_pb3d_grid()

```
integer function, public pb3d_ops::reconstruct_pb3d_grid (
               type(grid_type), intent(inout) grid,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional tot_rich,
               integer, dimension(3,2), intent(in), optional lim_pos,
               integer, dimension(2), intent(in), optional grid_limits )
```

Reconstructs grid variables from PB3D HDF5 output.

Also, if `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

With `tot_rich` the information from previous Richardson levels can be combined.

Furthermore, using `lim_pos`, you can obtain a subset of the data by directly passing its limits to the underlying HDF5 routine. This refers to the position dimensions only. If provided, the normal limits of a divided grid refer to the subset, as in copy_grid().

Note

1. `grid_` is added in front the data_name.
2. By providing `lim_pos` equal to 0 in the angular dimensions, the angular part can be discarded when reconstructing the grid.

Returns

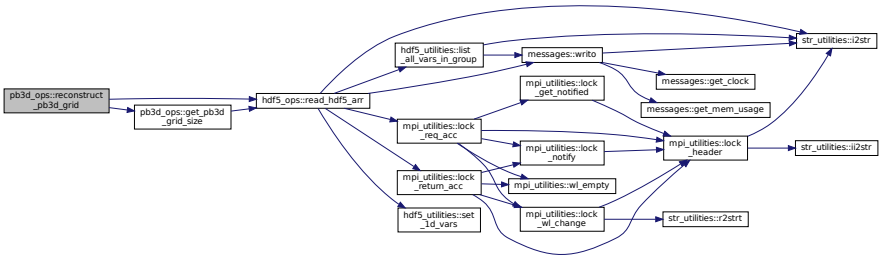
ierr

Parameters

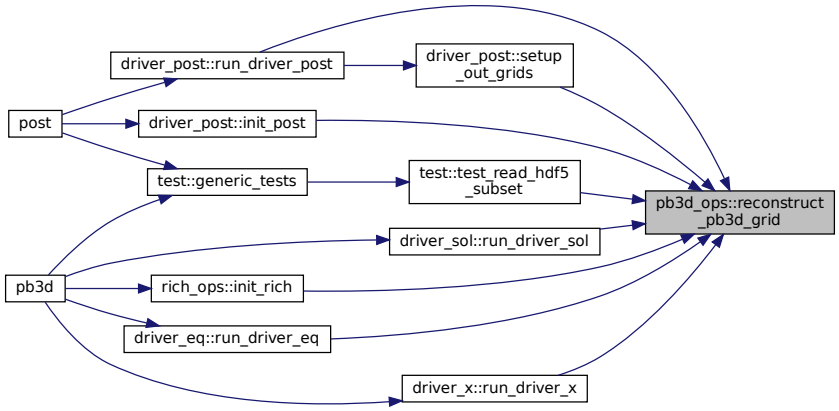
in	*data_name*	name of grid
in	*rich_lvl*	Richardson level to reconstruct
in	*tot_rich*	whether to combine with previous Richardson levels
in	*lim_pos*	position limits of subset of data
in	*grid_limits*	i_limit of grid

Definition at line 442 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.5 reconstruct_pb3d_in()

```
integer function, public pb3d_ops::reconstruct_pb3d_in (
              character(len=*), intent(in) data_name )
```

Reconstructs the input variables from PB3D HDF5 output.

Returns

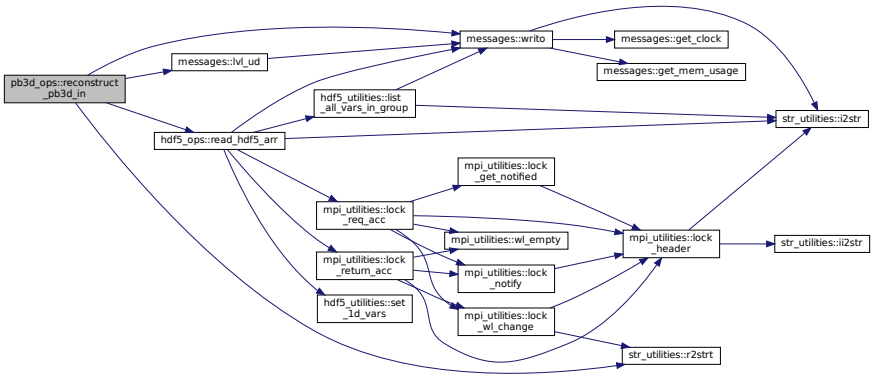
ierr

Parameters

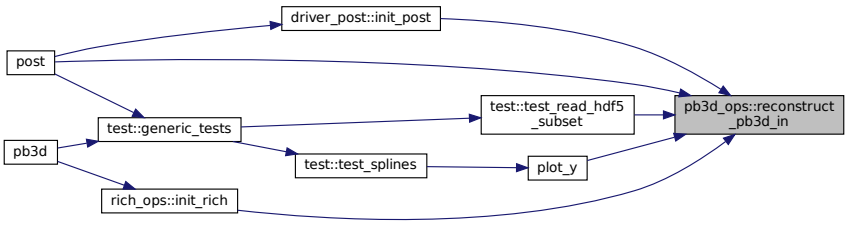
in	*data_name*	name of input variables

Definition at line 41 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.6 reconstruct_pb3d_sol()

```
integer function, public pb3d_ops::reconstruct_pb3d_sol (
              type(modes_type), intent(in) mds,
              type(grid_type), intent(in) grid_sol,
              type(sol_type), intent(inout) sol,
              character(len=*), intent(in) data_name,
              integer, intent(in), optional rich_lvl,
              integer, dimension(2), intent(in), optional lim_sec_sol,
              integer, dimension(1,2), intent(in), optional lim_pos )
```

Reconstructs the solution variables from PB3D HDF5 output.

Also, if `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

Furthermore, using `lim_pos`, you can obtain a subset of the data by directly passing its limits to the underlying HDF5 routine. This refers to the position dimensions only. If provided, the normal limits of a divided grid refer to the subset, as in copy_grid().

`lim_sec_sol`, on the other hand, selects a range of mode numbers.

Returns

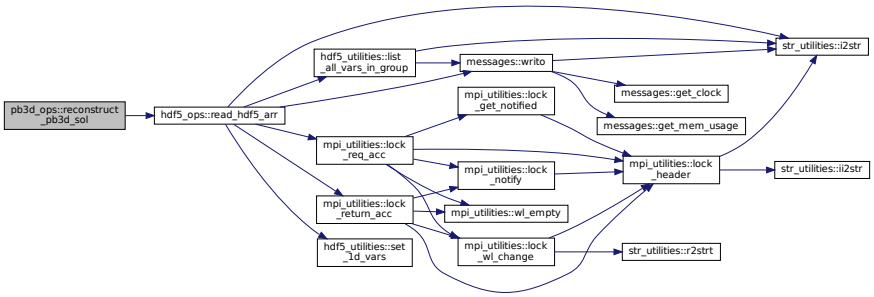
ierr

Parameters

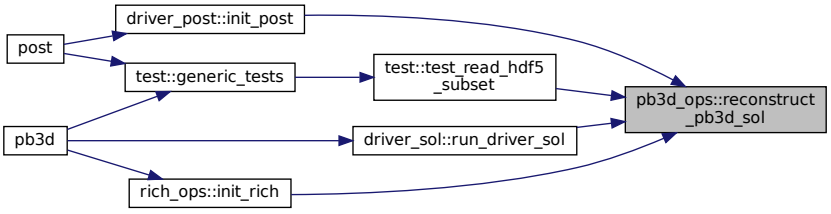
in	*mds*	general modes variables
in	*grid_sol*	solution grid
in,out	*sol*	solution variables
in	*data_name*	name to reconstruct
in	*rich_lvl*	Richardson level to reconstruct
in	*lim_sec_sol*	limits of m_X (pol. flux) or n_X (tor. flux)
in	*lim_pos*	position limits of subset of data

Definition at line 1359 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.7 reconstruct_pb3d_vac()

```
integer function, public pb3d_ops::reconstruct_pb3d_vac (
              type(vac_type), intent(inout) vac,
              character(len=*), intent(in) data_name,
              integer, intent(in), optional rich_lvl )
```

Reconstructs the vacuum variables from PB3D HDF5 output.

Also, if `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

Returns

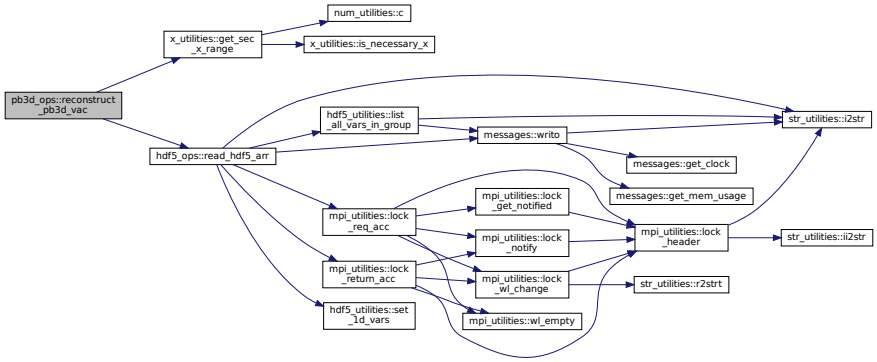
ierr

Parameters

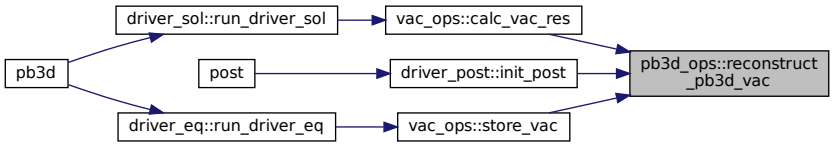
in,out	*vac*	vacuum variables
in	*data_name*	name to reconstruct
in	*rich_lvl*	Richardson level to reconstruct

Definition at line 1228 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.8 reconstruct_pb3d_x_1()

```
integer function, public pb3d_ops::reconstruct_pb3d_x_1 (
                type(modes_type), intent(in) mds,
                type(grid_type), intent(in) grid_X,
                type(x_1_type), intent(inout) X,
                character(len=*), intent(in) data_name,
                integer, intent(in), optional rich_lvl,
                logical, intent(in), optional tot_rich,
                integer, dimension(2), intent(in), optional lim_sec_X,
                integer, dimension(3,2), intent(in), optional lim_pos )
```

Reconstructs the vectorial perturbation variables from PB3D HDF5 output.

Also, if `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

With `tot_rich` the information from previous Richardson levels can be combined.

Furthermore, using `lim_pos`, you can obtain a subset of the data by directly passing its limits to the underlying HDF5 routine. This refers to the position dimensions only. If provided, the normal limits of a divided grid refer to the subset, as in copy_grid().

`lim_sec_X`, on the other hand, selects a range of mode numbers.

Returns

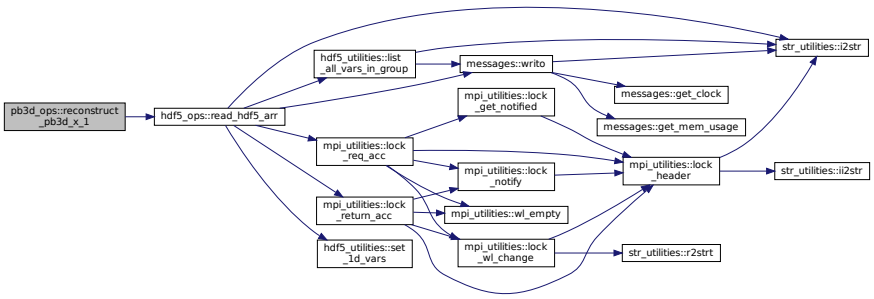
ierr

Parameters

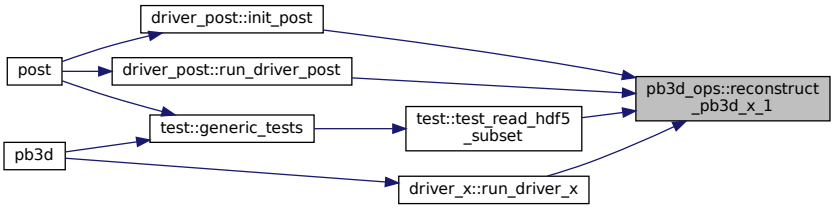
`in`	*mds*	general modes variables
`in`	*grid_x*	perturbation grid
`in,out`	*x*	vectorial perturbation variables
`in`	*data_name*	name to reconstruct
`in`	*rich_lvl*	Richardson level to reconstruct
`in`	*tot_rich*	whether to combine with previous Richardson levels
`in`	*lim_sec_x*	limits of m_X (pol. flux) or n_X (tor. flux)
`in`	*lim_pos*	position limits of subset of data

Definition at line 833 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.29.2.9 reconstruct_pb3d_x_2()

```
integer function, public pb3d_ops::reconstruct_pb3d_x_2 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               type(x_2_type), intent(inout) X,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional tot_rich,
               integer, dimension(2,2), intent(in), optional lim_sec_X,
               integer, dimension(3,2), intent(in), optional lim_pos,
               logical, intent(in), optional is_field_averaged )
```

Reconstructs the tensorial perturbation variables from PB3D HDF5 output.

Also, if `rich_lvl` is provided, `_R_[rich_lvl]` is appended to the data name if it is > 0.

With `tot_rich` the information from previous Richardson levels can be combined.

Furthermore, using `lim_pos`, you can obtain a subset of the data by directly passing its limits to the underlying HDF5 routine. This refers to the position dimensions only. If provided, the normal limits of a divided grid refer to the subset, as in copy_grid().

`lim_sec_X`, on the other hand, selects a range of mode numbers.

Note

The tensorial perturbation type can also be used for field- aligned variables, in which case the first index is assumed to have dimension 1 only. This can be triggered using `is_field_averaged`.

Returns

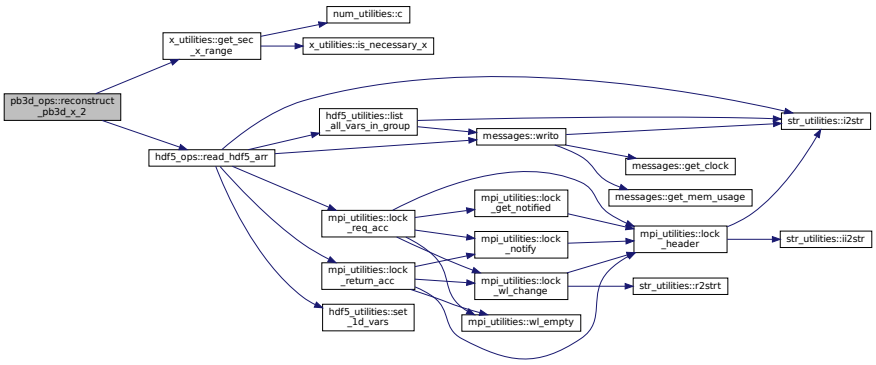
ierr

Parameters

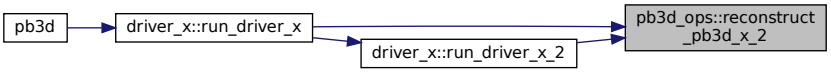
in	*mds*	general modes variables
in	*grid_x*	perturbation grid
in,out	*x*	tensorial perturbation vars
in	*data_name*	name to reconstruct
in	*rich_lvl*	Richardson level to reconstruct
in	*tot_rich*	whether to combine with previous Richardson levels
in	*lim_sec_x*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions
in	*lim_pos*	position limits of subset of data
in	*is_field_averaged*	if field-averaged, only one dimension for first index

Definition at line 992 of file PB3D_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.30 pb3d_utilities Module Reference

Numerical utilities related to PB3D operations.

Interfaces and Types

- interface conv_1d2nd

  Converts 1-D to n-D variables.

Functions/Subroutines

- integer function, dimension(3), public setup_par_id (grid, rich_lvl_max, rich_lvl_loc, tot_rich, par_lim, par_id_mem)

  Setup parallel id.
- integer function, dimension(2), public setup_rich_id (rich_lvl_max, tot_rich)

  Returns richardson id.

B.30.1 Detailed Description

Numerical utilities related to PB3D operations.

B.30.2 Function/Subroutine Documentation

B.30.2.1 setup_par_id()

```
integer function, dimension(3), public pb3d_utilities::setup_par_id (
               type(grid_type), intent(in) grid,
               integer, intent(in) rich_lvl_max,
               integer, intent(in) rich_lvl_loc,
               logical, intent(in), optional tot_rich,
               integer, dimension(2), intent(in), optional par_lim,
               integer, dimension(2), intent(inout), optional par_id_mem )
```

Setup parallel id.

The parallel id consists of:

- `par_id(1)`: start index
- `par_id(2)`: end index
- `par_id(3)`: stride

These are set up by considering the transformation between a point $r \in (1 \dots n)$ in the local variable, with corresponding indices $(a \dots b)$ indicated by `par_lim` in the HDF5 variable in memory, so that $n = b - a + 1$:

$$p + ks = a + r - 1,$$

where k is an integer, s is the stride for Richardson level i (with max. level I):

$$\begin{aligned} s &= 2^{I-1} \text{ for } i = 1 \\ &= 2^{I-i+1} \text{ for } i > 1 \end{aligned}$$

and where p is the starting point in the HDF5 variables:

$$\begin{aligned} p &= 1 \text{ for } i = 1 \\ &= 1 + s/2 \text{ for } i > 1 \end{aligned}$$

Therefore, the minimum possible r lies in $(0..s-1)$:

$$\mod(r-1, s) = r_{\text{min}} - 1,$$

which leads to

$$r_{\text{min}} = 1 + \mod(p-a, s).$$

The maximum possible r, then, lies in $(n-s+1..n)$, which leads to:

$$r_{\text{max}} = n - s + 1 + \mod(p-b-1, s).$$

These limits and the strides are saved in `par_id` = $\vec{r} = \begin{pmatrix} r_{\text{min}} \\ r_{\text{max}} \end{pmatrix}$.

If the optional indices a and b are not given they are assumed to be 1 and n, with $n = 1 + ks$, which simplifies the equations to:

$$\begin{aligned} r_{\text{min}} &= p \\ r_{\text{max}} &= n - p + 1. \end{aligned}$$

Optionally, the indices in the HDF5 arrays can also be returned in `par_id_mem` = $\begin{pmatrix} R_{\text{min}} \\ R_{\text{max}} \end{pmatrix}$. These are equal to $k-1$, where k is the integer refered to above:

$$\vec{R} = 1 + \frac{a - 1 - p + \vec{r}}{s},$$

where the addition between a vector and a scalar should be seen as the element-wise operation.

Parameters

in	*rich_lvl_max*	maximum Richardson level
in	*rich_lvl_loc*	local Richardson level
in	*tot_rich*	whether to combine with previous Richardson levels
in	*par_lim*	limits (a..b) of variable requested
in,out	*par_id_mem*	parallel id in memory

Returns

parallel id

Definition at line 181 of file PB3D_utilities.f90.

B.30.2.2 setup_rich_id()

```
integer function, dimension(2), public pb3d_utilities::setup_rich_id (
               integer, intent(in) rich_lvl_max,
               logical, intent(in), optional tot_rich )
```

Returns richardson id.

`rich_id` has the following structure:

- `rich_id(1)`: start Richardson level
- `rich_id(2)`: end Richardson level

Parameters

in	*rich_lvl_max*	maximum Richardson level
in	*tot_rich*	whether to combine with previous Richardson levels

Returns

Richardson id

Definition at line 234 of file PB3D_utilities.f90.

B.31 rich_ops Module Reference

Operations concerning Richardson extrapolation.

Functions/Subroutines

- integer function, public init_rich ()

  Initialize Richardson extrapolation system.
- subroutine, public term_rich ()

  Terminate the Richardson extrapolation system.
- logical function, public do_rich ()

  Tests whether this Richardson level should be done.
- integer function, public start_rich_lvl ()

  Start a Richardson level.
- subroutine, public stop_rich_lvl ()

  Stop a Richardson level.
- subroutine, public calc_rich_ex (sol_val)

  Calculates the coefficients of the Eigenvalues in the Richardson extrapolation.
- subroutine check_conv ()

  Decides whether convergence has been reached for the Richardson extrapolation.
- integer function, public find_max_rich_lvl (lvl_rich_req, max_lvl_rich_file)

  Probe to find out which Richardson levels are available.

B.31.1 Detailed Description

Operations concerning Richardson extrapolation.

B.31.2 Function/Subroutine Documentation

B.31.2.1 calc_rich_ex()

```
subroutine, public rich_ops::calc_rich_ex (
               complex(dp), dimension(:), intent(in) sol_val )
```

Calculates the coefficients of the Eigenvalues in the Richardson extrapolation.

This is done using the recursive formula:

$$s(l,i,:) = s(l,i-1,:) + \frac{1}{2^{2\,p\,i}-1}(s(l,i-1,:) - s(l-1,i-1,:)),$$

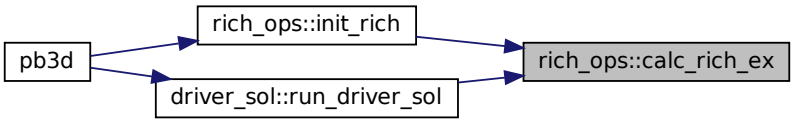
where s = `sol_val_rich`, i = `ir`, l = `lvl` and p = `norm_disc_prec_sol`, the order of the numerical discretization. as described in [4], algorithm 3.2, p. 306.

Parameters

in	*sol_val*	EV for this Richardson level

Definition at line 459 of file rich_ops.f90.

Here is the caller graph for this function:



B.31.2.2 check_conv()

`subroutine rich_ops::check_conv`

Decides whether convergence has been reached for the Richardson extrapolation.

This is done by checking whether the difference between the approximation of the Eigenvalues in this Richardson level and the previous Richardson level, with the same order of the error, falls below a threshold.

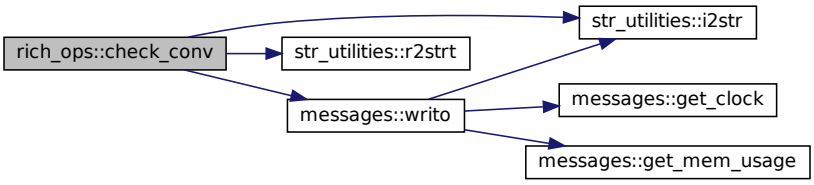
Also increases the next tol_SLEPC if it is too low.

See also

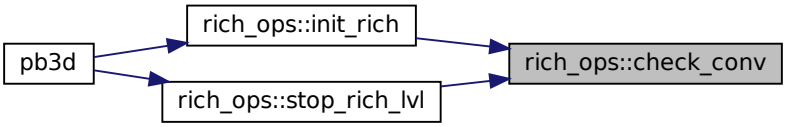
calc_rich_ex()

Definition at line 489 of file rich_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.31.2.3 do_rich()

```
logical function, public rich_ops::do_rich
```

Tests whether this Richardson level should be done.

Definition at line 313 of file rich_ops.f90.

Here is the caller graph for this function:

B.31.2.4 find_max_rich_lvl()

```
integer function, public rich_ops::find_max_rich_lvl (
               integer, intent(inout) lvl_rich_req,
               integer, intent(inout) max_lvl_rich_file )
```

Probe to find out which Richardson levels are available.

Returns

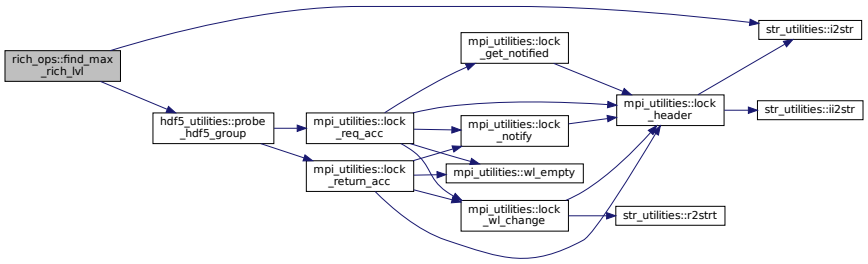
ierr

Parameters

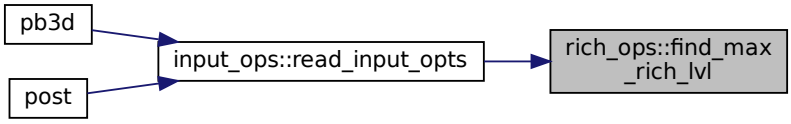
in,out	*lvl_rich_req*	requested Richardson level
in,out	*max_lvl_rich_file*	max. Richardson level found in file

Definition at line 539 of file rich_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.31.2.5 init_rich()

```
integer function, public rich_ops::init_rich
```

Initialize Richardson extrapolation system.

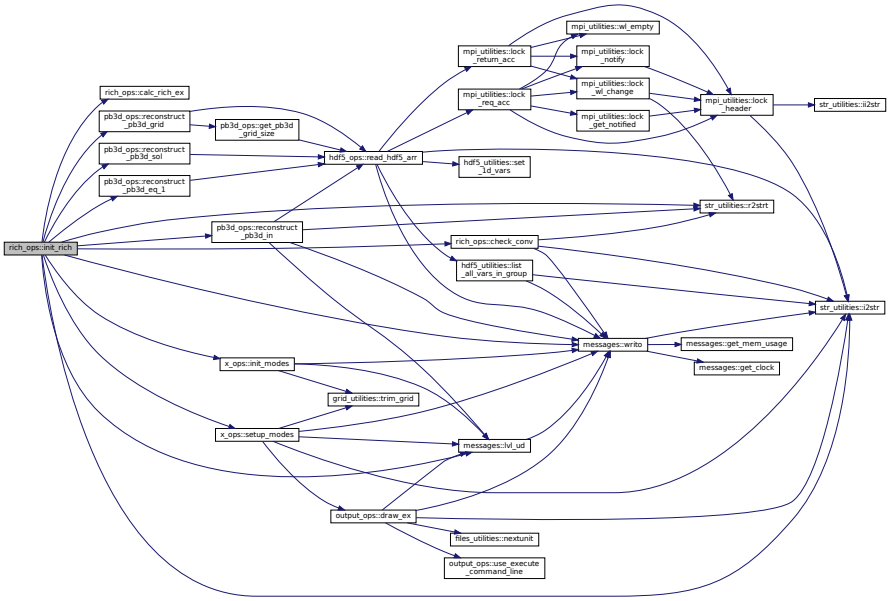
Needs to be called only once.

Returns

ierr

Definition at line 28 of file rich_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.31.2.6 start_rich_lvl()

```
integer function, public rich_ops::start_rich_lvl
```

Start a Richardson level.

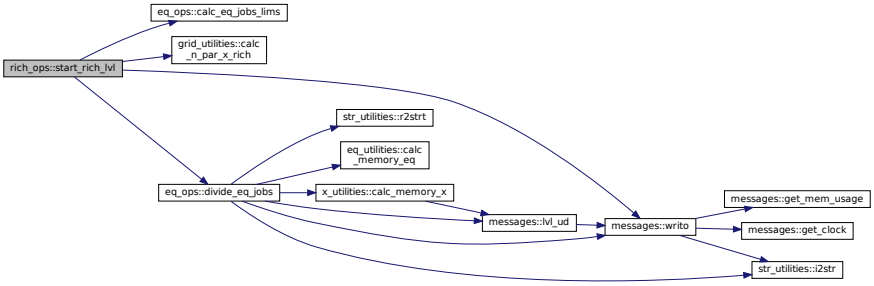
Calculates `n_par_X` for this level. Then uses this to divide the jobs. equilibrium Finally, the limits are set up for these jobs.

Returns

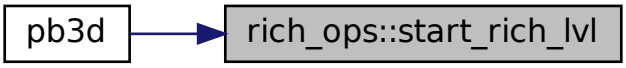
ierr

Definition at line 327 of file rich_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.31.2.7 stop_rich_lvl()

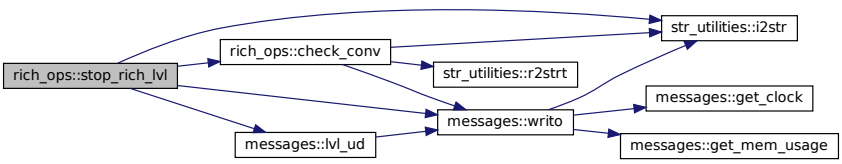
`subroutine, public rich_ops::stop_rich_lvl`

Stop a Richardson level.

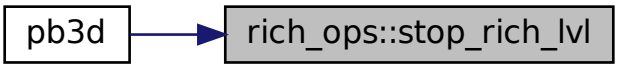
It decides whether convergence has been reached, and possibly sets a guess for the next level.

Definition at line 405 of file rich_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.31.2.8 term_rich()

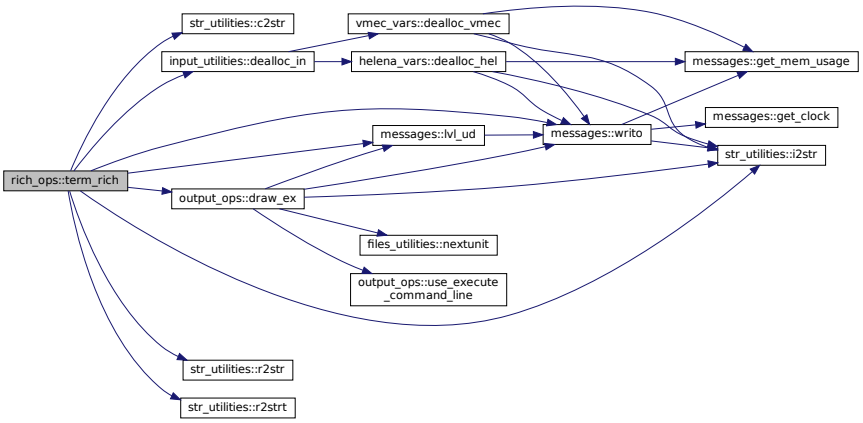
`subroutine, public rich_ops::term_rich`

Terminate the Richardson extrapolation system.

Needs to be called only once.

Definition at line 220 of file rich_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.32 rich_vars Module Reference

Variables concerning Richardson extrapolation.

Functions/Subroutines

- elemental character(len=max_str_ln) function, public rich_info ()
  Returns string with possible extension with Richardson level or nothing if only one level and one parallel job.

Variables

- integer, public rich_lvl
  current level of Richardson extrapolation
- integer, public n_par_x
  nr. of parallel points in field-aligned grid
- integer, parameter, public req_min_n_par_x = 20
  required minimum n_par_X
- integer, public min_n_par_x
  min. of n_par_X (e.g. first value in Richardson loop)
- logical, public use_guess
  whether a guess is formed from previous level of Richardson
- logical, public no_guess
  disable guessing Eigenfunction from previous level of Richardson
- logical, public rich_conv
  if Richarson extrapolation has converged
- complex(dp), dimension(:,:,:), allocatable, public sol_val_rich
  Richardson array of eigenvalues.
- real(dp), dimension(:,:), allocatable, public x_axis_rich
  x axis for plot of Eigenvalues with Richardson level
- real(dp), dimension(:), allocatable, public max_rel_err
  maximum relative error for all Richardson levels
- integer, dimension(:,:), allocatable, public loc_max_rel_err
  location of maximum of relative error

B.32.1 Detailed Description

Variables concerning Richardson extrapolation.

B.32.2 Function/Subroutine Documentation

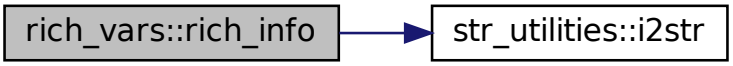
B.32.2.1 rich_info()

```
elemental character(len=max_str_ln) function, public rich_vars::rich_info
```

Returns string with possible extension with Richardson level or nothing if only one level and one parallel job.

Definition at line 35 of file rich_vars.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.32.3 Variable Documentation

B.32.3.1 loc_max_rel_err

```
integer, dimension(:,:), allocatable, public rich_vars::loc_max_rel_err
```

location of maximum of relative error

Definition at line 29 of file rich_vars.f90.

B.32.3.2 max_rel_err

```
real(dp), dimension(:), allocatable, public rich_vars::max_rel_err
```

maximum relative error for all Richardson levels

Definition at line 28 of file rich_vars.f90.

B.32.3.3 min_n_par_x

```
integer, public rich_vars::min_n_par_x
```

min. of n_par_X (e.g. first value in Richardson loop)

Definition at line 22 of file rich_vars.f90.

B.32.3.4 n_par_x

```
integer, public rich_vars::n_par_x
```

nr. of parallel points in field-aligned grid

Definition at line 20 of file rich_vars.f90.

B.32.3.5 no_guess

```
logical, public rich_vars::no_guess
```

disable guessing Eigenfunction from previous level of Richardson

Definition at line 24 of file rich_vars.f90.

B.32.3.6 req_min_n_par_x

```
integer, parameter, public rich_vars::req_min_n_par_x = 20
```

required minimum n_par_X

Definition at line 21 of file rich_vars.f90.

B.32.3.7 rich_conv

```
logical, public rich_vars::rich_conv
```

if Richarson extrapolation has converged

Definition at line 25 of file rich_vars.f90.

B.32.3.8 rich_lvl

```
integer, public rich_vars::rich_lvl
```

current level of Richardson extrapolation

Definition at line 19 of file rich_vars.f90.

B.32.3.9 sol_val_rich

```
complex(dp), dimension(:,:,:), allocatable, public rich_vars::sol_val_rich
```

Richardson array of eigenvalues.

Definition at line 26 of file rich_vars.f90.

B.32.3.10 use_guess

```
logical, public rich_vars::use_guess
```

whether a guess is formed from previous level of Richardson

Definition at line 23 of file rich_vars.f90.

B.32.3.11 x_axis_rich

```
real(dp), dimension(:,:), allocatable, public rich_vars::x_axis_rich
```

x axis for plot of Eigenvalues with Richardson level

Definition at line 27 of file rich_vars.f90.

B.33 slepc_ops Module Reference

Operations that use SLEPC (and PETSC) routines.

Functions/Subroutines

- integer function, public solve_ev_system_slepc (mds, grid_sol, X, vac, sol)
  Solve the eigenvalue system using SLEPC.
- integer function, public start_slepc (n_r_sol)
  This subroutine starts PETSC and SLEPC with the correct number of processors.
- integer function, public setup_mats (mds, grid_sol, X, A, B)
  Sets up the matrices $\overline{A}$ and $\overline{B}$ in the EV problem $\overline{A}\vec{X} = \lambda\overline{B}\vec{X}$..
- integer function, public set_bc (mds, grid_sol, X, vac, A, B)
  Sets the boundary conditions deep in the plasma (1) and at the edge (2).
- integer function, public setup_solver (X, A, B, solver)
  Sets up EV solver.
- integer function, public setup_guess (sol, A, solver)
  Sets up guess in solver.
- integer function, public get_solution (solver)
  Get the solution vectors.
- integer function, public summarize_solution (solver, max_n_EV)
  Summarizes solution.
- integer function, public store_results (mds, grid_sol, sol, solver, max_n_EV, A, B)
  Stores the results.
- integer function, public stop_slepc (A, B, solver)
  Stop PETSC and SLEPC.

Variables

- logical, public debug_setup_mats = .false.
  plot debug information for setup_mats
- logical, public debug_set_bc = .false.
  plot debug information for set_BC
- logical, public test_diff = .false.
  test introduction of numerical diff

B.33.1 Detailed Description

Operations that use SLEPC (and PETSC) routines.

Note

The routines here require a **TRIMMED** solution grid!

See also

References: [7]

B.33.2 Function/Subroutine Documentation

B.33.2.1 get_solution()

```
integer function, public slepc_ops::get_solution (
              intent(inout) solver )
```

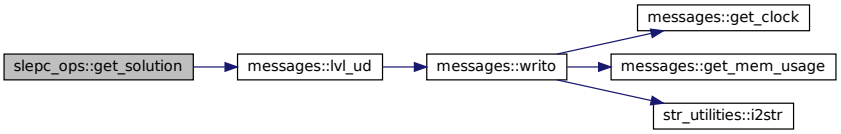
Get the solution vectors.

Returns

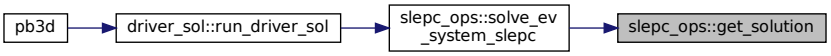
ierr

Definition at line 1534 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.2 set_bc()

```
integer function, public slepc_ops::set_bc (
              type(modes_type), intent(in) mds,
              type(grid_type), intent(in) grid_sol,
              type(x_2_type), intent(in) X,
              type(vac_type), intent(in) vac,
              intent(inout) A,
              intent(inout) B )
```

Sets the boundary conditions deep in the plasma (1) and at the edge (2).

Through `norm_disc_style_sol` it can be chosen whether the finite differences are:

1. central finite differences
2. left finite differences

Possibilities for `BC_style`:

1. Set to zero:
   - An artificial Eigenvalue `EV_BC` is introduced by setting the diagonal components of A to EV_BC and of B to 1, and the off-diagonal elements to zero.
2. Minimization of surface energy through asymmetric fin. differences:
   - For symmetric finite differences, the last `ndps` grid points are treated asymmetrically in order not go over the edge and the vacuum term is added to the edge element.
   - For left differences, this is already standard, so the method becomes identical to 2.
3. Minimization of surface energy through extension of grid:
   - For symmetric finite differences, `ndps` extra grid points are introduced after the edge and the vacuum term is added to the edge element.
   - For left finite differences, the vacuum term is just added to the edge element, so this method becomes idential to 3.
4. Explicit introduction of the surface energy minimization:
   - The equation due to the minimization of the vacuum term is introduced explicitely as an asymmetric finite difference equation in the last row. -This is done using left finite differences.

Makes use of n_r.

Returns

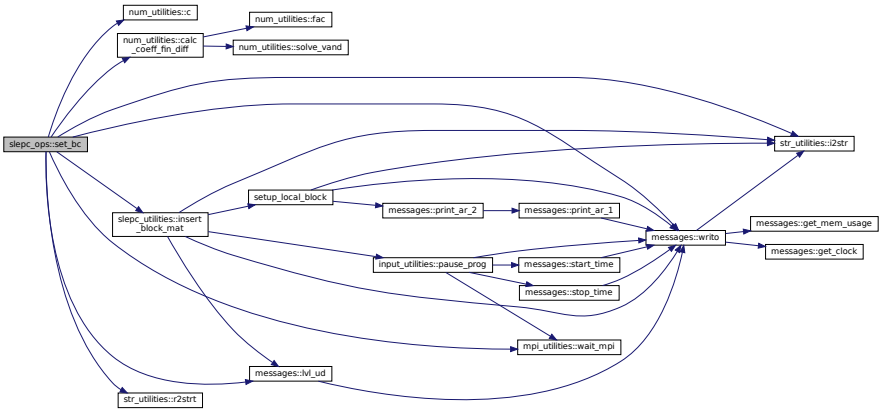
ierr

Parameters

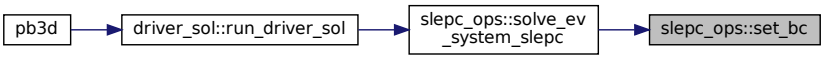
`in`	*mds*	general modes variables
`in`	*grid_sol*	solution grid
`in`	*x*	field-averaged perturbation variables (so only first index)
`in`	*vac*	vacuum variables

Definition at line 933 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.3 setup_guess()

```
integer function, public slepc_ops::setup_guess (
               type(sol_type), intent(in) sol,
               intent(in) A,
               intent(inout) solver )
```

Sets up guess in solver.

Returns

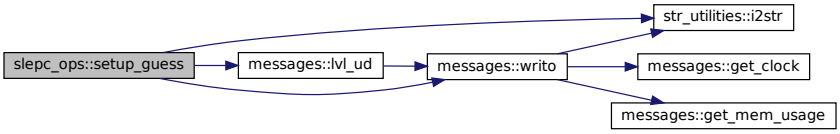
ierr

Parameters

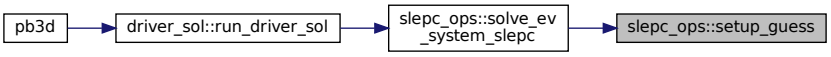
in	*sol*	solution variables

Definition at line 1453 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.4 setup_mats()

```
integer function, public slepc_ops::setup_mats (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_sol,
               type(x_2_type), intent(in), target X,
               intent(inout) A,
               intent(inout) B )
```

Sets up the matrices $\overline{\mathrm{A}}$ and $\overline{\mathrm{B}}$ in the EV problem $\overline{\mathrm{A}}\vec{X} = \lambda\overline{\mathrm{B}}\vec{X}$..

The matrices are set up in the solution grid, so some processes might not have any part in the storage thereof (if `sol_n_procs` $<$ `n_procs`), but each process sets up the part of the grid that corresponds to their own normal range in the solution grid.

Returns

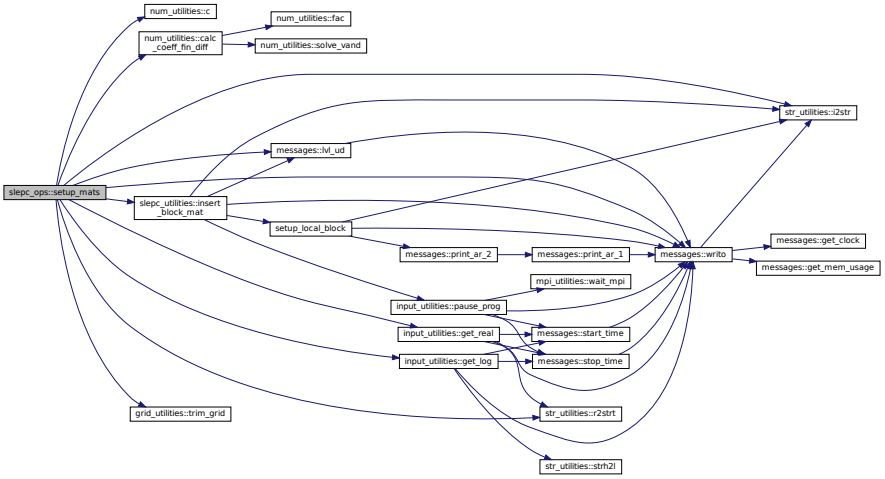
ierr

Parameters

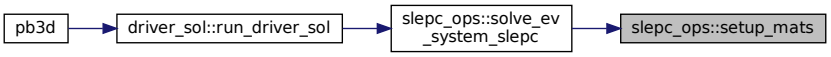
in	*mds*	general modes variables
in	*grid_sol*	solution grid
in	*x*	field-averaged perturbation variables (so only first index)

Definition at line 409 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.5 setup_solver()

```
integer function, public slepc_ops::setup_solver (
               type(x_2_type), intent(in) X,
               intent(in) A,
               intent(in) B,
               intent(inout) solver )
```

Sets up EV solver.

Returns

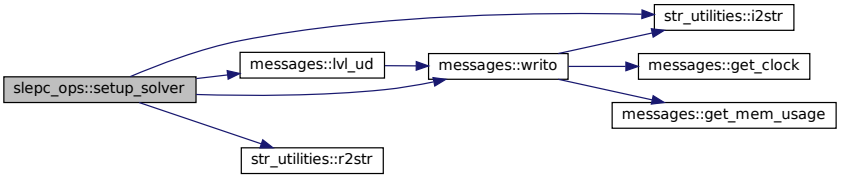
ierr

Parameters

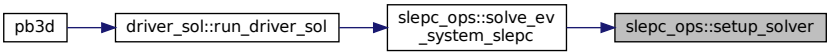
in	*x*	field-averaged perturbation variables (so only first index)

Definition at line 1326 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.6 solve_ev_system_slepc()

```
integer function, public slepc_ops::solve_ev_system_slepc (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_sol,
               type(x_2_type), intent(in) X,
               type(vac_type), intent(in) vac,
               type(sol_type), intent(inout) sol )
```

Solve the eigenvalue system using SLEPC.

This subroutine sets up the matrices A ad B of the generalized EV system described in [17] and solves them using the SLEPC suite. The solutions closest to a target indicated by `EV_guess` are obtained.

See also

read_input_opts()

Returns

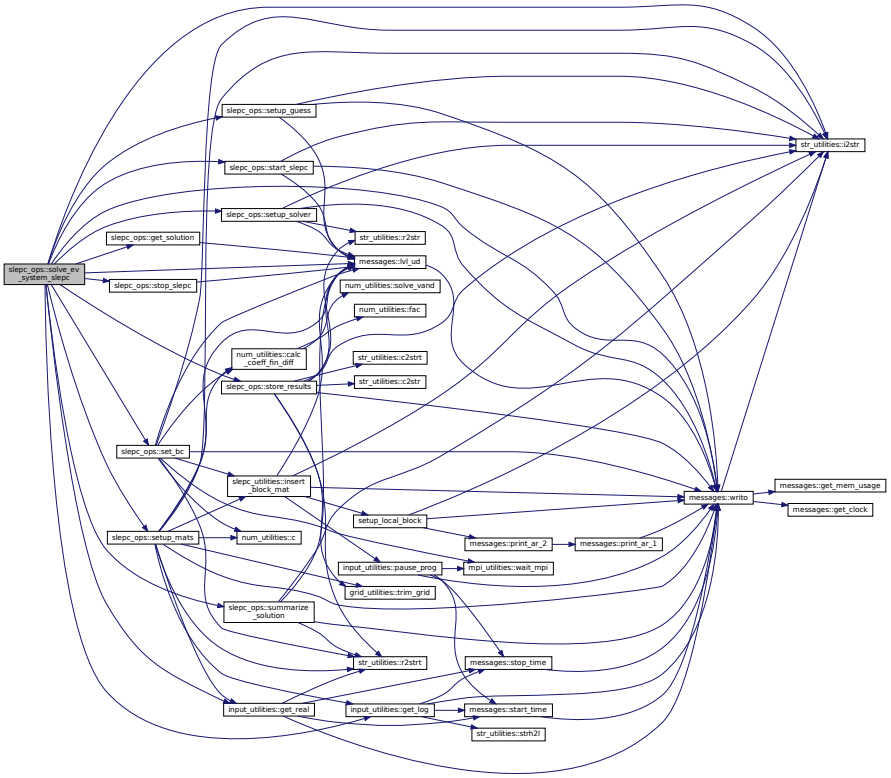
ierr

Parameters

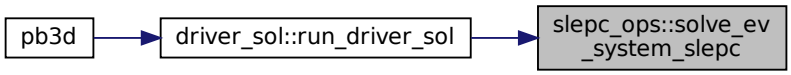
`in`	*mds*	general modes variables
`in`	*grid_sol*	solution grid
`in`	*x*	field-averaged perturbation variables (so only first index)
`in`	*vac*	vacuum variables
`in,out`	*sol*	solution variables

Definition at line 62 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.7 start_slepc()

```
integer function, public slepc_ops::start_slepc (
              integer, intent(in) n_r_sol )
```

This subroutine starts PETSC and SLEPC with the correct number of processors.

Returns

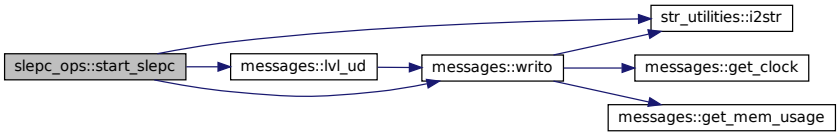
ierr

Parameters

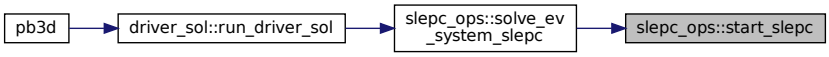
in	*n_r_sol*	nr. of normal points in solution grid

Definition at line 333 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.8 stop_slepc()

```
integer function, public slepc_ops::stop_slepc (
               intent(in) A,
               intent(in) B,
               intent(in) solver )
```

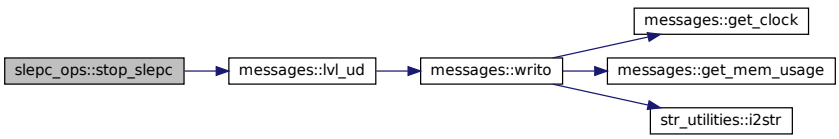
Stop PETSC and SLEPC.

Returns

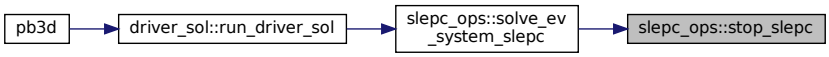
ierr

Definition at line 2054 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.9 store_results()

```
integer function, public slepc_ops::store_results (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_sol,
               type(sol_type), intent(inout) sol,
               intent(inout) solver,
               intent(inout) max_n_EV,
               intent(inout) A,
               intent(inout) B )
```

Stores the results.

Returns

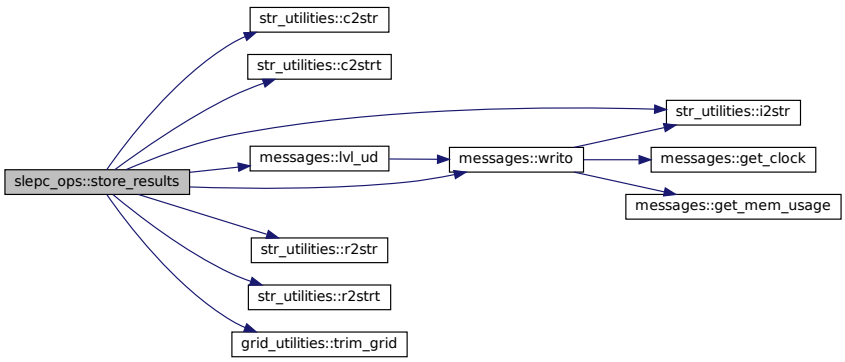
ierr

Parameters

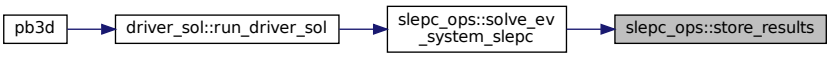
in	*mds*	general modes variables
in	*grid_sol*	solution grid
in,out	*sol*	solution variables

Definition at line 1634 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.2.10 summarize_solution()

```
integer function, public slepc_ops::summarize_solution (
               intent(inout) solver,
               intent(inout) max_n_EV )
```

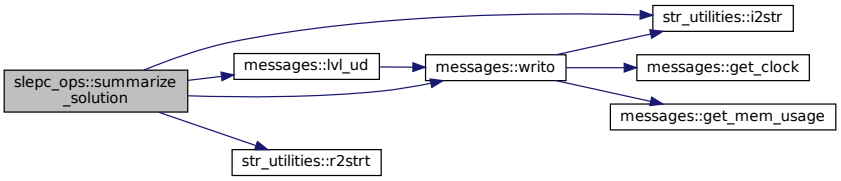
Summarizes solution.

Returns

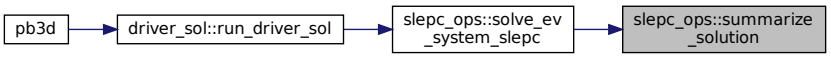
ierr

Definition at line 1564 of file SLEPC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.33.3 Variable Documentation

B.33.3.1 debug_set_bc

```
logical, public slepc_ops::debug_set_bc = .false.
```

plot debug information for set_BC

Note

Debug version only

Definition at line 46 of file SLEPC_ops.f90.

B.33.3.2 debug_setup_mats

```
logical, public slepc_ops::debug_setup_mats = .false.
```

plot debug information for setup_mats

Note

Debug version only

Definition at line 45 of file SLEPC_ops.f90.

B.33.3.3 test_diff

```
logical, public slepc_ops::test_diff = .false.
```

test introduction of numerical diff

Note

Debug version only

Definition at line 47 of file SLEPC_ops.f90.

B.34 slepc_utilities Module Reference

Numerical utilities related to SLEPC (and PETSC) operations.

Functions/Subroutines

- integer function, public insert_block_mat (mds, block, mat, r_id, ind, n_r, transp, overwrite, ind_insert)
  Insert a block pertaining to 1 normal point into a matrix.

Variables

- logical, public debug_insert_block_mat = .false.
  plot debug information for insert_block_mat()

B.34.1 Detailed Description

Numerical utilities related to SLEPC (and PETSC) operations.

See also

References: [7]

B.34.2 Function/Subroutine Documentation

B.34.2.1 insert_block_mat()

```
integer function, public slepc_utilities::insert_block_mat (
               type(modes_type), intent(in), target mds,
               dimension(:,:), intent(in) block,
               intent(inout) mat,
               intent(in) r_id,
               dimension(2), intent(in) ind,
               intent(in) n_r,
               optional transp,
               optional overwrite,
               optional ind_insert )
```

Insert a block pertaining to 1 normal point into a matrix.

Optionally, also set the Hermitian transpose and / or overwrite instead of add value.

This routine takes into account that if the mode numbers change as a function of the normal coordinate, as is the case for X_style 2 (fast), part of the local blocks, which were set up without taking this into account, can have information that correspond to mode numbers that are not present any more for off-diagonal entries, and lack an equal amount of information that corresponds to the mode numbers that have taken their places.

Omission of this effect is justifiable if there are many more mode couplings that are not omitted (i.e. when n_mod_X is large), so that it is a small effect, and/or when the coupling between these modes is already negligible.

Unfortunately, it can be argued that this is not generally the case for the fast version (X_style = 2) since $\widetilde{PV}^1$ and $\widetilde{PV}^2$ are both $\sim (nq-m)$, which indicates that the smallest elements in the corresponding off-diagonal blocks lie in the central columns for $\widetilde{PV}^1$ and in the central columns and rows for $\widetilde{PV}^2$, while the problematic elements lie farthest away from the central columns for $\widetilde{PV}^1$ and the central columns and rows for $\widetilde{PV}^2$, so these are generally not small.

The coefficients due to normal discretization get increasingly small for higher orders, though, which compensates this for higher orders.

Finally, an alternative would be to set the problematic elements not to zero but extrapolated from the closest elements, but then consistency would be sacrified, for example in the energy reconstruction of the POST-processing: In the energy reconstruction the exact same terms are neglected as well.

To conclude, for fast X_style 2, the number of modes (n_mod_X) has to be chosen high enough compared with the variation of the mode numbers; i.e. the variation of the safety factor or rotational transform.

Note

The grid in which the modes variables mds are tabulated up must be the same as the one in which mat is set up. If this is not the case, the indices in mds have to be adapted.

Returns

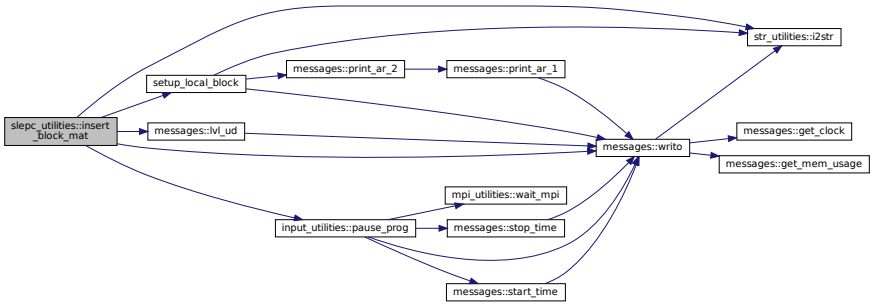
ierr

Parameters

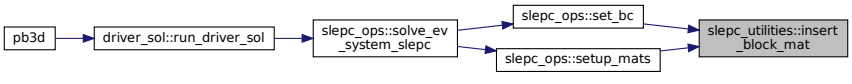
in	*mds*	general modes variables

Definition at line 83 of file SLEPC_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.34.3 Variable Documentation

B.34.3.1 debug_insert_block_mat

```
logical, public slepc_utilities::debug_insert_block_mat = .false.
```

plot debug information for insert_block_mat()

Note

Debug version only

Definition at line 31 of file SLEPC_utilities.f90.

B.35 sol_ops Module Reference

Operations on the solution vectors such as decompososing the energy, etc...

Functions/Subroutines

- subroutine, public plot_sol_vals (sol, last_unstable_id)
  Plots Eigenvalues.
- integer function, public plot_sol_vec (mds_X, mds_sol, grid_eq, grid_X, grid_sol, eq_1, eq_2, X, sol, XYZ, X_id, plot_var)
  Plots Eigenvectors.
- integer function, public plot_harmonics (mds, grid_sol, sol, X_id, res_surf)
  Plots the harmonics and their maximum in 2-D.
- integer function, public decompose_energy (mds_X, mds_sol, grid_eq, grid_X, grid_sol, eq_1, eq_2, X, sol, vac, X_id, B_aligned, XYZ, E_pot_int, E_kin_int)
  Decomposes the plasma potential and kinetic energy in its individual terms for an individual Eigenvalue.
- integer function calc_e (mds_X, mds_sol, grid_eq, grid_X, grid_sol, eq_1, eq_2, X, sol, vac, B_aligned, X_id, E_pot, E_kin, E_pot_int, E_kin_int)
  Calculate the energy terms in the energy decomposition.
- integer function, public print_output_sol (grid, sol, data_name, rich_lvl, remove_previous_arrs)
  Print solution quantities to an output file:

Variables

- logical, public debug_plot_sol_vec = .false.
  plot debug information for plot_sol_vec()
- logical, public debug_calc_e = .false.
  plot debug information for calc_E()
- logical, public debug_x_norm = .false.
  plot debug information X_norm
- logical, public debug_du = .false.
  plot debug information for calculation of DU

B.35.1 Detailed Description

Operations on the solution vectors such as decompososing the energy, etc...

B.35.2 Function/Subroutine Documentation

B.35.2.1 calc_e()

```
integer function sol_ops::calc_e (
               type(modes_type), intent(in) mds_X,
               type(modes_type), intent(in) mds_sol,
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(grid_type), intent(in) grid_sol,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               type(x_1_type), intent(in) X,
               type(sol_type), intent(in) sol,
               type(vac_type), intent(in) vac,
               logical, intent(in) B_aligned,
               integer, intent(in) X_id,
               complex(dp), dimension(:,:,:,:), intent(inout), allocatable E_pot,
               complex(dp), dimension(:,:,:,:), intent(inout), allocatable E_kin,
               complex(dp), dimension(7), intent(inout) E_pot_int,
               complex(dp), dimension(2), intent(inout) E_kin_int )
```

Calculate the energy terms in the energy decomposition.

Note

see explanation of routine in decompose_energy().

Returns

ierr

Parameters

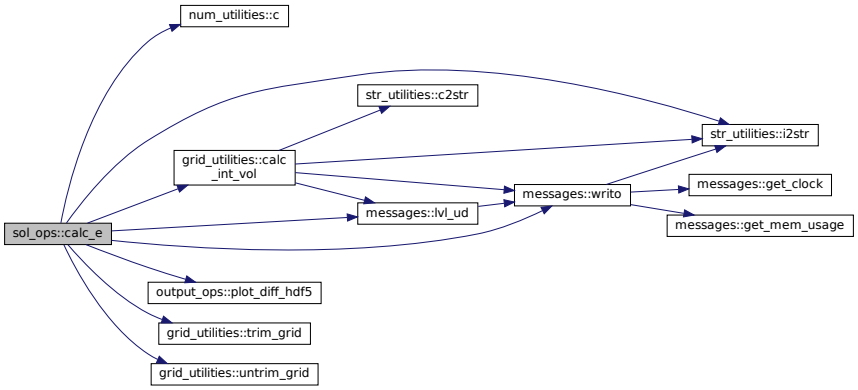
in	*mds_x*	general modes variables in perturbation grid
in	*mds_sol*	general modes variables in solution grid
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*grid_sol*	and solution grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x*	perturbation variables
in	*sol*	solution variables

Parameters

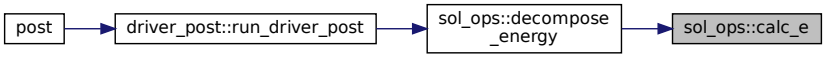
in	*vac*	vacuum variables
in	*b_aligned*	whether grid is field-aligned
in	*x_id*	nr. of Eigenvalue
in,out	*e_pot*	potential energy
in,out	*e_kin*	kinetic energy
in,out	*e_pot_int*	integrated potential energy
in,out	*e_kin_int*	integrated kinetic energy

Definition at line 1205 of file sol_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.35.2.2 decompose_energy()

```
integer function, public sol_ops::decompose_energy (
               type(modes_type), intent(in) mds_X,
               type(modes_type), intent(in) mds_sol,
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(grid_type), intent(in) grid_sol,
               type(eq_1_type), intent(in) eq_1,
```

```
type(eq_2_type), intent(in) eq_2,
type(x_1_type), intent(in) X,
type(sol_type), intent(in) sol,
type(vac_type), intent(in) vac,
integer, intent(in) X_id,
logical, intent(in) B_aligned,
real(dp), dimension(:,:,:,:), intent(in), optional XYZ,
complex(dp), dimension(7), intent(inout), optional E_pot_int,
complex(dp), dimension(2), intent(inout), optional E_kin_int )
```

Decomposes the plasma potential and kinetic energy in its individual terms for an individual Eigenvalue.

Use is made of variables representing the potential and kinetic energy [15].

- E_{pot}:
  - normal line bending term: $\frac{1}{\mu_0}\frac{Q_n^2}{h^{22}}$,
  - geodesic line bending term: $\frac{1}{\mu_0}\mathcal{J}^2\frac{h^{22}}{g_{33}}Q_g^2$,
  - normal ballooning term: $-2p'X^2\kappa_n$,
  - geodesic ballooning term: $-2p'XU^*\kappa_g$,
  - normal kink term: $-\sigma X^* Q_g$,
  - geodesic kink term: $\sigma U^* Q_n$,
- E_{kin}:
  - normal kinetic term: $\rho\frac{X^2}{h^{22}}$,
  - geodesic kinetic term: $\rho\mathcal{J}^2\frac{h^{22}}{g_{33}}U^2$.

The energy terms are calculated normally on the sol grid, interpolating the quantities defined on the eq grid, and angularly in the eq grid.

Optionally, the results can be plotted by providing X, Y and Z. By default, they are instead written to an output file.

Also, the fraction between potential and kinetic energy can be returned, compared with the eigenvalue.

Returns

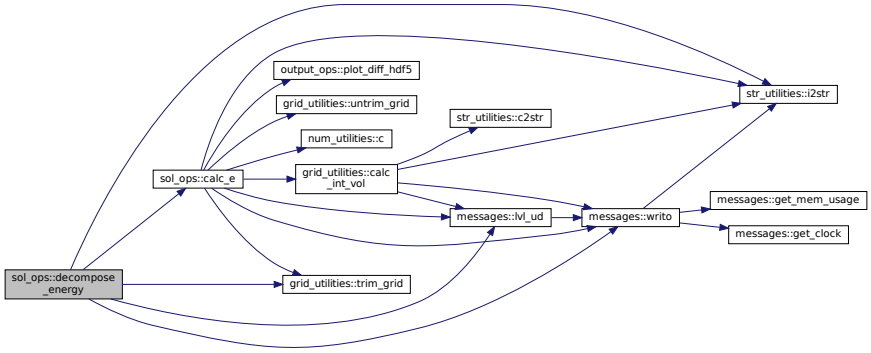
ierr

Parameters

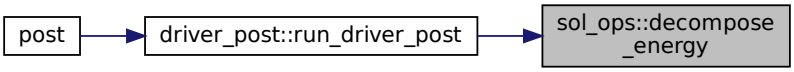
in	*mds_x*	general modes variables in pertubation grid
in	*mds_sol*	general modes variables in solution grid
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*grid_sol*	solution grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x*	perturbation variables
in	*sol*	solution variables
in	*vac*	vacuum variables
in	*x_id*	nr. of Eigenvalue
in	*b_aligned*	whether grid is field-aligned
in	*xyz*	X, Y and Z for plotting

Definition at line 991 of file sol_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.35.2.3 plot_harmonics()

```
integer function, public sol_ops::plot_harmonics (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_sol,
               type(sol_type), intent(in) sol,
               integer, intent(in) X_id,
               real(dp), dimension(:,:), intent(in) res_surf )
```

Plots the harmonics and their maximum in 2-D.

Returns

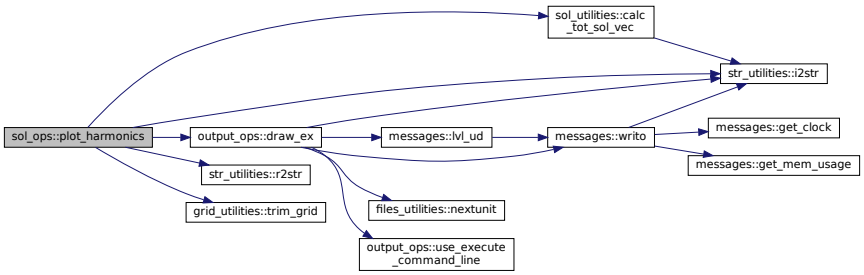
ierr

Parameters

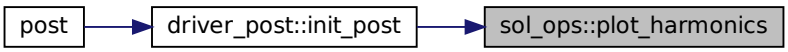
in	*mds*	general modes variables in solution grid
in	*grid_sol*	solution grid
in	*sol*	solution variables
in	*x_id*	nr. of Eigenvalue (for output name)
in	*res_surf*	resonant surfaces

Definition at line 646 of file sol_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.35.2.4 plot_sol_vals()

```
subroutine, public sol_ops::plot_sol_vals (
               type(sol_type), intent(in) sol,
               integer, intent(in) last_unstable_id )
```

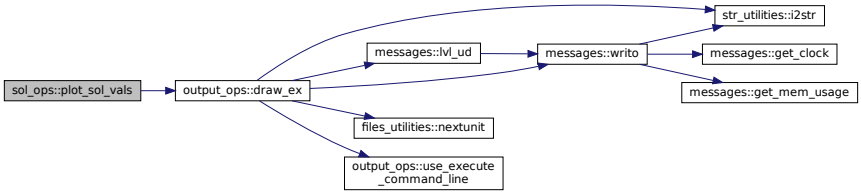
Plots Eigenvalues.

Parameters

in	*sol*	solution variables
in	*last_unstable↩ _id*	index of last unstable EV

Definition at line 37 of file sol_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.35.2.5 plot_sol_vec()

```
integer function, public sol_ops::plot_sol_vec (
               type(modes_type), intent(in) mds_X,
               type(modes_type), intent(in) mds_sol,
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(grid_type), intent(in) grid_sol,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               type(x_1_type), intent(in) X,
               type(sol_type), intent(in) sol,
               real(dp), dimension(:,:,:,:), intent(in) XYZ,
               integer, intent(in) X_id,
               logical, dimension(2), intent(in) plot_var )
```

Plots Eigenvectors.

This is done using the angular part of the the provided equilibrium grid and the normal part of the provided solution grid.

The perturbation grid is assumed to have the same angular coordinates as the equilibrium grid, and the normal coordinates correspond to either the equilibrium grid (`X_grid_style` 1), the solution grid (`X_grid_style` 2) or the enriched equilibrium grid (`X_grid_style` 3).

The output grid, furthermore, has the angular part corresponding to the equilibrium grid, and the normal part to the solution grid.

Note

The normalization factors are taken into account and the output is transformed back to unnormalized values:

$$\vec{\xi} \sim \frac{X_0}{R_0 B_0} ,$$
$$\vec{Q} \sim \frac{X_0}{R_0^2} ,$$

which translates to

$$X \sim X_0 ,$$
$$U \sim \frac{X_0}{R_0^2 B_0} ,$$
$$Qn \sim \frac{X_0 B_0}{R_0} ,$$
$$Qg \sim \frac{X_0}{R_0^3} ,$$

where X_0 is not determined but is common to all factors.

Returns

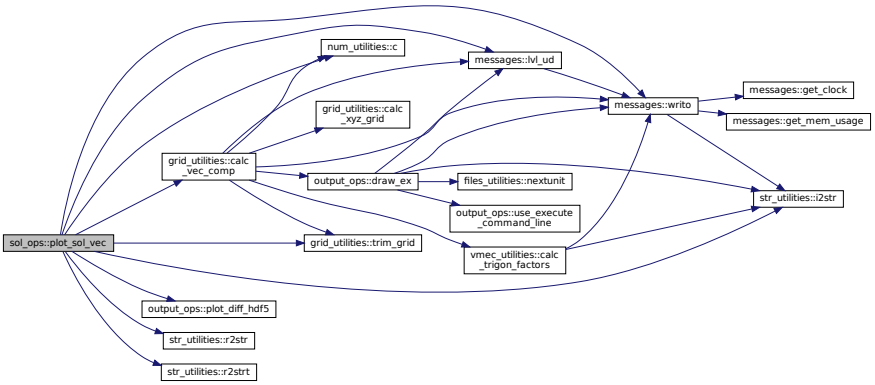
ierr

Parameters

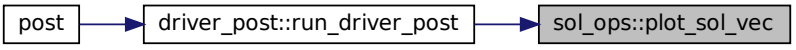
in	*mds_x*	general modes variables in perturbation grid
in	*mds_sol*	general modes variables in solution grid
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*grid_sol*	solution grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x*	perturbation variables
in	*sol*	solution variables
in	*xyz*	X, Y and Z of extended eq_grid
in	*x_id*	nr. of Eigenvalue (for output name)
in	*plot_var*	whether variables are plotted (1: plasma perturbation, 2: magnetic perturbation)

Definition at line 116 of file sol_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.35.2.6 print_output_sol()

```
integer function, public sol_ops::print_output_sol (
               type(grid_type), intent(in) grid,
               type(sol_type), intent(in) sol,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional remove_previous_arrs )
```

Print solution quantities to an output file:

- sol:
  - `val`
  - vec

If `rich_lvl` is provided, "`_[R_rich_lvl]`" is appended to the data name if it is $>$`0`.

Returns

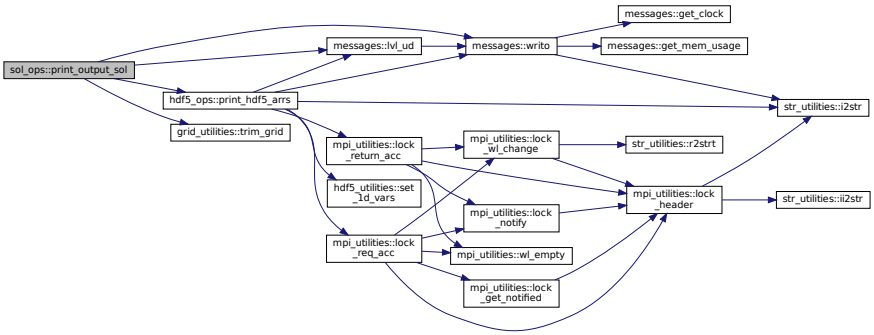
ierr

Parameters

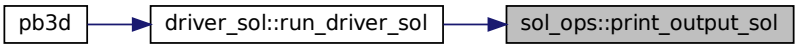
in	*grid*	solution grid variables
in	*sol*	solution variables
in	*data_name*	name under which to store
in	*rich_lvl*	Richardson level to print
in	*remove_previous_arrs*	remove previous variables if present

Definition at line 1566 of file sol_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.35.3 Variable Documentation

B.35.3.1 debug_calc_e

```
logical, public sol_ops::debug_calc_e = .false.
```

plot debug information for calc_E()

Note

Debug version only

Definition at line 29 of file sol_ops.f90.

B.35.3.2 debug_du

```
logical, public sol_ops::debug_du = .false.
```

plot debug information for calculation of `DU`

Note

Debug version only

Definition at line 31 of file sol_ops.f90.

B.35.3.3 debug_plot_sol_vec

```
logical, public sol_ops::debug_plot_sol_vec = .false.
```

plot debug information for plot_sol_vec()

Note

Debug version only

Definition at line 27 of file sol_ops.f90.

B.35.3.4 debug_x_norm

```
logical, public sol_ops::debug_x_norm = .false.
```

plot debug information `X_norm`

Note

Debug version only

Definition at line 30 of file sol_ops.f90.

B.36 sol_utilities Module Reference

Numerical utilities related to the solution vectors.

Interfaces and Types

- interface calc_xuq

  Calculates the normal ($\cdot_n$) or geodesic ($\cdot_g$) components of the plasma perturbation $\vec{\xi}$ or the magnetic perturbation $\vec{Q} = \nabla \times \left(\vec{x} \times \vec{B} \right)$.

Functions/Subroutines

- integer function, public calc_tot_sol_vec (mds, grid_sol, sol_vec_loc, sol_vec_tot, deriv)

  Calculate the total version of the solution vector from the local version.
- integer function, public calc_loc_sol_vec (mds, i_min, sol_vec_tot, sol_vec_loc)

  Calculate the local version of the solution vector from the total version.
- integer function, public interp_v_spline (V_i, V_o, r_i, r_o, extrap, ivs_stat)

  Interpolation for a quantity V using splines.

Variables

- logical, public debug_calc_xuq_arr = .false.

  plot debug information for calc_XUQ_arr
- logical, public debug_interp_v_spline = .false.

  debug information for interp_v_spline

B.36.1 Detailed Description

Numerical utilities related to the solution vectors.

B.36.2 Function/Subroutine Documentation

B.36.2.1 calc_loc_sol_vec()

```
integer function, public sol_utilities::calc_loc_sol_vec (
               type(modes_type), intent(in) mds,
               integer, intent(in) i_min,
               complex(dp), dimension(:,:), intent(in) sol_vec_tot,
               complex(dp), dimension(:,:), intent(inout), allocatable sol_vec_loc )
```

Calculate the local version of the solution vector from the total version.

See also

See calc_tot_sol_vec().

Returns

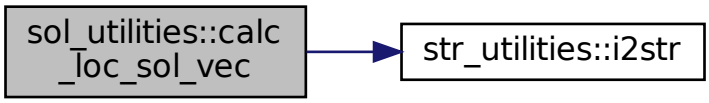
ierr

Parameters

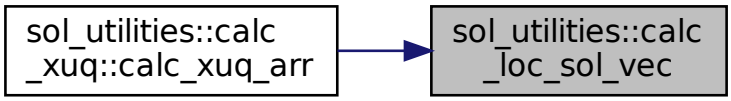
in	*mds*	general modes variables
in	*i_min*	i_min of grid in which variables are tabulated
in	*sol_vec_tot*	solution vector for all possible resonating modes
in,out	*sol_vec_loc*	solution vector for local resonating modes

Definition at line 722 of file sol_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.36.2.2 calc_tot_sol_vec()

```
integer function, public sol_utilities::calc_tot_sol_vec (
                type(modes_type), intent(in) mds,
                type(grid_type), intent(in) grid_sol,
                complex(dp), dimension(:,:), intent(in) sol_vec_loc,
                complex(dp), dimension(:,:), intent(inout), allocatable sol_vec_tot,
                integer, intent(in), optional deriv )
```

Calculate the total version of the solution vector from the local version.

This is the solution vector for all of the possible mode numbers, which can be different from the local mode numbers for X style 2 (fast):

- local: the size of the saved perturbation and solution variables, prescribed by the user as input:
  - either directly through `n_mod_X`
  - or through the limits on the modes using `min_sec_X` and `max_sec_X`.
- total: all the possible modes that can resonate in the plasma, which can be different from the local number for `X_style` 2, as the mode numbers depend on the normal coordinate.

If the output variable is not allocated, it is done here.

Optionally a derivative can be requested, depending on the normal discretization order.

Note

Need to provide the grid in which mds is tabulated.

Returns

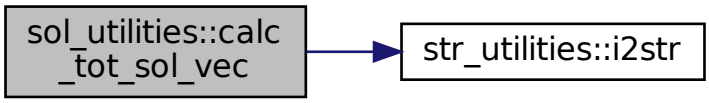
ierr

Parameters

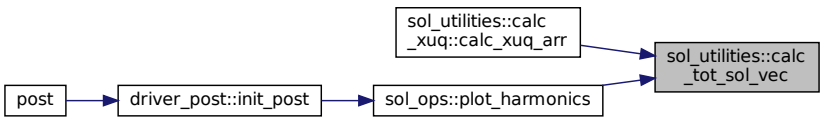
`in`	*mds*	general modes variables
`in`	*grid_sol*	solution grid
`in`	*sol_vec_loc*	solution vector for local resonating modes
`in,out`	*sol_vec_tot*	solution vector for all possible resonating modes
`in`	*deriv*	order of derivative

Definition at line 581 of file sol_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.36.2.3 interp_v_spline()

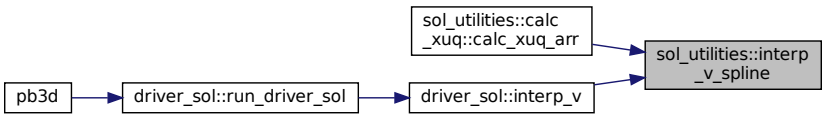
```
integer function, public sol_utilities::interp_v_spline (
               complex(dp), dimension(:,:,:), intent(in) V_i,
               complex(dp), dimension(:,:,:), intent(out) V_o,
               real(dp), dimension(:), intent(in) r_i,
               real(dp), dimension(:), intent(in) r_o,
               logical, intent(in) extrap,
               integer, intent(out), optional ivs_stat )
```

Interpolation for a quantity V using splines.

Optionally, a variable 'ivs_stat' is returned that indicates whether the interpolation was a plain copy (1), a linear interpolation (2) or a spline interpolation (3).

Definition at line 801 of file sol_utilities.f90.

Here is the caller graph for this function:



B.36.3 Variable Documentation

B.36.3.1 debug_calc_xuq_arr

```
logical, public sol_utilities::debug_calc_xuq_arr = .false.
```

plot debug information for calc_XUQ_arr

Note

Debug version only

Definition at line 25 of file sol_utilities.f90.

B.36.3.2 debug_interp_v_spline

```
logical, public sol_utilities::debug_interp_v_spline = .false.
```

debug information for interp_v_spline

Note

Debug version only

Definition at line 26 of file sol_utilities.f90.

B.37 sol_vars Module Reference

Variables pertaining to the solution quantities.

Interfaces and Types

- type sol_type
  solution type

Functions/Subroutines

- subroutine init_sol (sol, mds, grid_sol, n_EV, lim_sec_X)
  Initialize a solution type and allocate the variables.
- subroutine dealloc_sol (sol)
  Deallocates solution variables.

Variables

- integer, public n_alloc_sols
  nr. of allocated grids

B.37.1 Detailed Description

Variables pertaining to the solution quantities.

B.37.2 Function/Subroutine Documentation

B.37.2.1 dealloc_sol()

```
subroutine sol_vars::dealloc_sol (
              class(sol_type), intent(inout) sol )
```

Deallocates solution variables.

Note

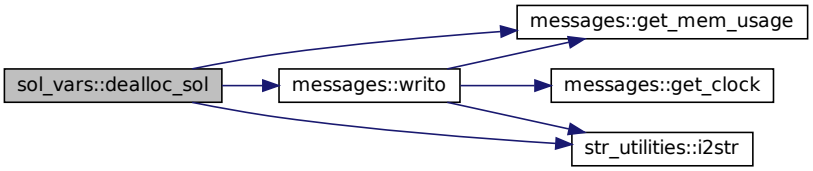
intent(out) automatically deallocates the variable

Parameters

in,out	*sol*	solution variables to be deallocated

Definition at line 106 of file sol_vars.f90.

Here is the call graph for this function:



B.37.2.2 init_sol()

```
subroutine sol_vars::init_sol (
               class(sol_type), intent(inout) sol,
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_sol,
               integer, intent(in) n_EV,
               integer, dimension(2), intent(in), optional lim_sec_X )
```

Initialize a solution type and allocate the variables.

The number of modes as well as n and m are also set up.

Optionally, the secondary mode number can be specified (m if poloidal flux is used as normal coordinate and c n if toroidal flux). By default, they are taken from the global X_vars variables.

Note

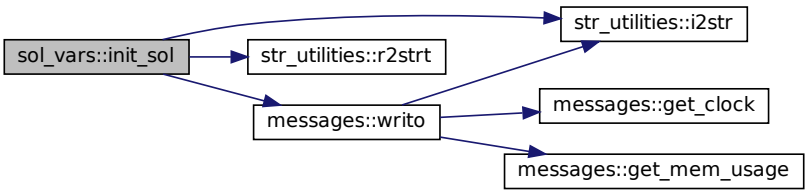
If the lowest limits of the grid is not 1 (e.g. grid_soli_min = 1 for first process), the input variable i_min should be set to set correctly. For a full grid, it should be set to 1.

Parameters

in,out	*sol*	solution variables
in	*mds*	general modes variables
in	*grid_sol*	solution grid
in	*n_ev*	nr. of Eigenvalues
in	*lim_↩ sec_x*	limits of m_X (pol. flux) or n_X (tor. flux)

Definition at line 60 of file sol_vars.f90.

Here is the call graph for this function:



B.37.3 Variable Documentation

B.37.3.1 n_alloc_sols

```
integer, public sol_vars::n_alloc_sols
```

nr. of allocated grids

Note

Debug version only

Definition at line 22 of file sol_vars.f90.

B.38 str_utilities Module Reference

Operations on strings.

Functions/Subroutines

- elemental character(len=max_str_ln) function, public i2str (k)
  Convert an integer to string.
- elemental character(len=max_str_ln) function, public ii2str (k)
  Convert an integer to string.
- elemental character(len=max_str_ln) function, public r2str (k)
  Convert a real (double) to string.
- elemental character(len=max_str_ln) function, public r2strt (k)
  Convert a real (double) to string.
- elemental character(len=max_str_ln) function, public c2str (k)
  Convert a complex (double) to string.
- elemental character(len=max_str_ln) function, public c2strt (k)
  Convert a complex (double) to string.
- character(len(input_string)) function, public strh2l (input_string)
  Convert a string to lowercase.
- character(len(input_string)) function, public strl2h (input_string)
  convert a string to uppercase.
- character((len(input_strings)+2) ∗size(input_strings)) function, public merge_strings (input_strings)
  Merge array of strings.

B.38.1 Detailed Description

Operations on strings.

B.38.2 Function/Subroutine Documentation

B.38.2.1 c2str()

```
elemental character(len=max_str_ln) function, public str_utilities::c2str (
               complex(dp), intent(in) k )
```

Convert a complex (double) to string.

Note

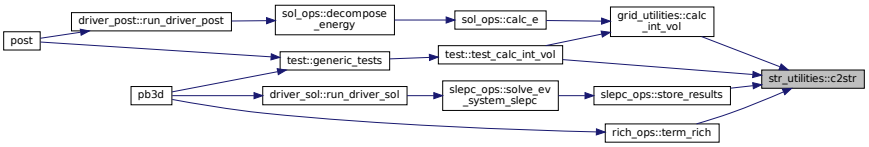
See http://www.fortran90.org/src/best-practices.html how to not lose precision.

Parameters

in	*k*	complex to convert

Definition at line 66 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.2 c2strt()

```
elemental character(len=max_str_ln) function, public str_utilities::c2strt (
              complex(dp), intent(in) k )
```

Convert a complex (double) to string.

Version with less precise output.

See also

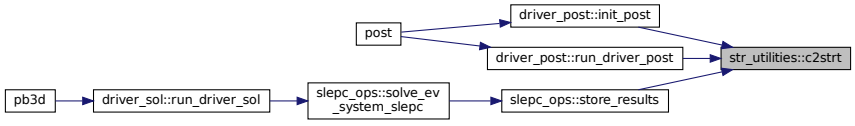
See c2str().

Parameters

in	*k*	complex to convert

Definition at line 88 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.3 i2str()

```
elemental character(len=max_str_ln) function, public str_utilities::i2str (
              integer, intent(in) k )
```

Convert an integer to string.

See also

from http://stackoverflow.com/questions/1262695/converting-integers-to-strings-in-fortran

Parameters

in	*k*	integer to convert

Definition at line 18 of file str_utilities.f90.

B.38.2.4 ii2str()

```
elemental character(len=max_str_ln) function, public str_utilities::ii2str (
               integer(kind=8), intent(in) k )
```

Convert an integer to string.

Version with kind 8 integers.

See also

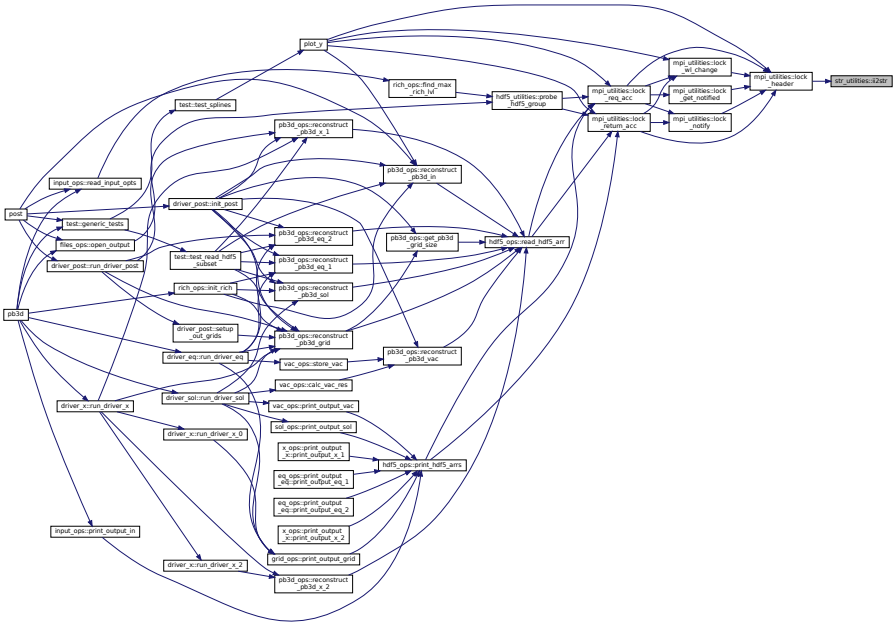
See i2str().

Parameters

in	*k*	integer to convert

Definition at line 30 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.5 merge_strings()

```
character((len(input_strings)+2)*size(input_strings)) function, public str_utilities::merge_strings (
            character(*), dimension(:), intent(in) input_strings )
```

Merge array of strings.

Parameters

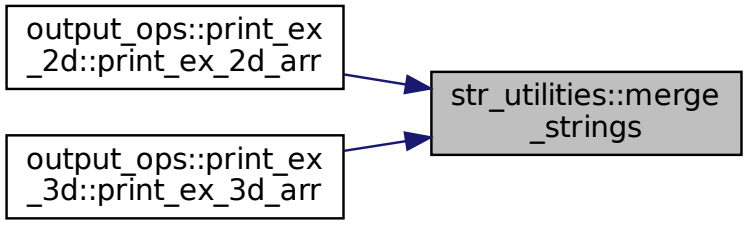
in	*input_strings*	array of strings

Returns

merged string

Definition at line 152 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.6 r2str()

```
elemental character(len=max_str_ln) function, public str_utilities::r2str (
               real(dp), intent(in) k )
```

Convert a real (double) to string.

Note

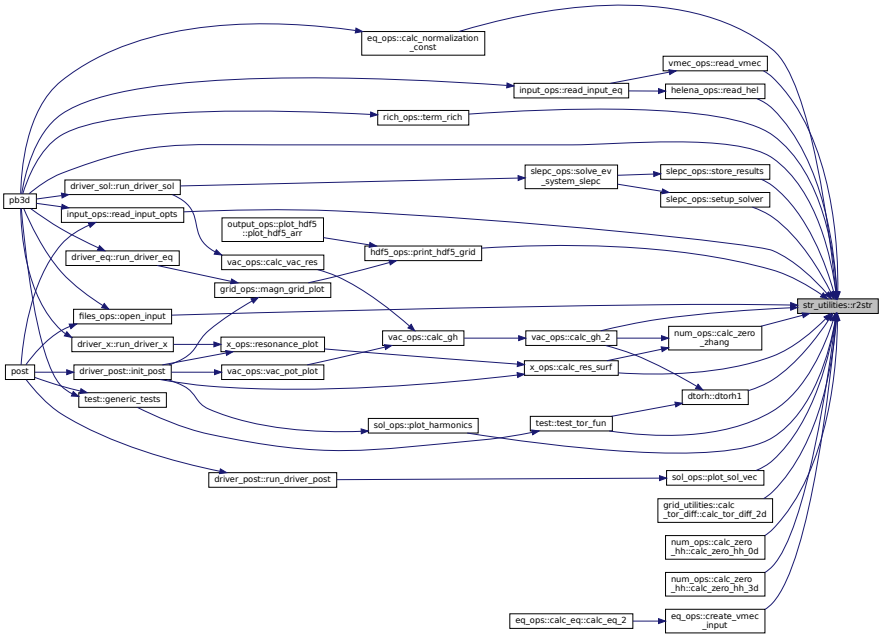
See http://www.fortran90.org/src/best-practices.html how to not lose precision.

Parameters

in	*k*	real to convert

Definition at line 42 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.7 r2strt()

```
elemental character(len=max_str_ln) function, public str_utilities::r2strt (
              real(dp), intent(in) k )
```

Convert a real (double) to string.

Version with less precise output.

See also

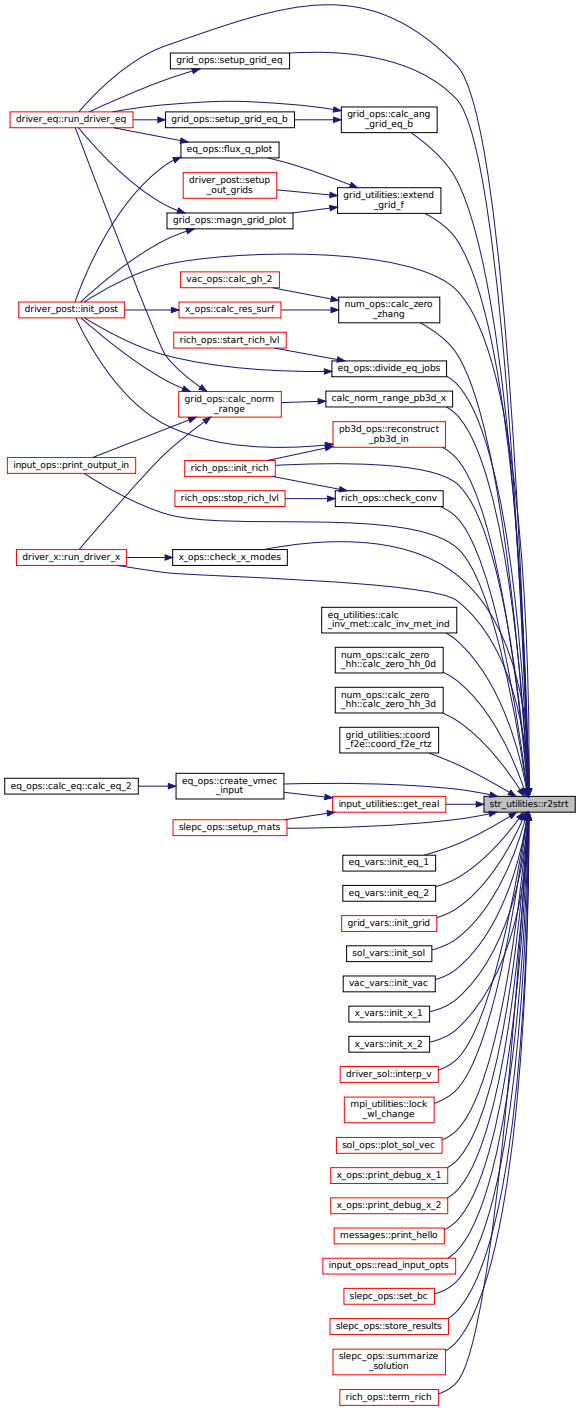
See r2str().

Parameters

in	*k*	real to convert

Definition at line 54 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.8 strh2l()

```
character(len(input_string)) function, public str_utilities::strh2l (
            character(*), intent(in) input_string )
```

Convert a string to lowercase.

See also

from [12], figure 3.5B, pg 80.

Parameters

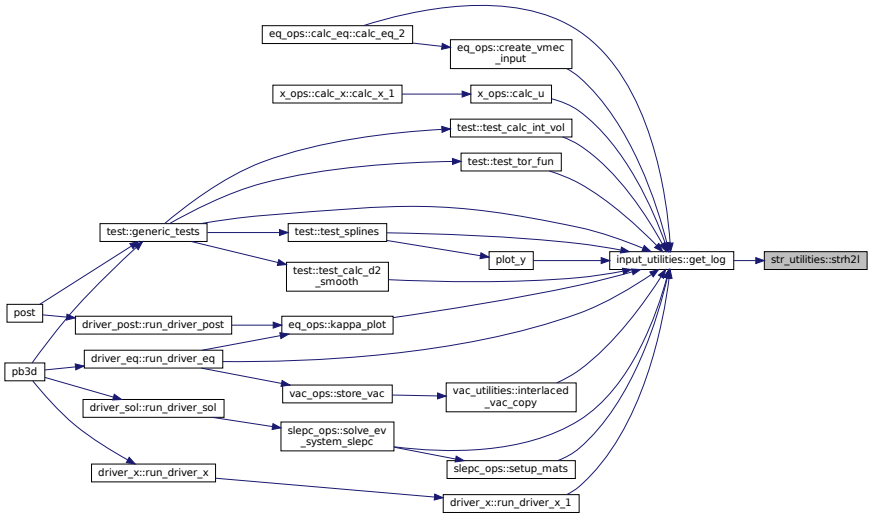
in	*input_string*	input string

Returns

lowercase version

Definition at line 109 of file str_utilities.f90.

Here is the caller graph for this function:



B.38.2.9 strl2h()

```
character(len(input_string)) function, public str_utilities::strl2h (
              character(*), intent(in) input_string )
```

convert a string to uppercase.

See also

See strh2l()

Parameters

in	*input_string*	input string

Returns

uppercase version

Definition at line 131 of file str_utilities.f90.

B.39 test Module Reference

Generic tests.

Functions/Subroutines

- integer function, public generic_tests ()
  Performs generic tests.
- integer function test_calc_d2_smooth ()
  Test calc_d2_smooth().
- integer function test_splines ()
  Test spline().
- integer function test_tor_fun ()
  Test calculation of toroidal functions.
- integer function test_read_hdf5_subset ()
  Tests reading of HDF5 subset.
- integer function test_calc_int_vol ()
  Tests calculation of volume integral.

B.39.1 Detailed Description

Generic tests.

B.39.2 Function/Subroutine Documentation

B.39.2.1 generic_tests()

```
integer function, public test::generic_tests
```

Performs generic tests.

These tests are run in interactive fashion by running the program with the `--test` command-line option.

See also

init_files()

Note

1. It is probably also recommended to run it with `--do_execute_command_line` in order to generate the plots in real time.
2. It has not been tested whether testing routines "pollutes" the further simulations. It is therefore best to not use the tests when doing a real run.

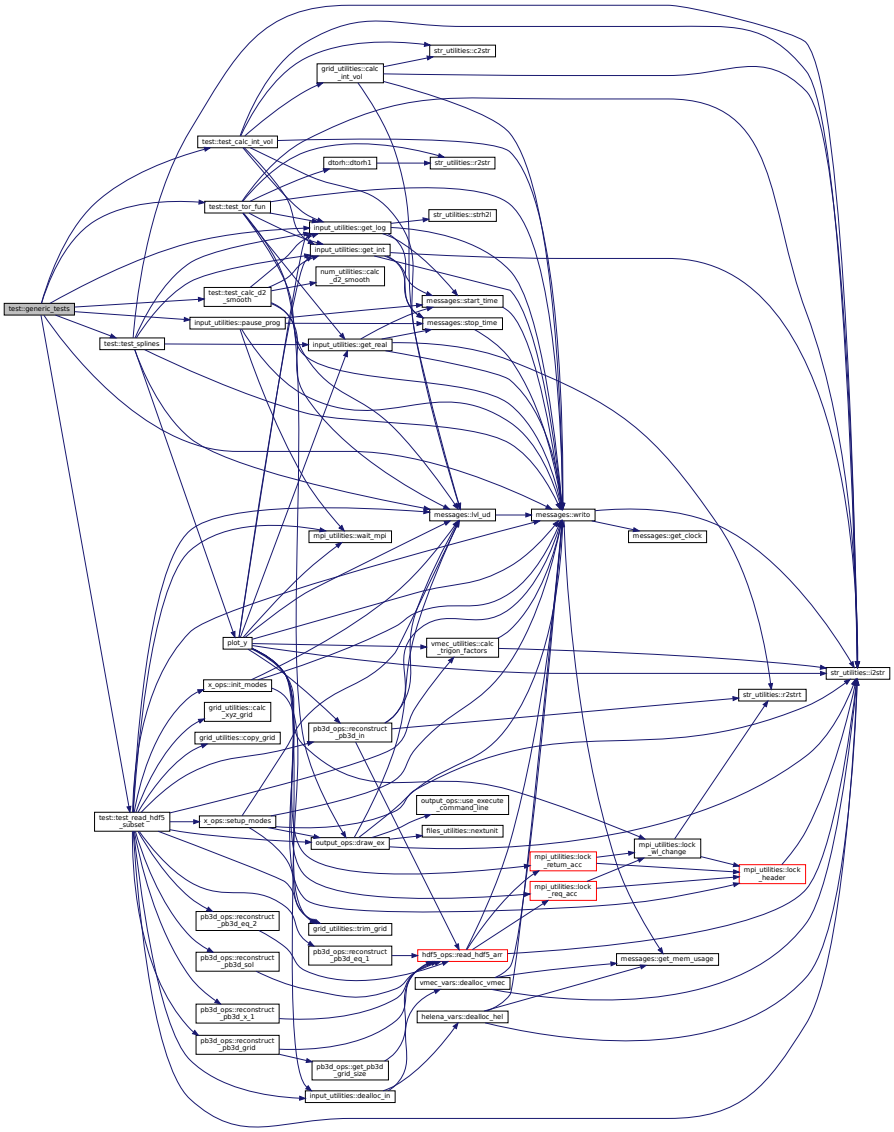
Debug version only

Returns

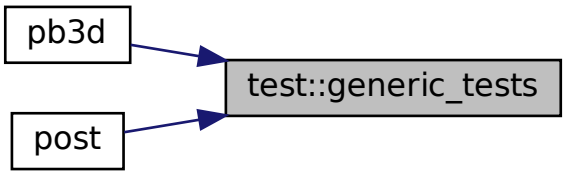
ierr

Definition at line 39 of file test.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.39.2.2 test_calc_d2_smooth()

```
integer function test::test_calc_d2_smooth
```

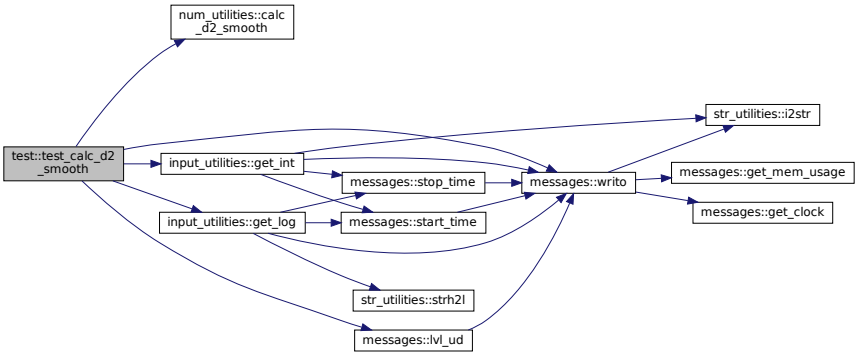
Test calc_d2_smooth().

Returns

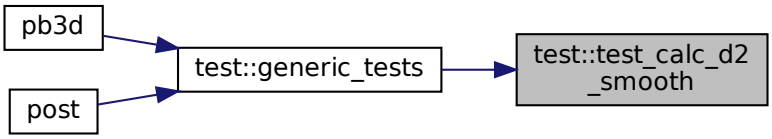
ierr

Definition at line 106 of file test.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.39.2.3 test_calc_int_vol()

```
integer function test::test_calc_int_vol
```

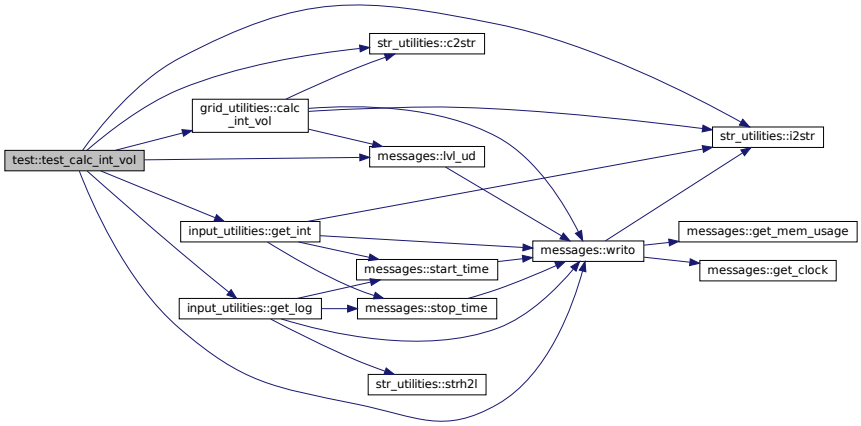
Tests calculation of volume integral.

Returns

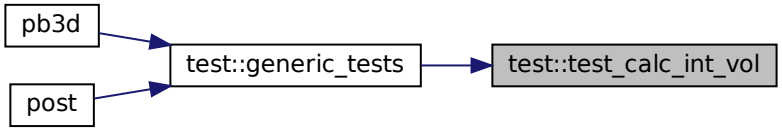
ierr

Definition at line 1335 of file test.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.39.2.4 test_read_hdf5_subset()

```
integer function test::test_read_hdf5_subset
```

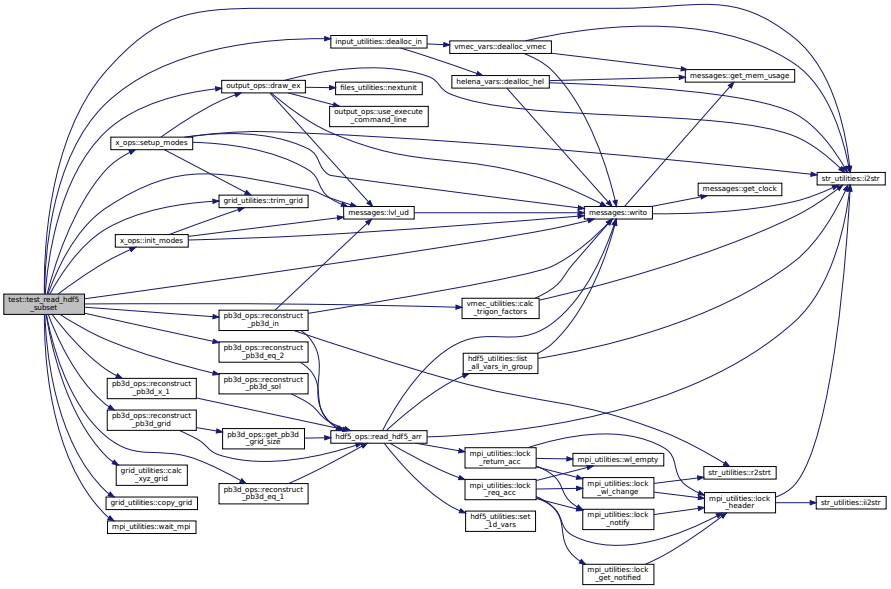
Tests reading of HDF5 subset.

Returns

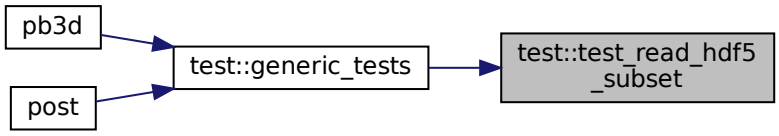
ierr

Definition at line 806 of file test.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.39.2.5 test_splines()

```
integer function test::test_splines
```

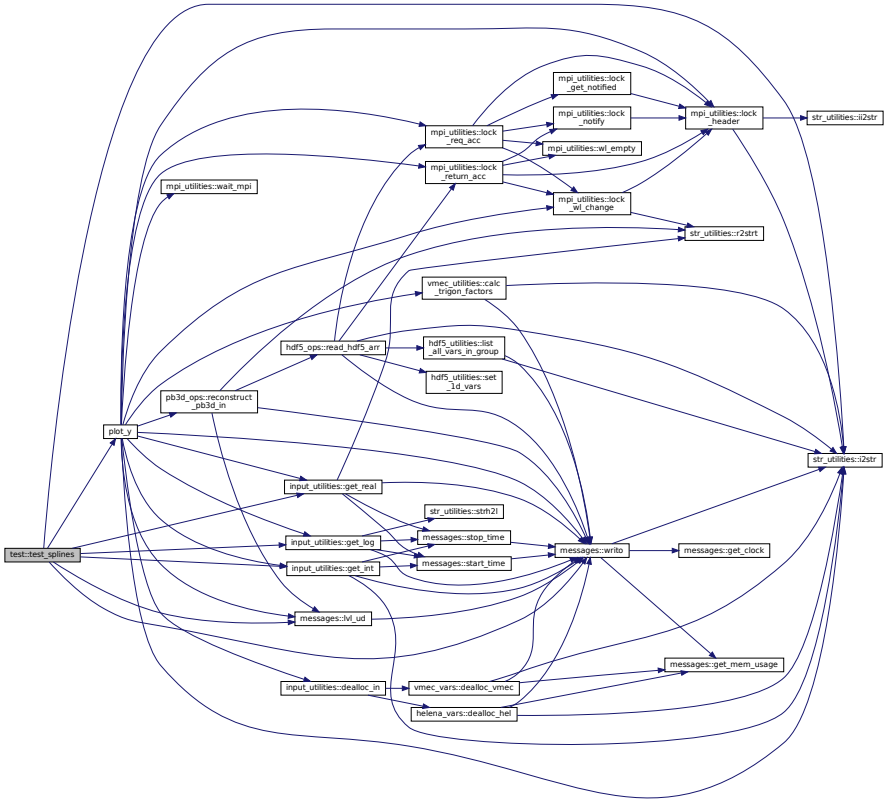
Test spline().

Returns

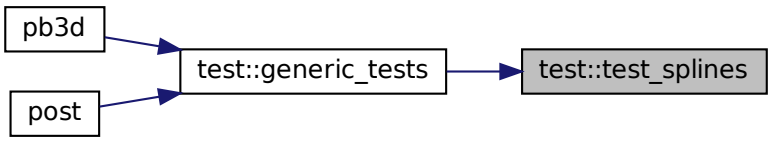
ierr

Definition at line 188 of file test.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.39.2.6 test_tor_fun()

```
integer function test::test_tor_fun
```

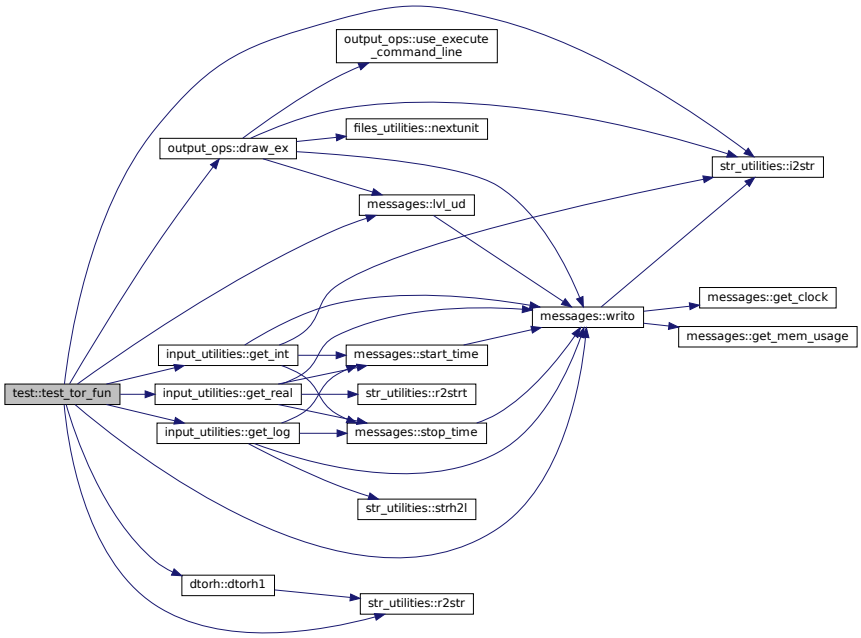
Test calculation of toroidal functions.

Returns

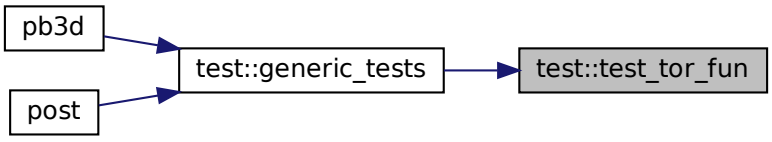
ierr

Definition at line 664 of file test.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40 vac_ops Module Reference

Operations and variables pertaining to the vacuum response.

Functions/Subroutines

- integer function, public store_vac (grid, eq_1, eq_2, vac)
  Stores calculation of the vacuum response by storing the necessary variables.
- integer function calc_gh (vac, n_r_in, lims_r_in, x_vec_in, G_in, H_in)
  Calculate the matrices `G` and `H`.
- integer function calc_gh_1 (vac, n_r_in, lims_r_in, x_vec_in, G_in, H_in, ext_in)
  Calculates matrices `G` and `H` in 3-D configuration.
- integer function calc_gh_2 (vac, n_r_in, lims_r_in, x_vec_in, G_in, H_in, ext_in)
  Calculates matrices `G` and `H` in axisymmetric configurations.
- integer function, public calc_vac_res (mds, vac)
  Calculates the vacuum response.
- integer function, public vac_pot_plot (grid_sol, vac, sol, X_id)
  Calculate vacuum potential at some positions that are not resonant.
- integer function solve_phi_bem (vac, R, Phi, n_RPhi, n_loc_RPhi, lims_c_RPhi, desc_RPhi)
  Calculates potential `Phi` on boundary of BEM in terms of some right-hand side .
- integer function, public print_output_vac (vac, data_name, rich_lvl)
  Print vacuum quantities of a certain order to an output file.

Variables

- logical, public debug_calc_gh = .false.
  plot debug information for calc_GH()
- logical, public debug_calc_vac_res = .false.
  plot debug information for calc_vac_res()
- logical, public debug_vac_pot_plot = .false.
  plot debug information for vac_pot_plot()

B.40.1 Detailed Description

Operations and variables pertaining to the vacuum response.

The vacuum response is calculated using the Boundary Element Method. There are two possibilities, indicated by the variable `style` in `vac_vars`.

1. 3-D field aligned:
   - Makes use of the collocation method for the grid points along the magnetic field lines.
   - The 3-D integrals are therefore integrated using Weyl's theorem [6], p. 5.
2. axisymmetric:
   - Makes use of an analytical expression for the toroidal integration of the Green's functions, as shown in [3].
   - The integration in the poloidal angle is done using the collocation method.

The final matrix equation is solved through Strumpack [10].

See also

See [15].

Todo The vacuum part of PB3D is still under construction and not usable yet.

B.40.2 Function/Subroutine Documentation

B.40.2.1 calc_gh()

```
integer function vac_ops::calc_gh (
               type(vac_type), intent(inout), target vac,
               integer, intent(in), optional n_r_in,
               integer, dimension(:,:), intent(in), optional, target lims_r_in,
               real(dp), dimension(:,:), intent(in), optional, target x_vec_in,
               real(dp), dimension(:,:), intent(in), optional, target G_in,
               real(dp), dimension(:,:), intent(in), optional, target H_in )
```

Calculate the matrices G and H.

This is done by iterating over the subintervals, calculating the contributions to the potential values on the left and right boundary of the interval, and adding these to the appropriate values in G and H.

As each process in the blacs grid only has access to its own values, an extra interval is is calculated to the left and right of the internal process grid edges.

Optionally, this procedure can be used to calculate the coefficients G and H with respect to other points, outside of the boundary. This is useful when the potential is to be calculated at external points.

In this case, however, no (near-) singular points are calculated, and if they appear any way, perhaps by accident, they will not be accurate.

\Note Multiple field lines are stored sequentially, which implies that the integral between the last point on a field line and the first point on the next field lines needs to be left out.

Todo For 3-D vacuums, step sizes are constant. Subsequent Richardson levels should therefore make use of the fact that the contribution to the points inherited from the previous levels can be just scaled by 1/2 and do not need to be recalculated. Currently, the copy is done correctly in interlaced_vac_copy(), but they are afterwards overwritten.

Returns

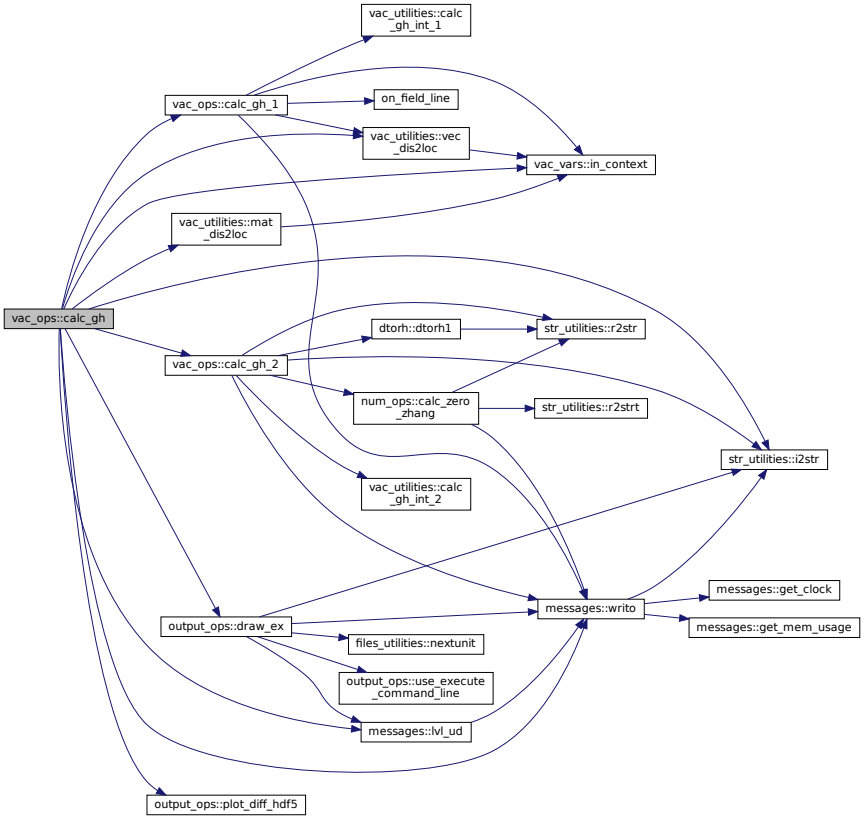
ierr

Parameters

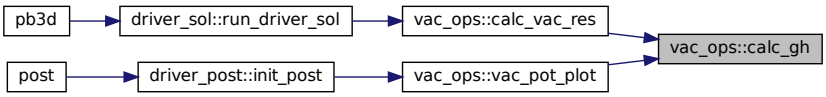
in,out	*vac*	vacuum variables
in	*n_r_in*	total number of rows of external poins of influence
in	*lims_r↩ _in*	row limits of external points of influence
in	*x_vec_in*	external position of influence
in	*g_in*	external G
in	*h_in*	external H

Definition at line 501 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40.2.2 calc_gh_1()

```
integer function vac_ops::calc_gh_1 (
              type(vac_type), intent(inout) vac,
              integer, intent(in) n_r_in,
              integer, dimension(:,:), intent(in) lims_r_in,
              real(dp), dimension(:,:), intent(in) x_vec_in,
              real(dp), dimension(:,:), intent(in), pointer G_in,
              real(dp), dimension(:,:), intent(in), pointer H_in,
              logical, intent(in) ext_in )
```

Calculates matrices `G` and `H` in 3-D configuration.

See also

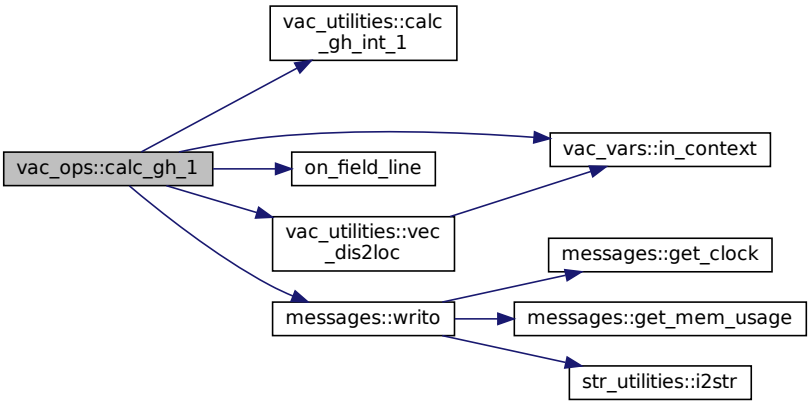
calc_gh.

Parameters

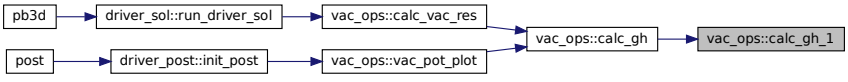
in,out	*vac*	vacuum variables
in	*n_r_in*	total number of rows of external poins of influence
in	*lims_r↩_in*	row limits of external points of influence
in	*x_vec_in*	position of influence
in	*g_in*	G at position of influence
in	*h_in*	H at position of influence
in	*ext_in*	position of influence is external

Definition at line 779 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40.2.3 calc_gh_2()

```
integer function vac_ops::calc_gh_2 (
                type(vac_type), intent(inout) vac,
                integer, intent(in) n_r_in,
                integer, dimension(:,:), intent(in) lims_r_in,
                real(dp), dimension(:,:), intent(in) x_vec_in,
                real(dp), dimension(:,:), intent(in), pointer G_in,
                real(dp), dimension(:,:), intent(in), pointer H_in,
                logical, intent(in) ext_in )
```

Calculates matrices G and H in axisymmetric configurations.

It makes use of toroidal functions.

See also

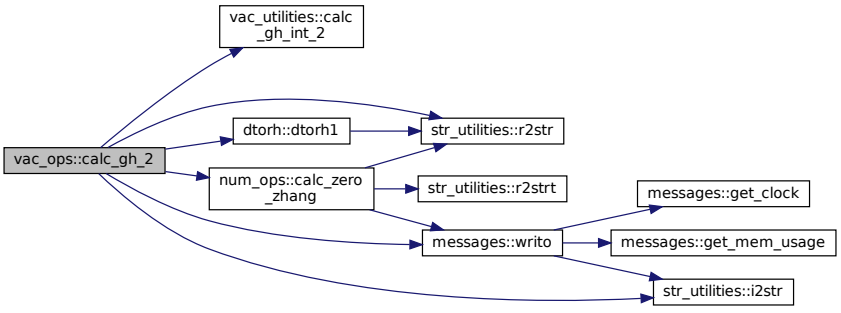
calc_gh.

Parameters

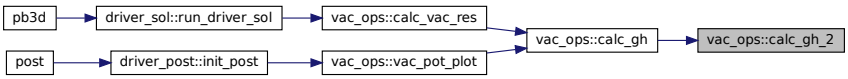
in,out	*vac*	vacuum variables
in	*n_r_in*	total number of rows of external poins of influence
in	*lims_r↩ _in*	row limits of external points of influence
in	*x_vec_in*	position of influence
in	*g_in*	G at position of influence
in	*h_in*	H at position of influence
in	*ext_in*	position of influence is external

Definition at line 983 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40.2.4 calc_vac_res()

```
integer function, public vac_ops::calc_vac_res (
               type(modes_type), intent(in) mds,
               type(vac_type), intent(inout) vac )
```

Calculates the vacuum response.

First `G` and `H` are completed if this is not the first Richardson level.

Then the vacuum contribution is calculated using the two-fold strategy of first solving $\overline{\mathrm{H}}\overline{\Phi} = \overline{\mathrm{GEP}}$, followed by left-multiplication of $\overline{\Phi}$ by $\overline{\mathrm{PE}}^{\dagger}$.

The non-square matrix $\overline{\mathrm{E}}$ contains the exponents for different mode numbers and different poloidal grid points, whereas $\overline{\mathrm{P}}$ are diagonal matrices that contain the factors $(nq - m)$.

In practice, the complex matrix $\overline{\mathrm{E}}$ is split in the two components of a real matrix twice the width.

For vacuum style 1, the poloidal grid points correspond to the paralell grid points and have to be provied by a `grid` variable.

If `jump_to_sol` is used for the current Richardson level, the vacuum quantities are not calculated, but just restored.

Currently, this procedure only works for vacuum style 2 (axisymmetric).

Returns

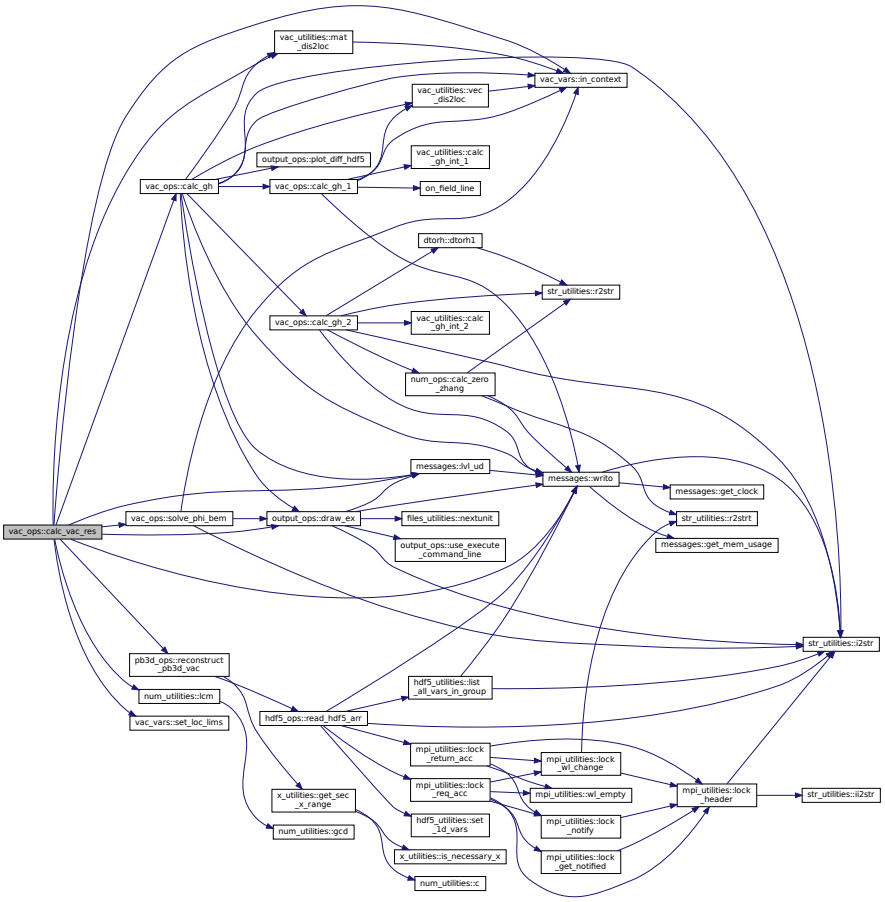
ierr

Parameters

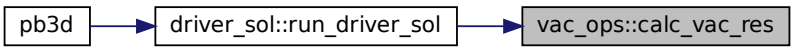
`in`	*mds*	general modes variables
`in,out`	*vac*	vacuum variables

Definition at line 1417 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40.2.5 print_output_vac()

```
integer function, public vac_ops::print_output_vac (
              type(vac_type), intent(in) vac,
              character(len=*), intent(in) data_name,
              integer, intent(in), optional rich_lvl )
```

Print vacuum quantities of a certain order to an output file.

- vac:
  - misc_vac
  - norm
  - x_vec
  - vac_res

If rich_lvl is provided, "_R_[rich_lvl]" is appended to the data name if it is >0.

Note

1. This routine should be called by a single process, in contrast to the other output routines such as eq_ops.print_output_eq(), print_output_sol(), ...
2. This process should be the last one, as it will set the boundary contribution.
3. This routine does not save the H and G matrices because they are large and foreseen to be saved directly as Hierarchical matrices in the future. They therefore need to be regenerated if Richardson restart is done, or when after a jump to solution, another Richardson level is attempted. In calc_vac, it is checked when copying the vacuum variables from previous to current Richardson level, whether the G and H matrices are allocated and if not, they are calculated.

Returns

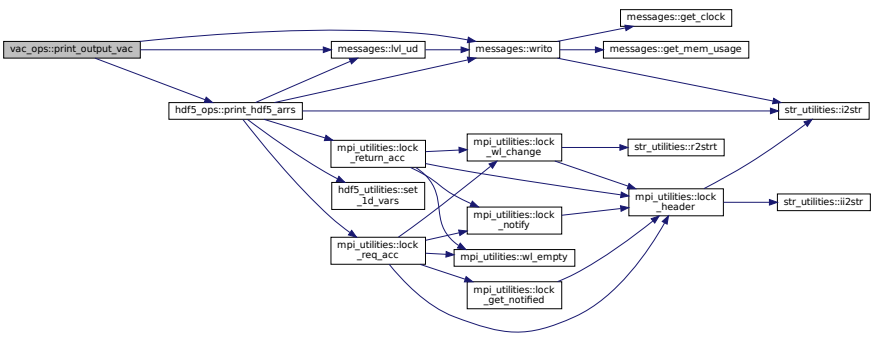
ierr

Parameters

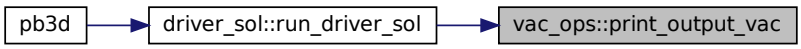
in	*vac*	vacuum variables
in	*data_name*	name under which to store
in	*rich_lvl*	Richardson level to print

Definition at line 2347 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40.2.6 solve_phi_bem()

```
integer function vac_ops::solve_phi_bem (
               type(vac_type), intent(in), target vac,
               real(dp), dimension(:,:), intent(in) R,
               real(dp), dimension(:,:), intent(inout), target Phi,
               integer, dimension(2), intent(in) n_RPhi,
               integer, dimension(2), intent(in) n_loc_RPhi,
               integer, dimension(:,:), intent(in) lims_c_RPhi,
               integer, dimension(blacsctxtsize), intent(in), target desc_RPhi )
```

Calculates potential `Phi` on boundary of BEM in terms of some right-hand side .

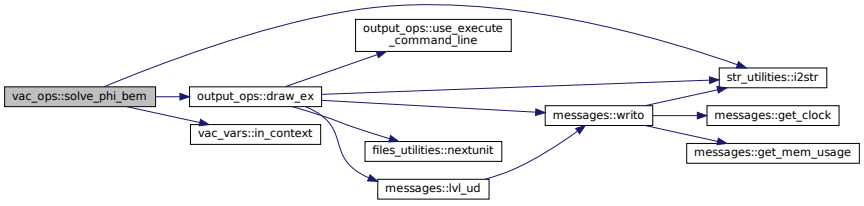
This is done by solving with STRUMPack the system of equations $\overline{\mathrm{H}}\overline{\Phi} = \overline{\mathrm{G}}\overline{\mathrm{R}}$, where $\overline{\mathrm{R}}$ is the right-hand side, for example equal to $\overline{\mathrm{EP}}$ to solve in function of the individual Fourier modes.

Parameters

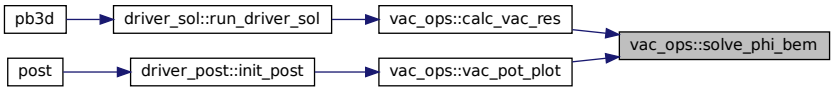
`in`	*vac*	vacuum variables
`in,out`	*phi*	$\overline{\Phi}$
`in`	*r*	$\overline{\mathrm{R}}$
`in`	*n_rphi*	n for $\overline{\mathrm{R}}$ and $\overline{\Phi}$
`in`	*n_loc_rphi*	local n for $\overline{\mathrm{R}}$ and $\overline{\Phi}$
`in`	*lims_c_rphi*	local limits for row and columns of $\overline{\mathrm{R}}$ and $\overline{\Phi}$
`in`	*desc_rphi*	descriptor for $\overline{\mathrm{R}}$ and $\overline{\Phi}$

Definition at line 2163 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



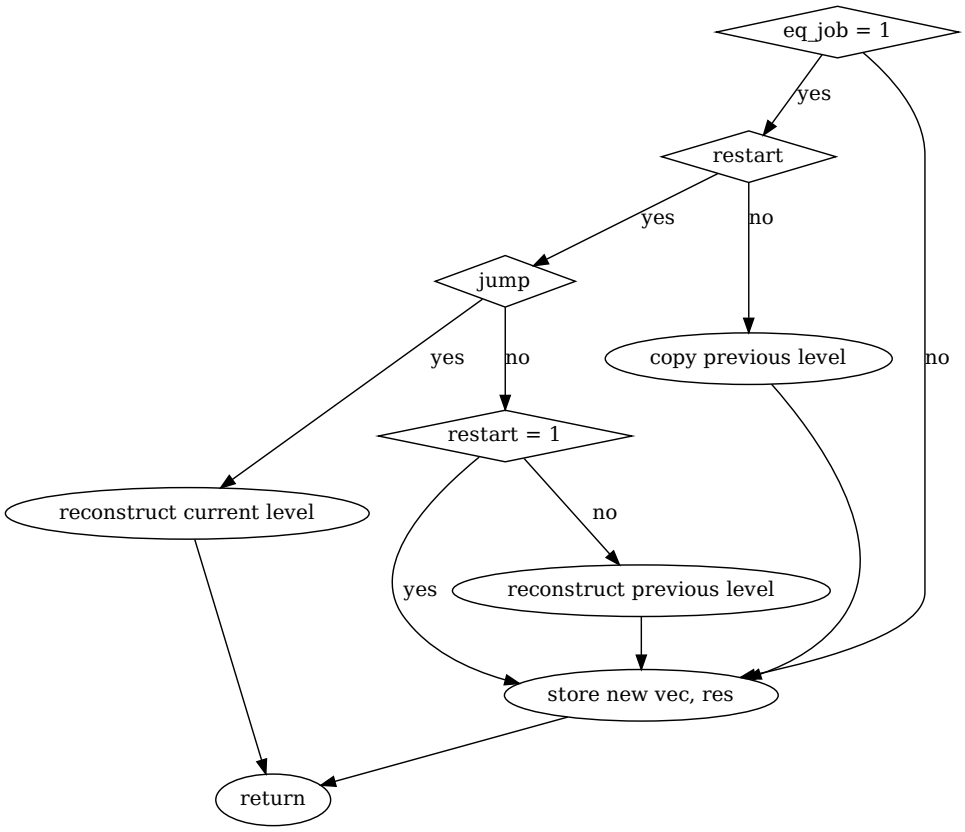
B.40.2.7 store_vac()

```
integer function, public vac_ops::store_vac (
               type(grid_type), intent(in) grid,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               type(vac_type), intent(inout) vac )
```

Stores calculation of the vacuum response by storing the necessary variables.

This is done by the last process, which is the one that contains the variables at the plasma edge, and then this is broadcasted to the other processes.

The workings of this routine are summarized in the following diagram for VMEC:



Before storing the new variables, the procedure checks the following things:

- For the first equilibrium job, this routine copies the results from the previous Richardson level into the appropriate subranges of the new vacuum variables if no restart of Richardson levels was done.
- If a restart was done and the level is greater than one, there is a reconstruction of the previous level's variables. But this will happen in calc_vac_res(), not in this procedure.

If there is a jump straight to the solution, however, the procedure returns after reconstructing the current level's variables.

If the previous G and H variables are empty, they are regenerated later, in calc_gh(). This indicates that a reconstruction happened, either because a Richardson restart was performed, or because a new Richardson level was started after a previous level in which it was jumped straight to the solution.

For HELENA, there is only 1 equilibrium job at the first Richardson level, and the vacuum has to be calculated only once then. If there is a jump to solution or if there is Richardson restart, it only needs to be reconstructed.

Returns

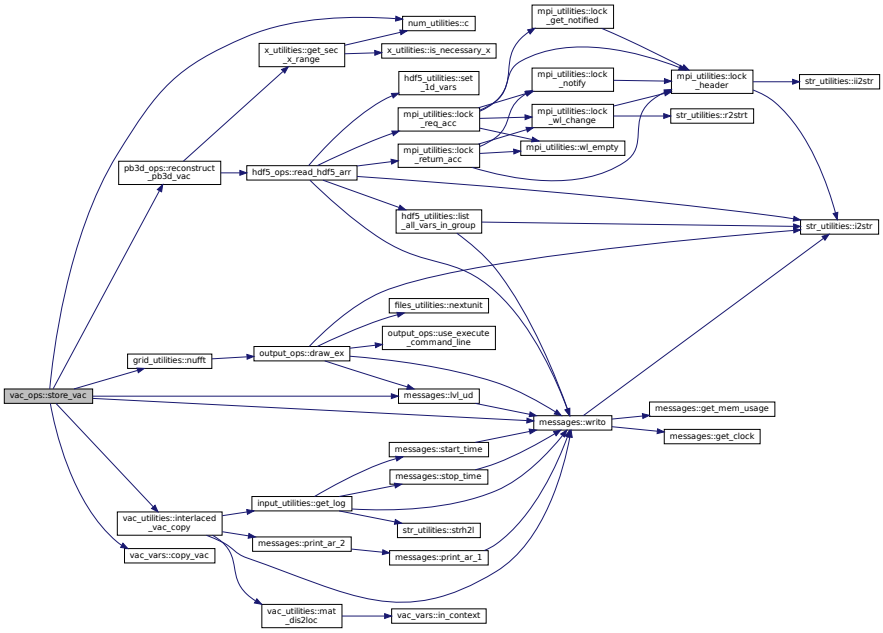
ierr

Parameters

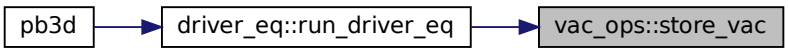
in	*grid*	equilibrium grid
in	*eq_1*	flux equilibrium variables
in	*eq_2*	metric equilibrium variables
in,out	*vac*	vacuum variables

Definition at line 113 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.40.2.8 vac_pot_plot()

```
integer function, public vac_ops::vac_pot_plot (
              type(grid_type), intent(in) grid_sol,
              type(vac_type), intent(inout) vac,
              type(sol_type), intent(in) sol,
              integer, intent(in) X_id )
```

Calculate vacuum potential at some positions that are not resonant.

This is done by using the relation between H and G on the plasma boundary, $\overline{\mathrm{H}}_\mathrm{s}\overline{\Phi}_\mathrm{s} = \overline{\mathrm{G}}_\mathrm{s}\overline{\mathrm{D}\phi}_\mathrm{s}$, where $\overline{\mathrm{D}\phi}_\mathrm{s}$ is the normal derivative of the potential on the plasma edge. The matrices $\overline{\mathrm{H}}$ and $\overline{\mathrm{G}}$ have to be calculated in advance.

This relation is then used at different positions to calculate the potential at a different position through $4\pi\phi = \left[-\overline{\mathrm{HI}}\overline{\mathrm{H}}_\mathrm{s}^{-1}\overline{\mathrm{G}}_\mathrm{s} + \overline{\mathrm{GI}}\right]\overline{\mathrm{D}\phi}_\mathrm{s}$, where $\bar{\mathrm{I}}$ is an integration rule.

This system of equations contains one row per point at which the potential is to be calculated.

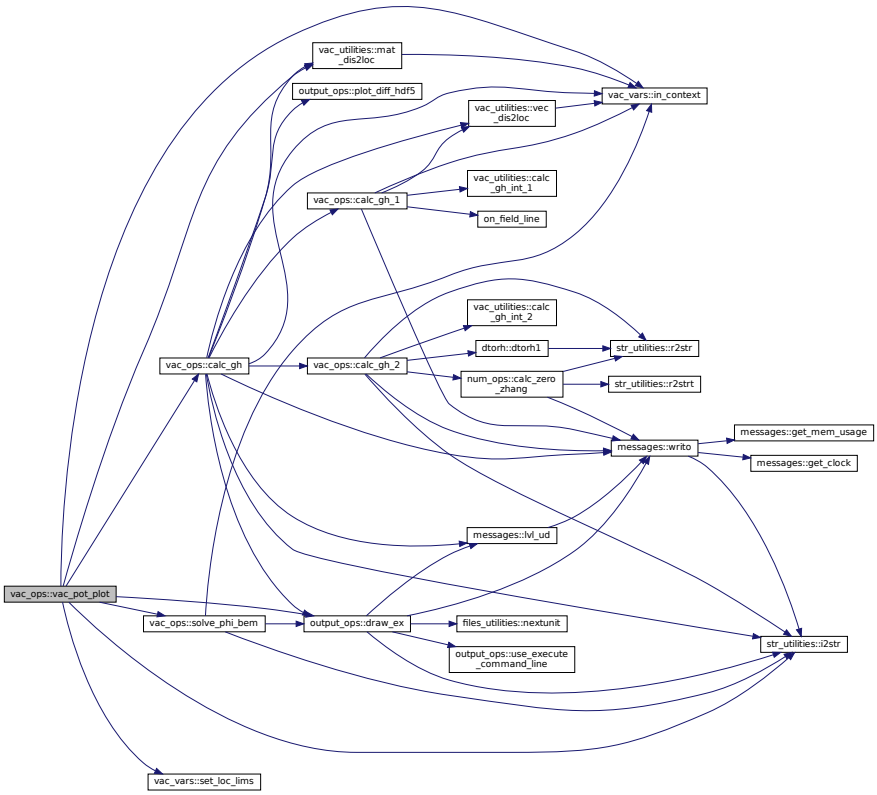
Currently, this routine creates a 3-D regular grid that is equidistant in cylindrical coordinates, with `n_vac_plot` values in R and Z, and `n_zeta_plot` values in the cylindrical angle. The limits in these variables, furthermore, are given by `min_Rvac_plot`, `max_Rvac_plot`, `min_Zvac_plot`, `max_Zvac_plot`, `min_zeta_plot` and `max_zeta_plot`.

Parameters

`in`	*grid_sol*	solution grid
`in,out`	*vac*	vacuum variables
`in`	*sol*	solution variables
`in`	*x_id*	nr. of Eigenvalue

Definition at line 1857 of file vac_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

B.40.3 Variable Documentation

B.40.3.1 debug_calc_gh

```
logical, public vac_ops::debug_calc_gh = .false.
```

plot debug information for calc_GH()

Note

Debug version only

Definition at line 46 of file vac_ops.f90.

B.40.3.2 debug_calc_vac_res

```
logical, public vac_ops::debug_calc_vac_res = .false.
```

plot debug information for calc_vac_res()

Note

Debug version only

Definition at line 47 of file vac_ops.f90.

B.40.3.3 debug_vac_pot_plot

```
logical, public vac_ops::debug_vac_pot_plot = .false.
```

plot debug information for vac_pot_plot()

Note

Debug version only

Definition at line 48 of file vac_ops.f90.

B.41 vac_utilities Module Reference

Numerical utilities related to the vacuum response.

Functions/Subroutines

- subroutine, public calc_gh_int_1 (G, H, x_s, x_in, norm_s, h_fac_in, step_size, tol)
  Calculate G_ij and H_ij on an interval for field line aligned configurations.
- subroutine, public calc_gh_int_2 (G, H, t_s, t_in, x_s, x_in, norm_s, norm_in, Aij, ql, tol, b_coeff, n_tor)
  Calculate G_ij and H_ij on an interval for axisymmetric configurations.
- integer function, public vec_dis2loc (ctxt, vec_dis, lims, vec_loc, proc)
  Gathers a distributed vector on a single process.
- integer function, public mat_dis2loc (ctxt, mat_dis, lims_r, lims_c, mat_loc, proc)
  Gathers a distributed vector on a single process.
- integer function, public interlaced_vac_copy (vac_old, vac)
  Copies vacuum variables of a previous level into the appropriate, interlaced place of a next level.

B.41.1 Detailed Description

Numerical utilities related to the vacuum response.

See also

vac_ops for information.

B.41.2 Function/Subroutine Documentation

B.41.2.1 calc_gh_int_1()

```
subroutine, public vac_utilities::calc_gh_int_1 (
               real(dp), dimension(2), intent(inout) G,
               real(dp), dimension(2), intent(inout) H,
               real(dp), dimension(2,3), intent(in) x_s,
               real(dp), dimension(3), intent(in) x_in,
               real(dp), dimension(2,3), intent(in) norm_s,
               real(dp), dimension(4), intent(in) h_fac_in,
               real(dp), dimension(2), intent(in) step_size,
               real(dp), intent(in) tol )
```

Calculate G_ij and H_ij on an interval for field line aligned configurations.

The indices for the source variables are `[left:right,dim]` where `dim` is the Cartesian dimension. The same holds for `ql`.

For subintegrals close to the singularity, the procedure uses the analytical approximation.

Note

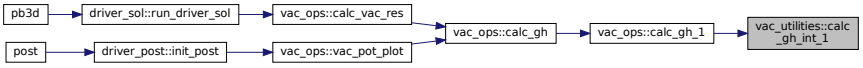
This routine does not calculate the contribution 2β.

Parameters

in	*x_s*	position vector at left and right limit of source interval
in	*x_in*	position vector at which to calculate influence
in	*norm_s*	normal vector at left and right limit of source interval
in	*h_fac_in*	metric factors at which to calculate influence
in	*step_size*	step sizes in parallel direction and alpha
in	*tol*	tolerance on distance between points

Definition at line 33 of file vac_utilities.f90.

Here is the caller graph for this function:



B.41.2.2 calc_gh_int_2()

```
subroutine, public vac_utilities::calc_gh_int_2 (
               real(dp), dimension(2), intent(inout) G,
               real(dp), dimension(2), intent(inout) H,
               real(dp), dimension(2), intent(in) t_s,
               real(dp), intent(in) t_in,
               real(dp), dimension(2,2), intent(in) x_s,
               real(dp), dimension(2), intent(in) x_in,
               real(dp), dimension(2,2), intent(in) norm_s,
               real(dp), dimension(2), intent(in) norm_in,
               real(dp), dimension(2), intent(in) Aij,
               real(dp), dimension(2,2), intent(in) ql,
               real(dp), intent(in) tol,
               real(dp), dimension(2), intent(in) b_coeff,
               integer, intent(in) n_tor )
```

Calculate G_ij and H_ij on an interval for axisymmetric configurations.

The indices for the source variables are `[left:right,dim]` where `dim` is the Cartesian dimension. The same holds for `ql`.

For subintegrals close to the singularity (i.e. where the tolerance is no being met), the routine approximates the toroidal functions with the analytical approximation. If one of the boundaries of the subintegral is far enough from the singularity, there is an additional contribution due to the difference between the actual toroidal function and the approximation. This contribution is zero if both boundaries are close, because in this case it is assumed that the entire integral is given by the analytical approximation.

Note

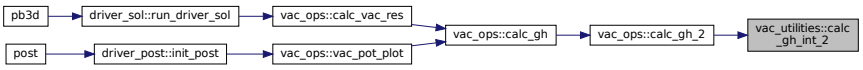
This routine does not calculate the contribution 2β.

Parameters

in	*t_s*	pol. angle at left and right limit of source interval
in	*t_in*	pol. angle at which to calculate influence
in	*x_s*	position vector at left and right limit of source interval
in	*x_in*	position vector at which to calculate influence
in	*norm↩ _s*	normal vector at left and right limit of source interval
in	*norm↩ _in*	normal vector at which to calculate influence
in	*aij*	Aij helper variable for H
in	*ql*	ql for n-1 and n at left and right limit of source interval
in	*tol*	tolerance on rho2
in	*b_coeff*	$b = \sum_{k=1}^{n} \frac{2}{2k-1}$ for n and n-1
in	*n_tor*	toroidal mode number

Definition at line 99 of file vac_utilities.f90.

Here is the caller graph for this function:



B.41.2.3 interlaced_vac_copy()

```
integer function, public vac_utilities::interlaced_vac_copy (
              type(vac_type), intent(in) vac_old,
              type(vac_type), intent(inout) vac )
```

Copies vacuum variables of a previous level into the appropriate, interlaced place of a next level.

This is done by multiplying left by $\overline{\overline{\mathsf{T}}}$ and right by $\overline{\overline{\mathsf{T}}}^T$ with

$$a_{ij} = \begin{cases} 1 & \text{for } (i,j) = (2j-1+(i-1)n_{\text{old}}, j+(i-1)n_{\text{old}}) \\ 0 & \text{otherwise} \end{cases}$$

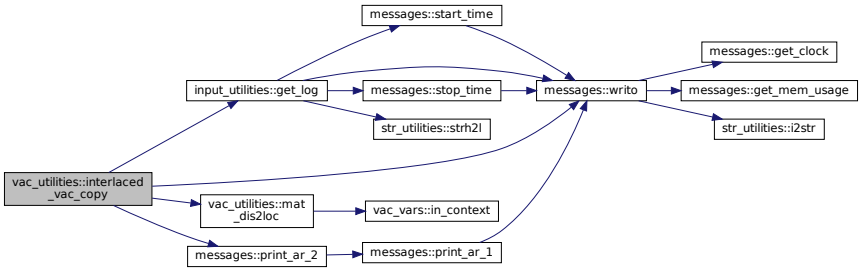
where i ranges from 1 to n_{old} and j from 1 to m_{old}, with the size of $\overline{\overline{\mathsf{T}}}$ equal to $n_{\text{new}} \times n_{\text{old}}$ where n refers to `n_bnd(1)` and m to `n_bnd(2)`.

Parameters

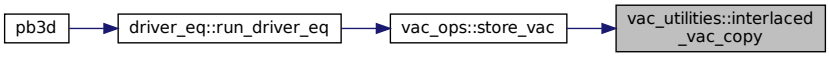
in	*vac_old*	old vacuum
in,out	*vac*	vacuum

Definition at line 355 of file vac_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.41.2.4 mat_dis2loc()

```
integer function, public vac_utilities::mat_dis2loc (
               integer, intent(in) ctxt,
               real(dp), dimension(:,:), intent(in) mat_dis,
               integer, dimension(:,:), intent(in) lims_r,
               integer, dimension(:,:), intent(in) lims_c,
               real(dp), dimension(:,:), intent(inout) mat_loc,
               integer, intent(in), optional proc )
```

Gathers a distributed vector on a single process.

See also

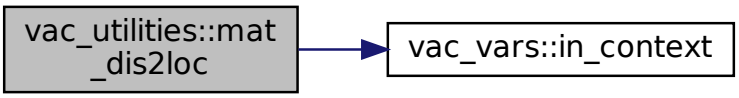
See vec_dis2loc() for exaplanation.

Parameters

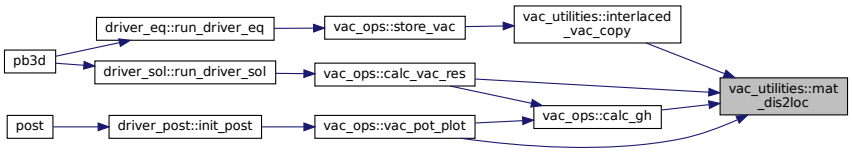
in	*ctxt*	context for matrix
in	*mat_dis*	distributed matrix
in	*lims_r*	limits for different subrows
in	*lims_c*	limits for different subcolumns
in,out	*mat_loc*	local matrix
in	*proc*	which process receives result

Definition at line 277 of file vac_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.41.2.5 vec_dis2loc()

```
integer function, public vac_utilities::vec_dis2loc (
               integer, intent(in) ctxt,
               real(dp), dimension(:), intent(in) vec_dis,
               integer, dimension(:,:), intent(in) lims,
               real(dp), dimension(:), intent(inout) vec_loc,
               integer, intent(in), optional proc )
```

Gathers a distributed vector on a single process.

For the distributed vector, the limits of the different subrows need to be provided, as discussed in init_vac().

By default, the result lands on the master process, but this can be changed using `proc`. If it is negative, all processes receive the result. In any case, `vec_loc` will be utilized.

Note

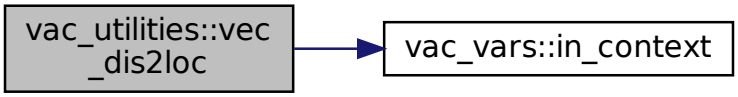
`vec_loc` needs to be allocated to the total dimensions, even when the results is not received by the process.

Parameters

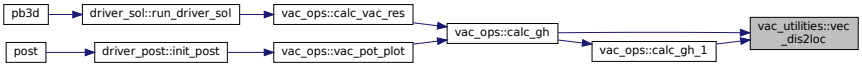
in	*ctxt*	context for vector
in	*vec_dis*	distributed vector
in	*lims*	limits for different subrows
in,out	*vec_loc*	local vector
in	*proc*	which process receives result

Definition at line 223 of file vac_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.42 vac_vars Module Reference

Variables pertaining to the vacuum quantities.

Interfaces and Types

- type vac_type
  vacuum type

Functions/Subroutines

- integer function init_vac (vac, style, n_bnd, prim_X, n_ang, jq)
  Initializes a vacuum type.
- subroutine, public set_loc_lims (n_loc, bs, ind_p, n_p, lims)
  Calculates the limits in local index.
- subroutine dealloc_vac (vac)
  Deallocates vacuum variables.
- integer function, public copy_vac (vac, vac_copy)
  Copy a vacuum type.
- logical function, public in_context (ctxt)
  Indicates whether current process is in the context.

Variables

- integer, public n_alloc_vacs
  nr. of allocated vacs

B.42.1 Detailed Description

Variables pertaining to the vacuum quantities.

B.42.2 Function/Subroutine Documentation

B.42.2.1 copy_vac()

```
integer function, public vac_vars::copy_vac (
               class(vac_type), intent(in) vac,
               class(vac_type), intent(inout) vac_copy )
```

Copy a vacuum type.

Note

The copy should be unallocated.

Parameters

in	*vac*	vac to be copied
in,out	*vac_copy*	copy

Definition at line 339 of file vac_vars.f90.

Here is the caller graph for this function:

B.42.2.2 dealloc_vac()

```
subroutine vac_vars::dealloc_vac (
               class(vac_type), intent(inout) vac )
```

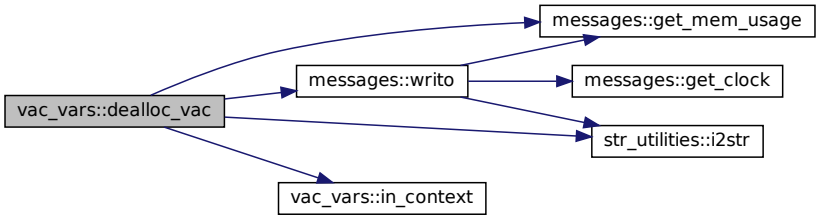
Deallocates vacuum variables.

Parameters

in,out	*vac*	vacuum variables to be deallocated

Definition at line 287 of file vac_vars.f90.

Here is the call graph for this function:



B.42.2.3 in_context()

```
logical function, public vac_vars::in_context (
               integer, intent(in) ctxt )
```

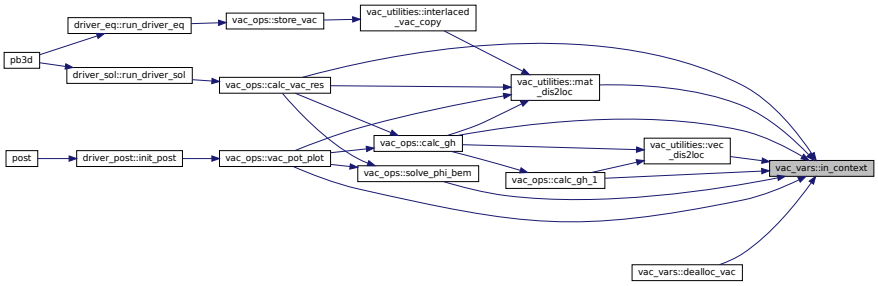
Indicates whether current process is in the context.

Parameters

in	*ctxt*	context for vector

Definition at line 378 of file vac_vars.f90.

Here is the caller graph for this function:



B.42.2.4 init_vac()

```
integer function vac_vars::init_vac (
               class(vac_type), intent(inout) vac,
               integer, intent(in) style,
               integer, intent(in) n_bnd,
               integer, intent(in) prim_X,
               integer, dimension(2), intent(in) n_ang,
               real(dp), intent(in) jq )
```

Initializes a vacuum type.

The number of modes as well as `n` and `m` are also set up.

The variables G and H are saved in a special format, based on the block-cyclical distribution employed in scalapack. As a hypothetical example, consider a process grid corresponding to block-size 1x2 and 2x3 processes with a total size of 3x15:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 \\ 3 & 3 & 4 & 4 & 5 & 5 & 3 & 3 & 4 & 4 & 5 & 5 & 3 & 3 & 4 \\ 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 \end{bmatrix}$$

Therefore, the 3-D storage convention used is, for example, for process 1:

$$\begin{bmatrix} (1,3) & (1,4) & (1,9) & (1,10) & (1,15) & (1,\cdot) \\ (3,3) & (3,4) & (3,9) & (3,10) & (3,15) & (3,\cdot) \end{bmatrix}$$

The information concerning the delimitations of the individual subintervals in the horizontal and vertical direction can be saved once per subinterval. For process 4 of above example, this would be

$$\begin{bmatrix} 2 \\ 2 \end{bmatrix};$$

for the vertical direction, and

$$\begin{bmatrix} 3 & 9 & 15 \\ 4 & 10 & 15 \end{bmatrix};$$

for the horizontal direction.

A value $\cdot$ indicates that the index is out of global bounds. Practically, this is checked by seeing whether the total index does not exceed the global bounds.

Returns

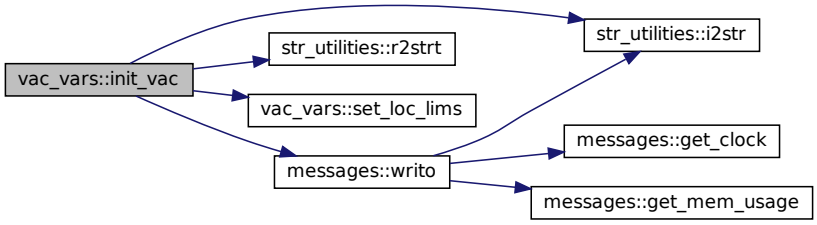
ierr

Parameters

in,out	*vac*	vacuum variables
in	*style*	style of vacuum (1: field-line 3-D, 2: axisymmetric)
in	*n_bnd*	total number of points in boundary
in	*prim↩_x*	primary mode number
in	*n_ang*	number of angles (1) and number of field lines (2)
in	*jq*	iota (tor. flux) or q (pol. flux) at edge

Definition at line 121 of file vac_vars.f90.

Here is the call graph for this function:



B.42.2.5 set_loc_lims()

```
subroutine, public vac_vars::set_loc_lims (
               integer, intent(in) n_loc,
               integer, intent(in) bs,
               integer, intent(in) ind_p,
               integer, intent(in) n_p,
               integer, dimension(:,:), intent(inout), allocatable lims )
```

Calculates the limits in local index.

See also

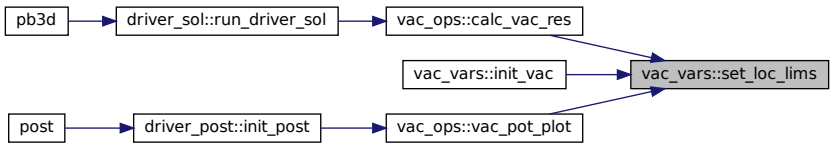
See init_var() for an explanation.

Parameters

in	*n_loc*	number of points owned by local process
in	*bs*	block size
in	*ind↩_p*	index of current process in this dimension
in	*n_p*	number of processes in this dimension
in,out	*lims*	limits for different subregions in this dimension

Definition at line 261 of file vac_vars.f90.

Here is the caller graph for this function:



B.42.3 Variable Documentation

B.42.3.1 n_alloc_vacs

```
integer, public vac_vars::n_alloc_vacs
```

nr. of allocated vacs

Note

Debug version only

Definition at line 23 of file vac_vars.f90.

B.43 vmec_ops Module Reference

Operations that concern the output of VMEC.

Functions/Subroutines

- integer function, public read_vmec (n_r_in, use_pol_flux_V)
  Reads the VMEC equilibrium data.
- subroutine, public normalize_vmec
  Normalizes VMEC input.

B.43.1 Detailed Description

Operations that concern the output of VMEC.

B.43.2 Function/Subroutine Documentation

B.43.2.1 normalize_vmec()

```
subroutine, public vmec_ops::normalize_vmec
```

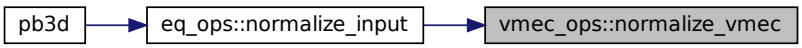
Normalizes VMEC input.

Note

The normal VMEC coordinate runs from 0 to 1, whatever the normalization.

Definition at line 328 of file VMEC_ops.f90.

Here is the caller graph for this function:



B.43.2.2 read_vmec()

```
integer function, public vmec_ops::read_vmec (
               integer, intent(inout) n_r_in,
               logical, intent(inout) use_pol_flux_V )
```

Reads the VMEC equilibrium data.

Returns

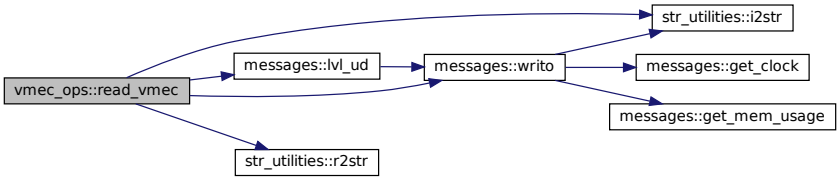
ierr

Parameters

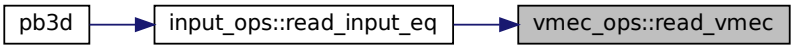
in,out	*n_r_in*	nr. of normal points in input grid
in,out	*use_pol_↩ flux_v*	.true. if VMEC equilibrium is based on poloidal flux

Definition at line 34 of file VMEC_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.44 vmec_utilities Module Reference

Numerical utilities related to the output of VMEC.

Interfaces and Types

- interface fourier2real
  Inverse Fourier transformation, from VMEC.

Functions/Subroutines

- integer function, public calc_trigon_factors (theta, zeta, trigon_factors)
  Calculate the trigonometric cosine and sine factors.

Variables

- logical, public debug_calc_trigon_factors = .false.
  plot debug information for calc_trigon_factors()

B.44.1 Detailed Description

Numerical utilities related to the output of VMEC.

B.44.2 Function/Subroutine Documentation

B.44.2.1 calc_trigon_factors()

```
integer function, public vmec_utilities::calc_trigon_factors (
               real(dp), dimension(:,:,:), intent(in) theta,
               real(dp), dimension(:,:,:), intent(in) zeta,
               real(dp), dimension(:,:,:,:,:), intent(inout), allocatable trigon_factors )
```

Calculate the trigonometric cosine and sine factors.

This is done on a grid (`1:mnmax_V`) at given 3D arrays for the (VMEC) E(quilibrium) angles θ_E and ζ_E.

The dimensions of the output array are

`(1:mnmax_V,1:n_ang_1,1:n_ang_2,1:n_r,1:2)`

where `mnmax_V` is the number of modes in VMEC `n_r` is the total number of normal points.

See also

See grid_vars.grid_type for a discussion on `ang_1` and `ang_2`.

Returns

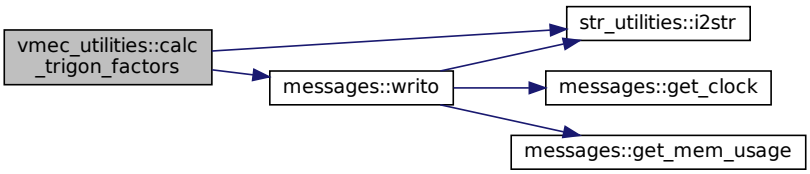
ierr

Parameters

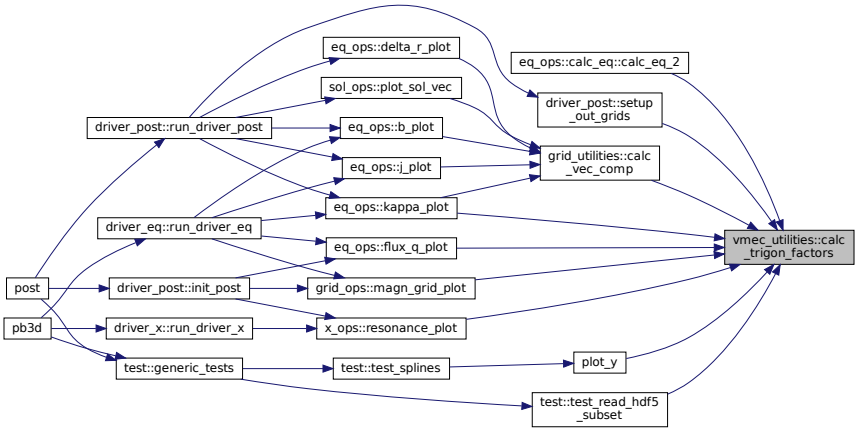
`in`	*theta*	poloidal angles in equilibrium coords.
`in`	*zeta*	toroidal angles in equilibrium coords.
`in,out`	*trigon_factors*	trigonometric factor cosine and sine at these angles

Definition at line 275 of file VMEC_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.44.3 Variable Documentation

B.44.3.1 debug_calc_trigon_factors

```
logical, public vmec_utilities::debug_calc_trigon_factors = .false.
```

plot debug information for calc_trigon_factors()

Note

Debug version only

Definition at line 22 of file VMEC_utilities.f90.

B.45 vmec_vars Module Reference

Variables that concern the output of VMEC.

Functions/Subroutines

- subroutine, public dealloc_vmec ()

  Deallocates VMEC quantities that are not used anymore.

Variables

- integer, dimension(:,:), allocatable, public mn_v
  m and n of modes
- real(dp), public b_0_v
  the magnitude of B at the magnetic axis, $\theta = \zeta = 0$
- real(dp), dimension(:,:), allocatable, public flux_t_v
  toroidal flux
- real(dp), dimension(:,:), allocatable, public flux_p_v
  poloidal flux
- real(dp), dimension(:,:), allocatable, public pres_v
  pressure
- real(dp), dimension(:,:), allocatable, public rot_t_v
  rotational transform
- real(dp), dimension(:,:), allocatable, public q_saf_v
  safety factor
- real(dp), dimension(:,:,:), allocatable, public r_v_c
  Coeff. of R in sine series (FM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public r_v_s
  Coeff. of R in cosine series (FM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public z_v_c
  Coeff. of Z in sine series (FM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public z_v_s
  Coeff. of Z in cosine series (FM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public l_v_c
  Coeff. of λ in sine series (HM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public l_v_s
  Coeff. of λ in cosine series (HM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public jac_v_c
  Coeff. of $\mathcal{J}$ in sine series (HM and FM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public jac_v_s
  Coeff. of $\mathcal{J}$ in cosine series (HM and FM) and norm. deriv.
- real(dp), dimension(:,:,:), allocatable, public b_v_sub_c
  Coeff. of B_i in sine series (r,theta,phi) (FM)
- real(dp), dimension(:,:,:), allocatable, public b_v_sub_s
  Coeff. of B_i in cosine series (r,theta,phi) (FM)
- real(dp), dimension(:,:), allocatable, public b_v_c
  Coeff. of magnitude of B in sine series (HM and FM)
- real(dp), dimension(:,:), allocatable, public b_v_s
  Coeff. of magnitude of B in cosine series (HM and FM)
- real(dp), dimension(:,:), allocatable, public j_v_sup_int
  Integrated poloidal and toroidal current (FM)

B.45.1 Detailed Description

Variables that concern the output of VMEC.

B.45.2 Function/Subroutine Documentation

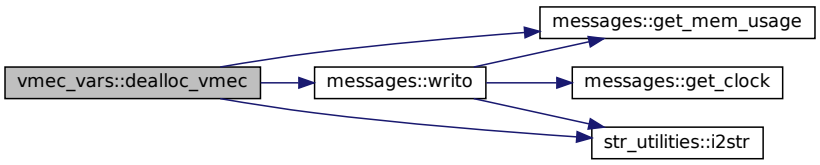
B.45.2.1 dealloc_vmec()

`subroutine, public vmec_vars::dealloc_vmec`

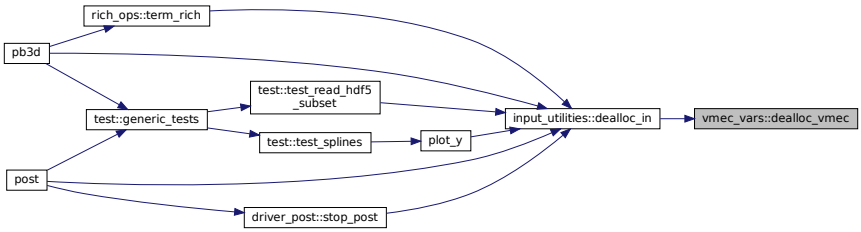
Deallocates VMEC quantities that are not used anymore.

Definition at line 57 of file VMEC_vars.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.45.3 Variable Documentation

B.45.3.1 b_0_v

`real(dp), public vmec_vars::b_0_v`

the magnitude of B at the magnetic axis, $\theta = \zeta = 0$

Definition at line 33 of file VMEC_vars.f90.

B.45.3.2 b_v_c

```
real(dp), dimension(:,:), allocatable, public vmec_vars::b_v_c
```

Coeff. of magnitude of B in sine series (HM and FM)

Note

Debug version only

Definition at line 50 of file VMEC_vars.f90.

B.45.3.3 b_v_s

```
real(dp), dimension(:,:), allocatable, public vmec_vars::b_v_s
```

Coeff. of magnitude of B in cosine series (HM and FM)

Note

Debug version only

Definition at line 51 of file VMEC_vars.f90.

B.45.3.4 b_v_sub_c

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::b_v_sub_c
```

Coeff. of B_i in sine series (r,theta,phi) (FM)

Note

Debug version only

Definition at line 47 of file VMEC_vars.f90.

B.45.3.5 b_v_sub_s

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::b_v_sub_s
```

Coeff. of B_i in cosine series (r,theta,phi) (FM)

Note

Debug version only

Definition at line 48 of file VMEC_vars.f90.

B.45.3.6 flux_p_v

```
real(dp), dimension(:,:), allocatable, public vmec_vars::flux_p_v
```

poloidal flux

Definition at line 35 of file VMEC_vars.f90.

B.45.3.7 flux_t_v

```
real(dp), dimension(:,:), allocatable, public vmec_vars::flux_t_v
```

toroidal flux

Definition at line 34 of file VMEC_vars.f90.

B.45.3.8 j_v_sup_int

```
real(dp), dimension(:,:), allocatable, public vmec_vars::j_v_sup_int
```

Integrated poloidal and toroidal current (FM)

Note

Debug version only

Definition at line 52 of file VMEC_vars.f90.

B.45.3.9 jac_v_c

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::jac_v_c
```

Coeff. of $\mathcal{J}$ in sine series (HM and FM) and norm. deriv.

Definition at line 45 of file VMEC_vars.f90.

B.45.3.10 jac_v_s

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::jac_v_s
```

Coeff. of $\mathcal{J}$ in cosine series (HM and FM) and norm. deriv.

Definition at line 46 of file VMEC_vars.f90.

B.45.3.11 l_v_c

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::l_v_c
```

Coeff. of λ in sine series (HM) and norm. deriv.

Definition at line 43 of file VMEC_vars.f90.

B.45.3.12 l_v_s

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::l_v_s
```

Coeff. of λ in cosine series (HM) and norm. deriv.

Definition at line 44 of file VMEC_vars.f90.

B.45.3.13 mn_v

```
integer, dimension(:,:), allocatable, public vmec_vars::mn_v
```

m and n of modes

Definition at line 32 of file VMEC_vars.f90.

B.45.3.14 pres_v

```
real(dp), dimension(:,:), allocatable, public vmec_vars::pres_v
```

pressure

Definition at line 36 of file VMEC_vars.f90.

B.45.3.15 q_saf_v

```
real(dp), dimension(:,:), allocatable, public vmec_vars::q_saf_v
```

safety factor

Definition at line 38 of file VMEC_vars.f90.

B.45.3.16 r_v_c

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::r_v_c
```

Coeff. of R in sine series (FM) and norm. deriv.

Definition at line 39 of file VMEC_vars.f90.

B.45.3.17 r_v_s

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::r_v_s
```

Coeff. of R in cosine series (FM) and norm. deriv.

Definition at line 40 of file VMEC_vars.f90.

B.45.3.18 rot_t_v

```
real(dp), dimension(:,:), allocatable, public vmec_vars::rot_t_v
```

rotational transform

Definition at line 37 of file VMEC_vars.f90.

B.45.3.19 z_v_c

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::z_v_c
```

Coeff. of Z in sine series (FM) and norm. deriv.

Definition at line 41 of file VMEC_vars.f90.

B.45.3.20 z_v_s

```
real(dp), dimension(:,:,:), allocatable, public vmec_vars::z_v_s
```

Coeff. of Z in cosine series (FM) and norm. deriv.

Definition at line 42 of file VMEC_vars.f90.

B.46 x_ops Module Reference

Operations considering perturbation quantities.

Interfaces and Types

- interface calc_x
  Calculates either vectorial or tensorial perturbation variables.
- interface print_output_x
  Print either vectorial or tensorial perturbation quantities of a certain order to an output file.
- interface redistribute_output_x
  Redistribute the perturbation variables.

Functions/Subroutines

- integer function, public init_modes (grid_eq, eq)
  Initializes some variables concerning the mode numbers.
- integer function, public setup_modes (mds, grid_eq, grid, plot_name)
  Sets up some variables concerning the mode numbers.
- integer function, public check_x_modes (grid_eq, eq)
  Checks whether the high-n approximation is valid:
- integer function, public calc_res_surf (mds, grid_eq, eq, res_surf, info, jq)
  Calculates resonating flux surfaces for the perturbation modes.
- integer function, public resonance_plot (mds, grid_eq, eq)
  plot q-profile or ι-profile in flux coordinates with $nq - m = 0$ or $n - \iota m = 0$ indicated if requested.
- integer function calc_u (grid_eq, grid_X, eq_1, eq_2, X)
  Calculate U_m^0, U_m^1 or U_n^0, U_n^1.

- integer function calc_pv (grid_eq, grid_X, eq_1, eq_2, X_a, X_b, X, lim_sec_X)

  calculate $\widetilde{PV}^i_{k,m}$ (pol. flux) or $\widetilde{PV}^i_{l,n}$ (tor. flux) at all equilibrium `loc_n_r` values.
- integer function calc_kv (grid_eq, grid_X, eq_1, eq_2, X_a, X_b, X, lim_sec_X)

  calculate $\widetilde{KV}^i_{k,m}$ (pol. flux) or $\widetilde{KV}^i_{l,n}$ (tor. flux) at all equilibrium `loc_n_r` values.
- integer function, public calc_magn_ints (grid_eq, grid_X, eq, X, X_int, prev_style, lim_sec_X)

  Calculate the magnetic integrals from $\widetilde{PV}^i_{k,m}$ and $\widetilde{KV}^i_{k,m}$ in an equidistant grid where the step size can vary depending on the normal coordinate.
- integer function, public divide_x_jobs (arr_size)

  Divides the perturbation jobs.
- integer function, public print_debug_x_1 (mds, grid_X, X_1)

  Prints debug information for X_1 driver.
- integer function, public print_debug_x_2 (mds, grid_X, X_2_int)

  Prints debug information for X_2 driver.

Variables

- logical, public debug_check_x_modes_2 = .false.

  plot debug information for check_x_modes_2()

B.46.1 Detailed Description

Operations considering perturbation quantities.

B.46.2 Function/Subroutine Documentation

B.46.2.1 calc_kv()

```
integer function x_ops::calc_kv (
                type(grid_type), intent(in) grid_eq,
                type(grid_type), intent(in) grid_X,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(in), target eq_2,
                type(x_1_type), intent(in) X_a,
                type(x_1_type), intent(in) X_b,
                type(x_2_type), intent(inout) X,
                integer, dimension(2,2), intent(in), optional lim_sec_X )
```

calculate $\widetilde{KV}^i_{k,m}$ (pol. flux) or $\widetilde{KV}^i_{l,n}$ (tor. flux) at all equilibrium `loc_n_r` values.

Like in calc_U(), use is made of variables `Ti` that are set up in the equilibrium grid and are afterwards converted to `Ti_X` in the perturbation grid, which needs to have the same angular coordinates as the equilibrium grid:

$$T_1 = \rho \mathcal{J}^2 \frac{h^{22}}{\mu_0} g_{33}$$

$$T_2 = \rho \frac{1}{h^{22}} ,$$

with T_i = Ti.

The interpolated Ti_X are then used to calculate $\widetilde{KV}^i_{k,m}$:

$$\begin{aligned}\widetilde{KV}^0_{k,m} &= T_1 U^{0*}_k U^0_m + T_2 \\ \widetilde{KV}^1_{k,m} &= T_1 U^{0*}_k U^1_m \\ \widetilde{KV}^2_{k,m} &= T_1 U^{1*}_k U^1_m \,,\end{aligned}$$

where

$$DU^i_k = i(nq - m)U^i_k + \frac{\partial U^i_k}{\partial \theta}.$$

This is valid for poloidal Flux coordinates and where n is to be replaced by m and $(nq - m)$ by $(n - \iota m)$ for toroidal Flux coordinates.

See also

See [16] .

Returns

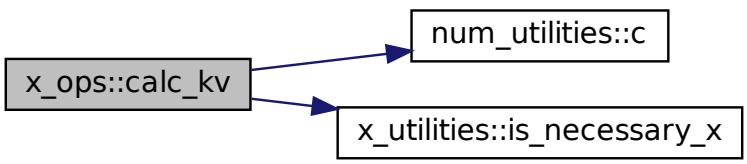
ierr

Parameters

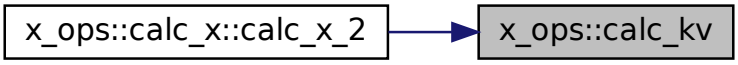
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x_a*	vectorial perturbation variables of dimension 2
in	*x_b*	vectorial perturbation variables of dimension 2
in,out	*x*	tensorial perturbation variables
in	*lim_sec_x*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions

Definition at line 2879 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.2 calc_magn_ints()

```
integer function, public x_ops::calc_magn_ints (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(eq_2_type), intent(in), target eq,
               type(x_2_type), intent(in) X,
               type(x_2_type), intent(inout) X_int,
               integer, intent(in), optional prev_style,
               integer, dimension(2,2), intent(in), optional lim_sec_X )
```

Calculate the magnetic integrals from $\widetilde{PV}^i_{k,m}$ and $\widetilde{KV}^i_{k,m}$ in an equidistant grid where the step size can vary depending on the normal coordinate.

All the variables should thus be field-line oriented. The result is saved in the first index of the X variables, the other can be ignored.

Using `prev_style`, the results of this calculation on X can be combined with the ones already present in `X_int`.

- `prev_style=0`: Overwrite [def].
- `prev_style=1`: Add to current integral.
- `prev_style=2`: Divide by 2 and add to current integral. Also modify the indices of the current integral:
  - for `magn_int_style` = 1: `1 2 2 2 2 2 1 -> 2 2 2 2 2 2`,
  - for `magn_int_style` = 2: `1 3 3 2 3 3 1 -> 3 2 3 3 2 3`.
- `prev_style=3`: Add to current integral. Also modify the indices of the current integral as in `prev_style` = 2.
- else: ignore previous magnetic integrals.

Therefore, it is to be used in order. For example:

1. (R=1,E=1): style 0,
2. $\rightarrow$ (R=1,E=2): style 1,
3. $\rightarrow$ (R=2,E=1): style 2,
4. $\rightarrow$ (R=2,E=2): style 3.

Afterwards, it would cycle through 2 and 3, as style 0 and 1 are only for the first Richardson level.

Note

1. The variable type for the integrated tensorial perturbation variables is also `X_2`, but it is assumed that the first index has dimension 1, the rest is ignored.
2. Simpson's 3/8 rule converges faster than the trapezoidal rule, but normally needs a better starting point (i.e. higher `min_n_par_X`)
3. For alpha style 1, Weyl's lemma is used to convert the surface integral into a line integral along the magnetic field. This means that the resulting line integral still needs to be multiplied by $\frac{2\pi}{M}$, where M is the number of times the poloidal or toroidal angle completes a full 2π tour, depending on `use_pol_flux_F` [6].
4. For alpha style 2, this factor is just the step size in alpha.

See also

See x_vars.x_2_type.

Returns

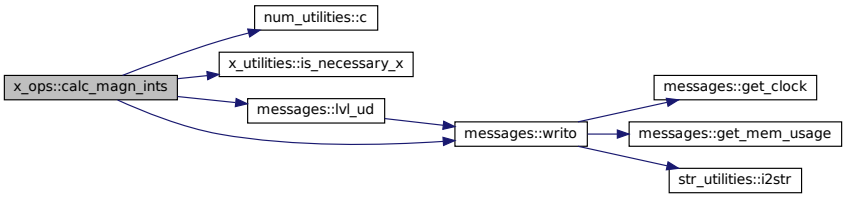
ierr

Parameters

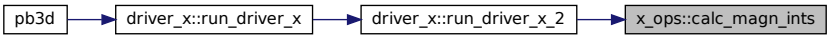
`in`	*grid_eq*	equilibrium grid
`in`	*grid_x*	perturbation grid
`in`	*eq*	metric equilibrium
`in`	*x*	tensorial perturbation variables
`in,out`	*x_int*	interpolated tensorial perturbation variables, containing all mode combinations
`in`	*prev_style*	style to treat X_prev
`in`	*lim_sec↩ _x*	limits of `m_X` (pol flux) or `n_X` (tor flux) for both dimensions

Definition at line 3081 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.3 calc_pv()

```
integer function x_ops::calc_pv (
                type(grid_type), intent(in) grid_eq,
                type(grid_type), intent(in) grid_X,
                type(eq_1_type), intent(in), target eq_1,
                type(eq_2_type), intent(in), target eq_2,
                type(x_1_type), intent(in) X_a,
                type(x_1_type), intent(in) X_b,
                type(x_2_type), intent(inout) X,
                integer, dimension(2,2), intent(in), optional lim_sec_X )
```

calculate $\widetilde{PV}^i_{k,m}$ (pol. flux) or $\widetilde{PV}^i_{l,n}$ (tor. flux) at all equilibrium `loc_n_r` values.

Like in calc_U(), use is made of variables `Ti` that are set up in the equilibrium grid and are afterwards converted to `Ti_X` in the perturbation grid, which needs to have the same angular coordinates as the equilibrium grid:

$$
\begin{aligned}
T_1 &= \frac{h^{22}}{\mu_0} g_{33} \\
T_2 &= \mathcal{J} S + \mu_0 \sigma \frac{g_{33}}{\mathcal{J}} h^{22} \\
T_3 &= \frac{\sigma}{\mathcal{J}} T_2 \\
T_4 &= \frac{1}{\mu_0} \mathcal{J}^2 h^{22} \\
T_5 &= 2p' \kappa_n \, ,
\end{aligned}
$$

with T_i = `Ti`.

The interpolated Ti_X are then used to calculate $\widetilde{PV}^i_{k,m}$:

$$
\begin{aligned}
\widetilde{PV}^0_{k,m} &= T_1 (DU^{0*}_k - T_2)(DU^0_m - T_2) - T_3 + (nq - m)(nq - k) T_4 - T_5 \\
\widetilde{PV}^1_{k,m} &= T_1 (DU^{0*}_k - T_2) DU^1_m \\
\widetilde{PV}^2_{k,m} &= T_1 DU^{1*}_k DU^1_m \, ,
\end{aligned}
$$

where

$$
DU^i_k = i(nq - m) U^i_k + \frac{\partial U^i_k}{\partial \theta} .
$$

This is valid for poloidal Flux coordinates and where n is to be replaced by m and $(nq - m)$ by $(n - \iota m)$ for toroidal Flux coordinates.

See also

See [16] .

Returns

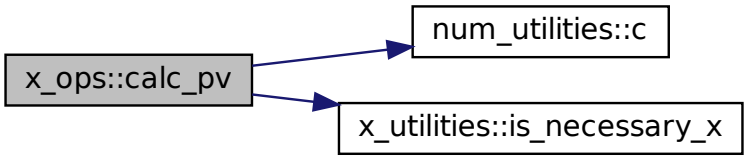
ierr

Parameters

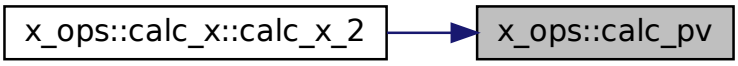
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x_a*	vectorial perturbation variables of dimension 2
in	*x_b*	vectorial perturbation variables of dimension 2
in,out	*x*	tensorial perturbation variables
in	*lim_↩ sec_x*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions

Definition at line 2646 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.4 calc_res_surf()

```
integer function, public x_ops::calc_res_surf (
                type(modes_type), intent(in) mds,
                type(grid_type), intent(in) grid_eq,
                type(eq_1_type), intent(in) eq,
                real(dp), dimension(:,:), intent(inout), allocatable res_surf,
```

```
logical, intent(in), optional info,
real(dp), dimension(:), intent(inout), optional, allocatable jq )
```

Calculates resonating flux surfaces for the perturbation modes.

The output consists of mode number, resonating normal position and the fraction $\frac{m}{n}$ or $\frac{n}{m}$. for those modes for which a solution is found that is within the plasma range.

It contains three pieces of information:

- `(:,1)`: the mode index
- `(:,2)`: the radial position in Flux coordinates
- `(:,3)`: the fraction $\frac{m}{n}$ or $\frac{n}{m}$

for every single mode in sec of `mds`, which can be tabulated in an arbitrary grid, not necessarily the equilibrium one.

Optionally, the total safety factor or rotational transform can be returned to the master.

Also, information can be displayed with \info.

Returns

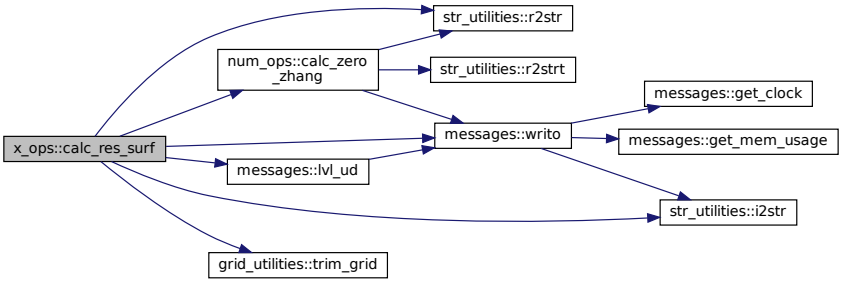
ierr

Parameters

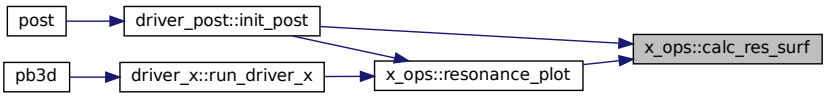
`in`	*mds*	general modes variables
`in`	*grid_eq*	equilibrium grid_eq
`in`	*eq*	flux equilibrium
`in,out`	*res_surf*	resonant surface
`in`	*info*	if info is displayed
`in,out`	*jq*	either safety factor or rotational transform

Definition at line 1638 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.5 calc_u()

```
integer function x_ops::calc_u (
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(eq_1_type), intent(in), target eq_1,
               type(eq_2_type), intent(in), target eq_2,
               type(x_1_type), intent(inout) X )
```

Calculate U_m^0, U_m^1 or U_n^0, U_n^1.

This is done at eq `loc_n_r` values of the normal coordinate, `n_par` values of the parallel coordinate and `size_X` values of the poloidal mode number, or of the toroidal mode number, as well as the poloidal derivatives

Use is made of variables `Ti` that are set up in the equilibrium grid and are afterwards converted to `Ti_X` in the perturbation grid, which needs to have the same angular coordinates as the equilibrium grid:

$$
\begin{aligned}
T_1 &= \frac{B_\alpha}{B_\theta} \\
T_2 &= \Theta^\alpha + q'\theta \\
T_3 &= \frac{B_\alpha}{B_\theta} q' + \frac{\mathcal{J}^2}{B_\theta} \mu_0 p' \\
T_4 &= \frac{B_\alpha}{B_\theta} q'\theta - \frac{B_\psi}{B_\theta} \\
T_5 &= \frac{B_\alpha}{B_\theta} \frac{\partial \Theta^\theta}{\partial \theta} - \frac{\partial \Theta^\theta}{\partial \alpha} \\
T_6 &= \frac{B_\alpha}{B_\theta} \Theta^\theta \,,
\end{aligned}
$$

with T_i = `Ti`.

which is valid for poloidal Flux coordinates.

For toroidal Flux coordinates, q' has to be replaced by $-\iota'$ and θ by ζ.

The interpolated `Ti_X` are then used to calculate U:

$$\begin{aligned} U_0 = &- T_2 && \text{(order 1)} \\ &+ \frac{i}{n}\left(T_3 + i(nq-m)T_4\right) && \text{(order 2)} \\ &+ \left(\frac{i}{n}\right)^2 i(nq-m)\left(-T_5 - (nq-m)T_6\right) && \text{(order 3)} \\ U_1 = &\frac{i}{n} && \text{(order 1)} \\ &+ \left(\frac{i}{n}\right)^2 i(nq-m)(-T_1) && \text{(order 2).} \end{aligned}$$

This is valid for poloidal Flux coordinates and where n is to be replaced by m and $(nq-m)$ by $(n-\iota m)$ for toroidal Flux coordinates.

For VMEC, these factors are also derived in the parallel coordinate.

See also

See [16].

Returns

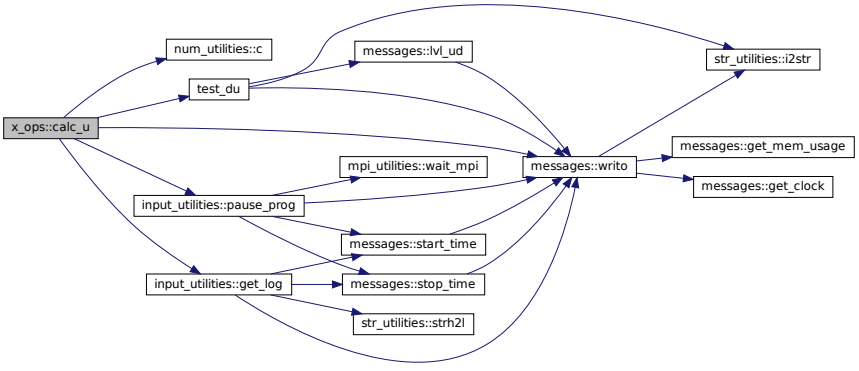
ierr

Parameters

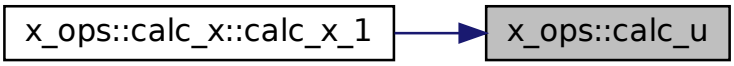
`in`	*grid_eq*	equilibrium grid variables
`in`	*grid_x*	perturbation grid variables
`in`	*eq_1*	flux equilibrium
`in`	*eq_2*	metric equilibrium
`in,out`	*x*	vectorial perturbation variables

Definition at line 2097 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.6 check_x_modes()

```
integer function, public x_ops::check_x_modes (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq )
```

Checks whether the high-n approximation is valid:

This depends on the `X_style`:

For X_style 1 (prescribed): every mode should resonate at least somewhere in the whole normal range:

- $\frac{|nq-m|}{|n|} < T$ and $\frac{|nq-m|}{|m|} < T$ for poloidal flux
- $\frac{|q-\iota m|}{|m|} < T$ and $\frac{|n-\iota m|}{|n|} < T$ for toroidal flux

where T = `tol` $<<$ 1.

This condition is determined by the sign of q (or ι) and given by:

- $\max\left(q_{\text{min}} - T, \frac{q_{\text{min}}}{1+T}\right) < \frac{m}{n} < \min\left(q_{\text{max}} + T, \frac{q_{\text{max}}}{1-T}\right), q > 0$
- $\max\left(q_{\text{min}} - T, \frac{q_{\text{min}}}{1-T}\right) < \frac{m}{n} < \min\left(q_{\text{max}} + T, \frac{q_{\text{max}}}{1+T}\right), q < 0$

for poloidal flux.

For toroidal flux, q should be replaced by ι and $\frac{m}{n}$ by $\frac{n}{m}$.

For `X_style` 2 (fast): the resonance has been taken care of, but it remains to be checked whether the number of modes is efficient.

Returns

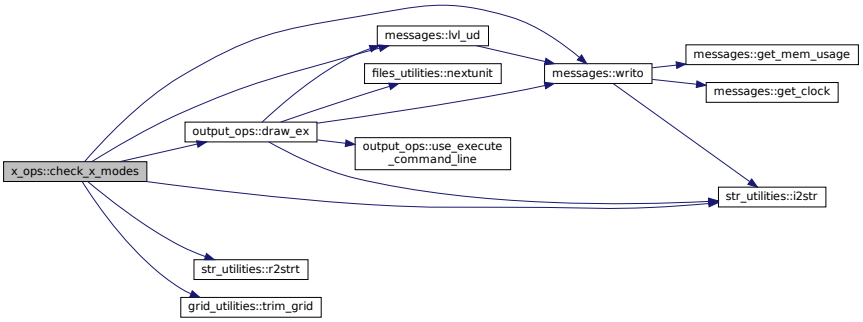
ierr

Parameters

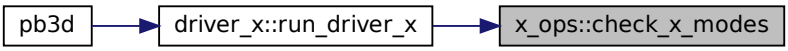
in	*grid_eq*	equilibrium grid
in	*eq*	flux equilibrium

Definition at line 1350 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.7 divide_x_jobs()

```
integer function, public x_ops::divide_x_jobs (
              integer, intent(in) arr_size )
```

Divides the perturbation jobs.

This concerns the calculation of the magnetic integrals of blocks of tensorial perturbation variables. These are set up using the equilibrium and vectorial perturbation variables that are stored in memory, but only the integrated tensorial result is stored in memory, with negligible variables size.

The size of the (k,m) pairs to be calculated is determined by looking at what fits in memory when the equilibrium and vectorial perturbation variables are stored in memory first.

Returns

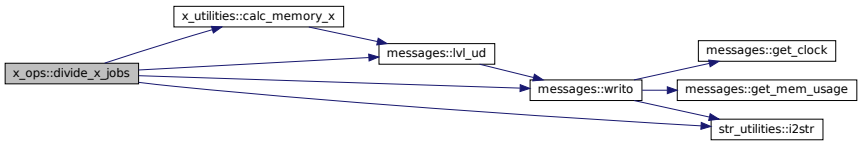
ierr

Parameters

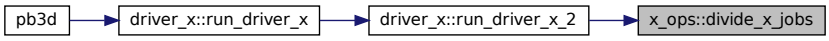
in	*arr_size*	array size (using loc_n_r)

Definition at line 3364 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.8 init_modes()

```
integer function, public x_ops::init_modes (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq )
```

Initializes some variables concerning the mode numbers.

Setup minimum and maximum of mode numbers at every flux surface in equilibrium coordinates

- `min_n_X`, `max_n_X`
- `min_m_X`, `max_m_X`,

It functions depending on the `X_style` used: 1 (prescribed) or 2 (fast). For the fast style, at every flux surface the range of modes is sought that resonates most:

- $m = nq \pm \frac{N}{2}$ for poloidal flux
- $n = \iota m \pm \frac{N}{2}$ for toroidal flux

where N = `n_mod_X`.

However, at the same time, both `m` and `n` have to be larger, in absolute value, than `min_sec_X`. Therefore, the range of width `n_mod_X` can be shifted upwards.

Returns

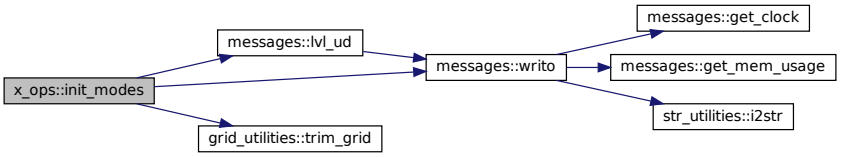
ierr

Parameters

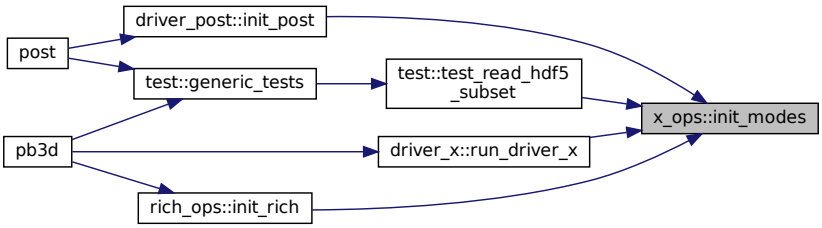
in	*grid_eq*	equilibrium grid
in	*eq*	flux equilibrium

Definition at line 919 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.9 print_debug_x_1()

```
integer function, public x_ops::print_debug_x_1 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               type(x_1_type), intent(in) X_1 )
```

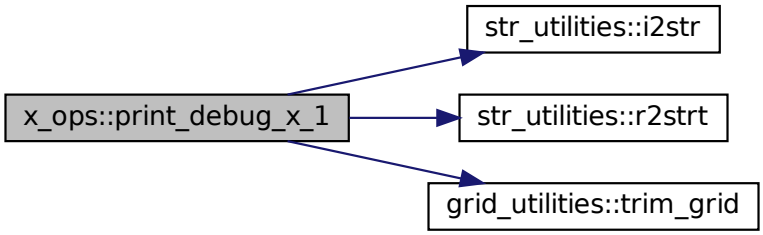
Prints debug information for X_1 driver.

Parameters

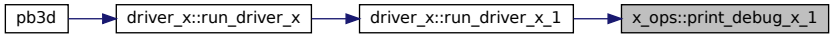
in	*mds*	general modes variables
in	*grid_X*	perturbation grid
in	*x_1*	vectorial X variables

Definition at line 3490 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.10 print_debug_x_2()

```
integer function, public x_ops::print_debug_x_2 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               type(x_2_type), intent(in) X_2_int )
```

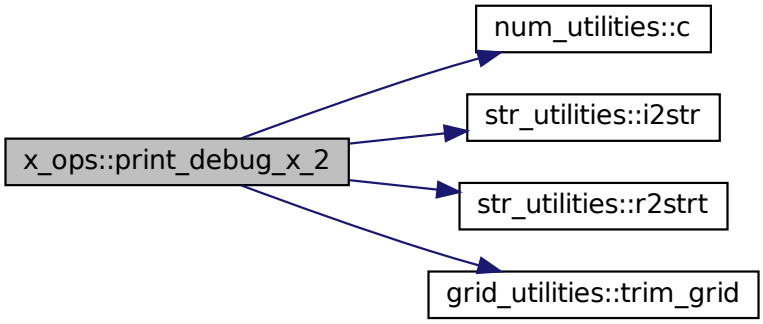
Prints debug information for X_2 driver.

Parameters

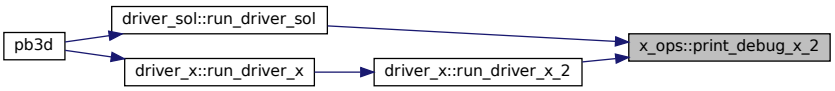
in	*mds*	general modes variables
in	*grid_x*	perturbation grid
in	*x_2_int*	tensorial X variables

Definition at line 3633 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.11 resonance_plot()

```
integer function, public x_ops::resonance_plot (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq )
```

plot q-profile or ι-profile in flux coordinates with $nq - m = 0$ or $n - \iota m = 0$ indicated if requested.

The plot will be done in the grid in which `mds` is tabulated, which is not necessarily the equilibrium one.

Returns

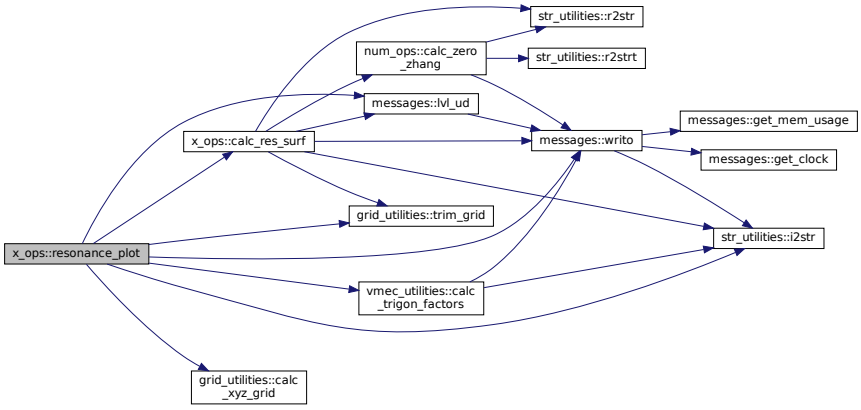
ierr

Parameters

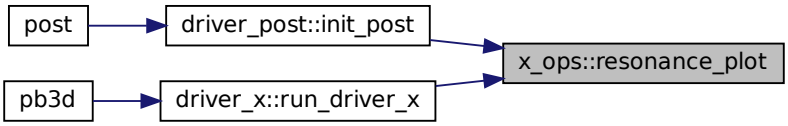
in	*mds*	general modes variables
in	*grid_eq*	equilibrium grid
in	*eq*	flux equilibrium

Definition at line 1814 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.2.12 setup_modes()

```
integer function, public x_ops::setup_modes (
               type(modes_type), intent(inout), target mds,
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid,
               character(len=*), intent(in), optional plot_name )
```

Sets up some variables concerning the mode numbers.

Apart from the mode numbers

- n, m,

also the variables of the secondary modes

- sec,

are set up in the coordinates of the grid passed. The normal values of this grid are saved as well, in

- r_F.

All variables are tabulated in the full normal grid. The mode indices, however, are chosen so that there is maximum overlap between different normal positions. This is trivial for `X_style` 1 (prescribed), but for `X_style` 2 (fast), this is best explained by an example:

B.46.2.13 kd m1 m2 m3

1 10 11 12 <- start 2 10 11 12 <- no change in limits 2 13 11 12 <- limits shift up by one 3 13 11 12 <- no change in limits 4 13 14 12 <- limits shift up by one 5 16 14 15 <- limits shift up by two 6 13 14 15 <- limits shift down by one

This procedure makes use of the global variables

- `min_m_X`, max_m_X, min_n_X, max_m_X

that have to be set up using init_nm_x().

Optionally, `n` and `m` can be plot.

Returns

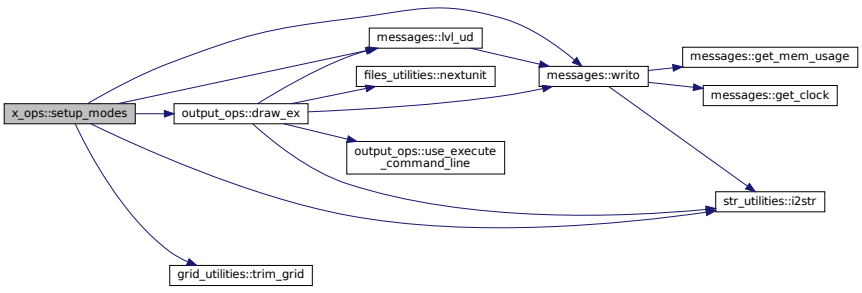
ierr

Parameters

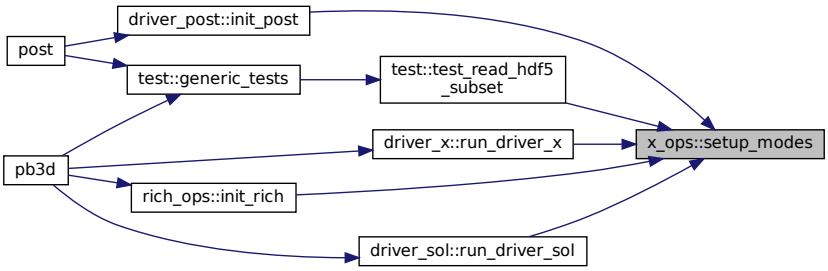
`in,out`	*mds*	modes variables
`in`	*grid_eq*	equilibrium grid
`in`	*grid*	grid at which to calculate modes
`in`	*plot_name*	name to be used when plotting `n` and `m`

Definition at line 1060 of file X_ops.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.46.3 Variable Documentation

B.46.3.1 debug_check_x_modes_2

```
logical, public x_ops::debug_check_x_modes_2 = .false.
```

plot debug information for check_x_modes_2()

Note

Debug version only

Definition at line 26 of file X_ops.f90.

B.47 x_utilities Module Reference

Numerical utilities related to perturbation operations.

Interfaces and Types

- interface sec_ind_loc2tot

  Returns the `sec_ind_tot` used to refer to a perturbation quantity.

Functions/Subroutines

- subroutine, public get_sec_x_range (sec_X_range_loc, sec_X_range_tot, m, sym, lim_sec_X)

  Gets one of the the local ranges of contiguous tensorial perturbation variables to be printed or read during one call of the corresponding HDF5 variables.
- logical function, public do_x ()

  Tests whether this perturbatino job should be done.
- logical function, public is_necessary_x (sym, sec_X_id, lim_sec_X)

  Determines whether a variable needs to be considered:
- integer function, public calc_memory_x (ord, arr_size, n_mod, mem_size)

  Calculate memory in MB necessary for X variables.
- integer function, public trim_modes (mds_i, mds_o, id_lim_i, id_lim_o)

  Limit input mode range to output mode range.

B.47.1 Detailed Description

Numerical utilities related to perturbation operations.

B.47.2 Function/Subroutine Documentation

B.47.2.1 calc_memory_x()

```
integer function, public x_utilities::calc_memory_x (
               integer, intent(in) ord,
               integer, intent(in) arr_size,
               integer, intent(in) n_mod,
               real(dp), intent(inout) mem_size )
```

Calculate memory in MB necessary for X variables.

This depends on the order:

- order 1:
  - `4x n_par_X x n_geo x loc_n_r x n_mod`
- order 2:
  - `2x n_par_X x n_geo x loc_n_r x n_mod^2`
  - `4x n_par_X x n_geo x loc_n_r x n_mod(n_mod+1)/2`
- higher order: not used

where `n_par_X x n_geo x loc_n_r` should be passed as `arr_size` and `n_mod` as well.

Returns

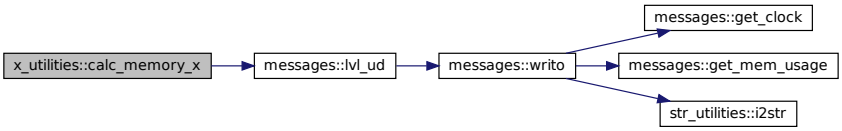
ierr

Parameters

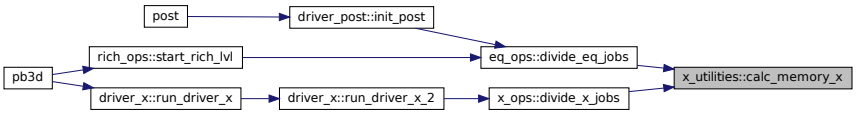
in	*ord*	order of data
in	*arr_size*	size of part of X array
in	*n_mod*	number of modes
in,out	*mem_size*	total size

Definition at line 236 of file X_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.47.2.2 do_x()

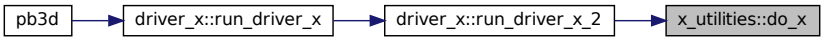
`logical function, public x_utilities::do_x`

Tests whether this perturbatino job should be done.

Also increments `X_job_nr`

Definition at line 179 of file X_utilities.f90.

Here is the caller graph for this function:



B.47.2.3 get_sec_x_range()

```
subroutine, public x_utilities::get_sec_x_range (
               integer, dimension(2), intent(inout) sec_X_range_loc,
               integer, dimension(2), intent(inout) sec_X_range_tot,
               integer, intent(in) m,
               logical, intent(in) sym,
               integer, dimension(2,2), intent(in), optional lim_sec_X )
```

Gets one of the the local ranges of contiguous tensorial perturbation variables to be printed or read during one call of the corresponding HDF5 variables.

More specifically, a range of indices k in the first dimension is given for every value of the indices m in the second dimension.

An example is now given for the subrange [2:3,2:5] of a the total range [1:5, 1:5]. For asymmetric variables the situation is simple: The k range is [2:3] for all 5 values of m. However, for symmetric variables, the upper diagonal values are not stored, which gives k ranges [2:3], [3:3] and no range for m = 4 and 5.

This routine then translates these ranges to the corresponding 1-D ranges that are used in the actual variables. For above example, the total indices are

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 3 & 8 & 13 & 18 & 23 \\ 4 & 9 & 14 & 19 & 24 \\ 5 & 10 & 15 & 20 & 25 \end{pmatrix} \rightarrow [7:8], [12:13], [17:18] \text{ and } [22:23],$$

for asymmetric variables and

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 6 & 7 & 8 & 9 \\ 3 & 7 & 10 & 11 & 12 \\ 4 & 8 & 11 & 13 & 14 \\ 5 & 9 & 12 & 14 & 15 \end{pmatrix} \rightarrow [6:7], [10:10], [:] \text{ and } [:],$$

for symmetric variables.

These can then related to the local indices for the variables in this perturbation job.

For above example, the results are:

[1:2], [3:4], [5:6] and [7:8],

for asymmetric variables and

[1:2], [3:3], [:] and [:],

for symmetric variables.

As can be seen, the local ranges of the variables in the submatrix of this perturbation job are (designed to be) contiguous, but the total ranges of the variables in the submatrix are clearly not in general.

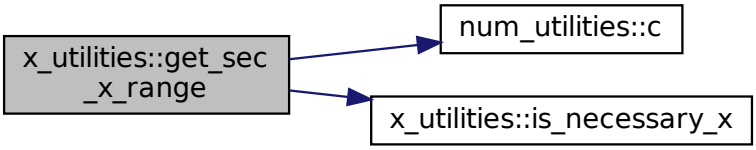
The procedure outputs both the local and total ranges.

Parameters

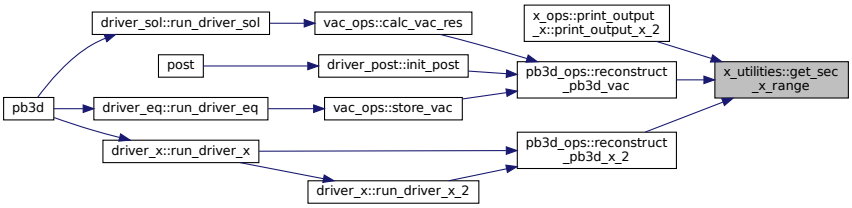
in,out	*sec_x_range_loc*	start and end of local range in dimension 1 (vertical)
in,out	*sec_x_range_tot*	start and end of total range in dimension 1 (vertical)
in	*m*	dimension 2 (horizontal)
in	*sym*	whether the variable is symmetric
in	*lim_sec_x*	limits of m_X (pol. flux) or n_X (tor. flux)

Definition at line 128 of file X_utilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



B.47.2.4 is_necessary_x()

```
logical function, public x_utilities::is_necessary_x (
               logical, intent(in) sym,
               integer, dimension(2), intent(in) sec_X_id,
               integer, dimension(2,2), intent(in), optional lim_sec_X )
```

Determines whether a variable needs to be considered:

This depends on whether the quantity is symmetric or not:

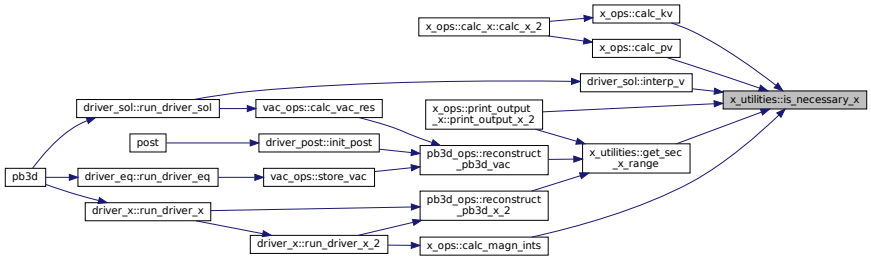
- Only if it is on or below the diagonal for symmetric quantities.
- Always for asymmetric quantities

Parameters

in	*sym*	whether the variable is symmetric
in	*sec_x_id*	mode indices
in	*lim_↩ sec_x*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions

Definition at line 197 of file X_utilities.f90.

Here is the caller graph for this function:



B.47.2.5 trim_modes()

```
integer function, public x_utilities::trim_modes (
               type(modes_type), intent(in) mds_i,
               type(modes_type), intent(in) mds_o,
               integer, dimension(2), intent(inout) id_lim_i,
               integer, dimension(2), intent(inout) id_lim_o )
```

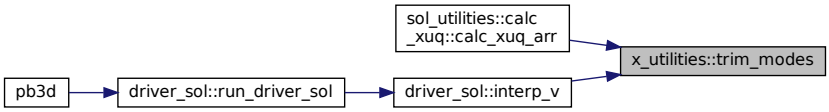
Limit input mode range to output mode range.

Parameters

in	*mds_i*	general modes variables for input
in	*mds_o*	general modes variables for output
in,out	*id_↩ lim_i*	limits on input modes
in,out	*id_↩ lim_o*	limits on output modes

Definition at line 293 of file X_utilities.f90.

Here is the caller graph for this function:



B.48 x_vars Module Reference

Variables pertaining to the perturbation quantities.

Interfaces and Types

- type modes_type
  mode number type
- interface set_nm_x
  Sets n_X and m_X.
- type x_1_type
  vectorial perturbation type
- type x_2_type
  tensorial perturbation type

Functions/Subroutines

- subroutine init_x_1 (X, mds, grid_X, lim_sec_X)
  Initializes a vectorial perturbation.
- subroutine init_x_2 (X, mds, grid_X, lim_sec_X, is_field_averaged)
  Initializes a tensorial perturbation.
- subroutine copy_x_1 (X_i, mds, grid_i, X_o)
  Deep copy of vectorial perturbation variables.
- subroutine copy_x_2 (X_i, mds, grid_i, X_o)
  Deep copy of tensorial perturbation variables.
- integer function, public set_nn_mod (sym, lim_sec_X)
  Sets number of entries for tensorial perturbation variables.
- subroutine dealloc_mds (mds)
  Deallocates modes variables.
- subroutine dealloc_x_1 (X)
  Deallocates vectorial perturbation variables.
- subroutine dealloc_x_2 (X)
  Deallocates tensorial perturbation variables.

Variables

- type(modes_type), public mds_x
  modes variables for perturbation grid
- type(modes_type), public mds_sol
  modes variables for solution grid
- integer, public prim_x
  n_X (pol. flux) or m_X (tor. flux)
- integer, public min_sec_x
  m_X (pol. flux) or n_X (tor. flux) (only for X style 1)
- integer, public max_sec_x
  m_X (pol. flux) or n_X (tor. flux) (only for\ c X style 1)
- integer, public n_mod_x
  size of m_X (pol. flux) or n_X (tor. flux)
- integer, public min_nm_x = 5
  minimum for the high-n theory (debable)
- integer, dimension(:), allocatable, public min_n_x
  lowest poloidal mode number m_X, in total eq grid
- integer, dimension(:), allocatable, public max_n_x
  highest poloidal mode number m_X, in total eq grid
- integer, dimension(:), allocatable, public min_m_x
  lowest poloidal mode number m_X, in total eq grid
- integer, dimension(:), allocatable, public max_m_x
  highest poloidal mode number m_X, in total eq grid
- real(dp), public min_r_sol
  min. normal range for pert.
- real(dp), public max_r_sol
  max. normal range for pert.
- integer, public n_alloc_x_1s
  nr. of allocated X_1's
- integer, public n_alloc_x_2s
  nr. of allocated X_2's

B.48.1 Detailed Description

Variables pertaining to the perturbation quantities.

B.48.2 Function/Subroutine Documentation

B.48.2.1 copy_x_1()

```
subroutine x_vars::copy_x_1 (
               class(x_1_type), intent(in) X_i,
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_i,
               type(x_1_type), intent(inout) X_o )
```

Deep copy of vectorial perturbation variables.

Parameters

in	*x_i*	X_1 to be copied
in	*mds*	general modes variables
in	*grid_i*	grid of eq_i
in,out	*x_o*	copied X_1

Definition at line 338 of file X_vars.f90.

B.48.2.2 copy_x_2()

```
subroutine x_vars::copy_x_2 (
               class(x_2_type), intent(in) X_i,
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_i,
               type(x_2_type), intent(inout) X_o )
```

Deep copy of tensorial perturbation variables.

Parameters

in	*x_i*	X_2 to be copied
in	*mds*	general modes variables
in	*grid_i*	grid of eq_i
in,out	*x_o*	copied X_1

Definition at line 359 of file X_vars.f90.

B.48.2.3 dealloc_mds()

```
subroutine x_vars::dealloc_mds (
               class(modes_type), intent(inout) mds )
```

Deallocates modes variables.

Parameters

in,out	*mds*	modes variables to be deallocated

Definition at line 412 of file X_vars.f90.

B.48.2.4 dealloc_x_1()

```
subroutine x_vars::dealloc_x_1 (
               class(x_1_type), intent(inout) X )
```

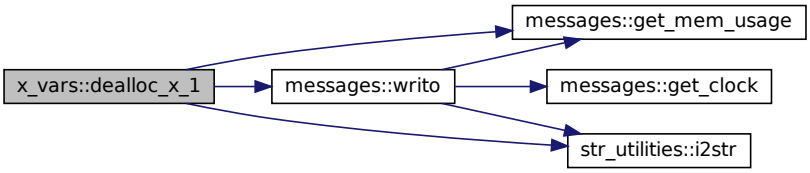
Deallocates vectorial perturbation variables.

Parameters

in,out	X	perturbation variables to be deallocated

Definition at line 428 of file X_vars.f90.

Here is the call graph for this function:



B.48.2.5 dealloc_x_2()

```
subroutine x_vars::dealloc_x_2 (
               class(x_2_type), intent(inout) X )
```

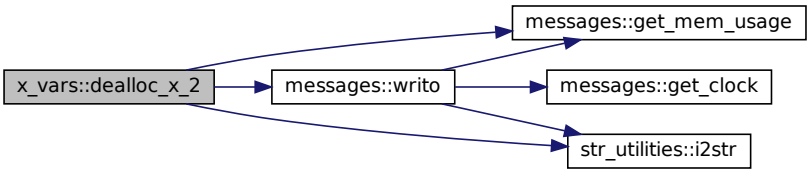
Deallocates tensorial perturbation variables.

Parameters

in,out	X	perturbation variables to be deallocated

Definition at line 474 of file X_vars.f90.

Here is the call graph for this function:



B.48.2.6 init_x_1()

```
subroutine x_vars::init_x_1 (
               class(x_1_type), intent(inout) X,
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               integer, dimension(2), intent(in), optional lim_sec_X )
```

Initializes a vectorial perturbation.

Allocates the variables, the number of modes, as well as `n` and `m`.

Optionally, the secondary mode numbers can be specified (`m` if poloidal flux is used and `n` if toroidal flux). By default, they are taken from the global `X_vars` variables.

Note

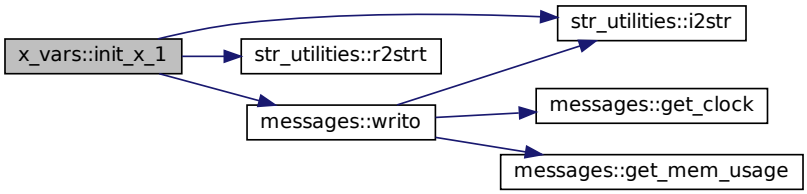
If the lowest limits of the grid is not 1 (e.g. `grid_soli_min = 1` for first process), the input variable `i_min` should be set to set correctly. For a full grid, it should be set to 1.

Parameters

`in,out`	*X*	vectorial perturbation variables
`in`	*mds*	general modes variables
`in`	*grid_x*	perturbation grid
`in`	*lim_↩ sec_x*	limits of `m_X` (pol. flux) or `n_X` (tor. flux)

Definition at line 204 of file X_vars.f90.

Here is the call graph for this function:



B.48.2.7 init_x_2()

```
subroutine x_vars::init_x_2 (
               class(x_2_type), intent(inout) X,
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               integer, dimension(2,2), intent(in), optional lim_sec_X,
               logical, intent(in), optional is_field_averaged )
```

Initializes a tensorial perturbation.

See also

See init_X_1().

Note

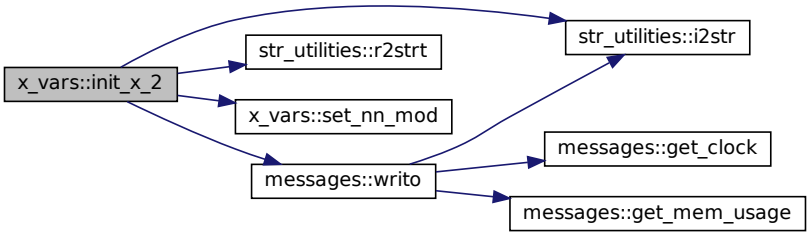
The tensorial perturbation type can also be used for field-aligned variables, in which case the first index is assumed to have dimension 1 only. This can be triggered using `is_field_averaged`. There is no difference between a tensorial perturbation type with size of first dimension set to one through the use of this flag, or through other means.

Parameters

`in,out`	*x*	tensorial perturbation variables
`in`	*mds*	general modes variables
`in`	*grid_x*	perturbation grid
`in`	*lim_sec_x*	limits of `m_X` (pol. flux) or `n_X` (tor. flux) for both dimensions
`in`	*is_field_averaged*	if field-aligned, only one dimension for first index

Definition at line 271 of file X_vars.f90.

Here is the call graph for this function:



B.48.2.8 set_nn_mod()

```
integer function, public x_vars::set_nn_mod (
               logical, intent(in) sym,
               integer, dimension(2,2), intent(in), optional lim_sec_X )
```

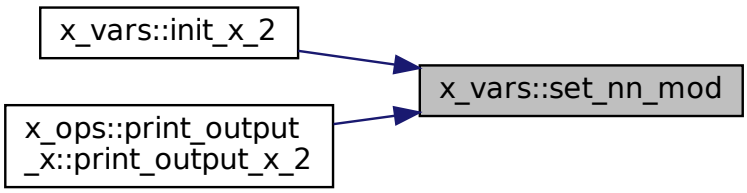
Sets number of entries for tensorial perturbation variables.

Parameters

in	*sym*	whether the variable is symmetric
in	*lim_↩ sec_X*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions

Definition at line 383 of file X_vars.f90.

Here is the caller graph for this function:



B.48.3 Variable Documentation

B.48.3.1 max_m_x

```
integer, dimension(:), allocatable, public x_vars::max_m_x
```

highest poloidal mode number m_X, in total eq grid

Definition at line 134 of file X_vars.f90.

B.48.3.2 max_n_x

```
integer, dimension(:), allocatable, public x_vars::max_n_x
```

highest poloidal mode number m_X, in total eq grid

Definition at line 132 of file X_vars.f90.

B.48.3.3 max_r_sol

```
real(dp), public x_vars::max_r_sol
```

max. normal range for pert.

Definition at line 136 of file X_vars.f90.

B.48.3.4 max_sec_x

```
integer, public x_vars::max_sec_x
```

m_X (pol. flux) or n_X (tor. flux) (only for\ c X style 1)

Definition at line 128 of file X_vars.f90.

B.48.3.5 mds_sol

```
type(modes_type), public x_vars::mds_sol
```

modes variables for solution grid

Definition at line 125 of file X_vars.f90.

B.48.3.6 mds_x

```
type(modes_type), public x_vars::mds_x
```

modes variables for perturbation grid

Definition at line 124 of file X_vars.f90.

B.48.3.7 min_m_x

```
integer, dimension(:), allocatable, public x_vars::min_m_x
```

lowest poloidal mode number `m_X`, in total eq grid

Definition at line 133 of file X_vars.f90.

B.48.3.8 min_n_x

```
integer, dimension(:), allocatable, public x_vars::min_n_x
```

lowest poloidal mode number `m_X`, in total eq grid

Definition at line 131 of file X_vars.f90.

B.48.3.9 min_nm_x

```
integer, public x_vars::min_nm_x = 5
```

minimum for the high-n theory (debable)

Definition at line 130 of file X_vars.f90.

B.48.3.10 min_r_sol

```
real(dp), public x_vars::min_r_sol
```

min. normal range for pert.

Definition at line 135 of file X_vars.f90.

B.48.3.11 min_sec_x

```
integer, public x_vars::min_sec_x
```

m_X (pol. flux) or n_X (tor. flux) (only for X style 1)

Definition at line 127 of file X_vars.f90.

B.48.3.12 n_alloc_x_1s

```
integer, public x_vars::n_alloc_x_1s
```

nr. of allocated X_1's

Note

Debug version only

Definition at line 138 of file X_vars.f90.

B.48.3.13 n_alloc_x_2s

```
integer, public x_vars::n_alloc_x_2s
```

nr. of allocated X_2's

Note

Debug version only

Definition at line 139 of file X_vars.f90.

B.48.3.14 n_mod_x

```
integer, public x_vars::n_mod_x
```

size of m_X (pol. flux) or n_X (tor. flux)

Definition at line 129 of file X_vars.f90.

B.48.3.15 prim_x

```
integer, public x_vars::prim_x
```

n_X (pol. flux) or m_X (tor. flux)

Definition at line 126 of file X_vars.f90.

Appendix C

Interfaces and Types

C.1 num_utilities::add_arr_mult Interface Reference

Add to an array (3) the product of arrays (1) and (2).

Public Member Functions

- integer function add_arr_mult_3_3 (arr_1, arr_2, arr_3, deriv)
  version with `arr_1` and `arr_2` in 3 coords.
- integer function add_arr_mult_3_1 (arr_1, arr_2, arr_3, deriv)
  version with `arr_1` in 3 coords and `arr_2` only in the flux coord.
- integer function add_arr_mult_1_1 (arr_1, arr_2, arr_3, deriv)
  Version with `arr_1` and `arr_2` only in the flux coord.

C.1.1 Detailed Description

Add to an array (3) the product of arrays (1) and (2).

The derivatives are distributed between both acording to the binomial theorem. Both arrays are given on a 3-D or 1-D grid.

Returns

ierr

Definition at line 39 of file num_utilities.f90.

C.1.2 Member Function/Subroutine Documentation

C.1.2.1 add_arr_mult_1_1()

```
integer function num_utilities::add_arr_mult::add_arr_mult_1_1 (
               real(dp), dimension(1:,0:), intent(in) arr_1,
               real(dp), dimension(1:,0:), intent(in) arr_2,
               real(dp), dimension(1:), intent(out) arr_3,
               integer, dimension(3), intent(in) deriv )
```

Version with `arr_1` and `arr_2` only in the flux coord.

Parameters

in	*arr_1*	arr_1
in	*arr_2*	arr_2
out	*arr_3*	arr_3
in	*deriv*	derivatives

Definition at line 492 of file num_utilities.f90.

C.1.2.2 add_arr_mult_3_1()

```
integer function num_utilities::add_arr_mult::add_arr_mult_3_1 (
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) arr_1,
               real(dp), dimension(1:,0:), intent(in) arr_2,
               real(dp), dimension(1:,1:,1:), intent(out) arr_3,
               integer, dimension(3), intent(in) deriv )
```

version with arr_1 in 3 coords and arr_2 only in the flux coord.

Parameters

in	*arr_1*	arr_1
in	*arr_2*	arr_2
out	*arr_3*	arr_3
in	*deriv*	derivatives

Definition at line 441 of file num_utilities.f90.

C.1.2.3 add_arr_mult_3_3()

```
integer function num_utilities::add_arr_mult::add_arr_mult_3_3 (
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) arr_1,
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) arr_2,
               real(dp), dimension(1:,1:,1:), intent(out) arr_3,
               integer, dimension(3), intent(in) deriv )
```

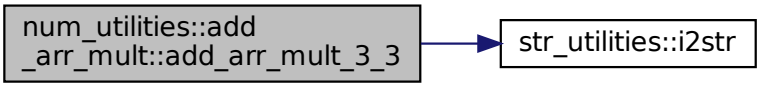
version with arr_1 and arr_2 in 3 coords.

Parameters

in	*arr_1*	arr_1
in	*arr_2*	arr_2
out	*arr_3*	arr_3
in	*deriv*	derivatives

Definition at line 363 of file num_utilities.f90.

Here is the call graph for this function:



C.2 mpi_utilities::broadcast_var Interface Reference

Wrapper function to broadcast a single variable using MPI.

Public Member Functions

- integer function broadcast_var_real (var, source)
  real version
- integer function broadcast_var_int (var, source)
  integer version
- integer function broadcast_var_log (var, source)
  logical version
- integer function broadcast_var_complex_arr (var, source)
  complex array version
- integer function broadcast_var_real_arr (var, source)
  real array version
- integer function broadcast_var_int_arr (var, source)
  integer array version
- integer function broadcast_var_log_arr (var, source)
  logical array version

C.2.1 Detailed Description

Wrapper function to broadcast a single variable using MPI.

Returns

ierr

Definition at line 87 of file MPI_utilities.f90.

C.2.2 Member Function/Subroutine Documentation

C.2.2.1 broadcast_var_complex_arr()

```
integer function mpi_utilities::broadcast_var::broadcast_var_complex_arr (
               complex(dp), dimension(:), intent(in) var,
               integer, intent(in), optional source )
```

complex array version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 659 of file MPI_utilities.f90.

C.2.2.2 broadcast_var_int()

```
integer function mpi_utilities::broadcast_var::broadcast_var_int (
               integer, intent(in) var,
               integer, intent(in), optional source )
```

integer version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 619 of file MPI_utilities.f90.

C.2.2.3 broadcast_var_int_arr()

```
integer function mpi_utilities::broadcast_var::broadcast_var_int_arr (
               integer, dimension(:), intent(in) var,
               integer, intent(in), optional source )
```

integer array version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 701 of file MPI_utilities.f90.

C.2.2.4 broadcast_var_log()

```
integer function mpi_utilities::broadcast_var::broadcast_var_log (
               logical, intent(in) var,
               integer, intent(in), optional source )
```

logical version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 639 of file MPI_utilities.f90.

C.2.2.5 broadcast_var_log_arr()

```
integer function mpi_utilities::broadcast_var::broadcast_var_log_arr (
               logical, dimension(:), intent(in) var,
               integer, intent(in), optional source )
```

logical array version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 721 of file MPI_utilities.f90.

C.2.2.6 broadcast_var_real()

```
integer function mpi_utilities::broadcast_var::broadcast_var_real (
               real(dp), intent(in) var,
               integer, intent(in), optional source )
```

real version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 598 of file MPI_utilities.f90.

C.2.2.7 broadcast_var_real_arr()

```
integer function mpi_utilities::broadcast_var::broadcast_var_real_arr (
               real(dp), dimension(:), intent(in) var,
               integer, intent(in), optional source )
```

real array version

Parameters

in	*var*	variable to be broadcast
in	*source*	process that sends

Definition at line 680 of file MPI_utilities.f90.

C.3 num_utilities::bubble_sort Interface Reference

Sorting with the bubble sort routine.

Public Member Functions

- subroutine bubble_sort_int (a, piv)
  integer version
- subroutine bubble_sort_real (a, piv)
  real version

C.3.1 Detailed Description

Sorting with the bubble sort routine.

Optionally, the pivots can be given back.

Note

Adapted from http://rosettacode.org/wiki/Category:Fortran

Definition at line 237 of file num_utilities.f90.

C.3.2 Member Function/Subroutine Documentation

C.3.2.1 bubble_sort_int()

```
subroutine num_utilities::bubble_sort::bubble_sort_int (
               integer, dimension(:), intent(inout) a,
               integer, dimension(:), intent(inout), optional piv )
```

integer version

Parameters

in,out	*a*	vector to sort
in,out	*piv*	pivots

Definition at line 1539 of file num_utilities.f90.

C.3.2.2 bubble_sort_real()

```
subroutine num_utilities::bubble_sort::bubble_sort_real (
               real(dp), dimension(:), intent(inout) a,
               integer, dimension(:), intent(inout), optional piv )
```

real version

Parameters

in,out	*a*	vector to sort
in,out	*piv*	pivots

Definition at line 1505 of file num_utilities.f90.

C.4 num_utilities::calc_det Interface Reference

Calculate determinant of a matrix.

Public Member Functions

- integer function calc_det_0d (det_0D, A)
  private constant version
- integer recursive function calc_det_3d (detA, A, n)
  private array version

C.4.1 Detailed Description

Calculate determinant of a matrix.

This matrix can be defined on a 3-D grid or constant. The storage convention described in eq_vars.eq_2_type is used.

In the former case the size of the matrix (last two indices) should be small, as the direct formula employing cofactors is used through a recursive formulation.

In the latter case, lapack routines are used.

See also

Adapted from http://dualm.wordpress.com/2012/01/06/computing-determinant-in-fortran/

Returns

ierr

Definition at line 63 of file num_utilities.f90.

C.4.2 Member Function/Subroutine Documentation

C.4.2.1 calc_det_0d()

```
integer function num_utilities::calc_det::calc_det_0d (
               real(dp), intent(inout) det_0D,
               real(dp), dimension(:,:), intent(in) A )
```

private constant version

Parameters

in,out	*det_0d*	output
in	*a*	input

Definition at line 622 of file num_utilities.f90.

C.4.2.2 calc_det_3d()

```
integer recursive function num_utilities::calc_det::calc_det_3d (
               real(dp), dimension(:,:,:), intent(inout) detA,
               real(dp), dimension(:,:,:,:), intent(in) A,
               integer, intent(in) n )
```

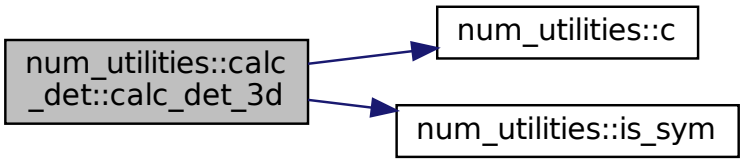
private array version

Parameters

in,out	*deta*	output
in	*a*	input
in	*n*	size of matrix

Definition at line 543 of file num_utilities.f90.

Here is the call graph for this function:



C.5 eq_ops::calc_eq Interface Reference

Calculate the equilibrium quantities on a grid determined by straight field lines.

Public Member Functions

- integer function calc_eq_1 (grid_eq, eq)
  flux version
- integer function calc_eq_2 (grid_eq, eq_1, eq_2, dealloc_vars)
  metric version

C.5.1 Detailed Description

Calculate the equilibrium quantities on a grid determined by straight field lines.

This grid has the dimensions (n_par,loc_n_r).

Optionally, for eq_2, the used variables can be deallocated on the fly, to limit memory usage.

Returns

ierr

Definition at line 48 of file eq_ops.f90.

C.5.2 Member Function/Subroutine Documentation

C.5.2.1 calc_eq_1()

```
integer function eq_ops::calc_eq::calc_eq_1 (
               type(grid_type), intent(inout) grid_eq,
               type(eq_1_type), intent(inout) eq )
```

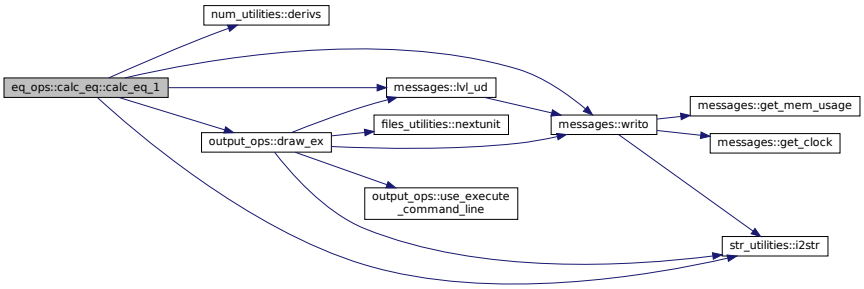
flux version

Parameters

in,out	*grid_eq*	equilibrium grid
in,out	*eq*	flux equilibrium variables

Definition at line 281 of file eq_ops.f90.

Here is the call graph for this function:



C.5.2.2 calc_eq_2()

```
integer function eq_ops::calc_eq::calc_eq_2 (
                type(grid_type), intent(inout) grid_eq,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(inout) eq_2,
                logical, intent(in), optional dealloc_vars )
```

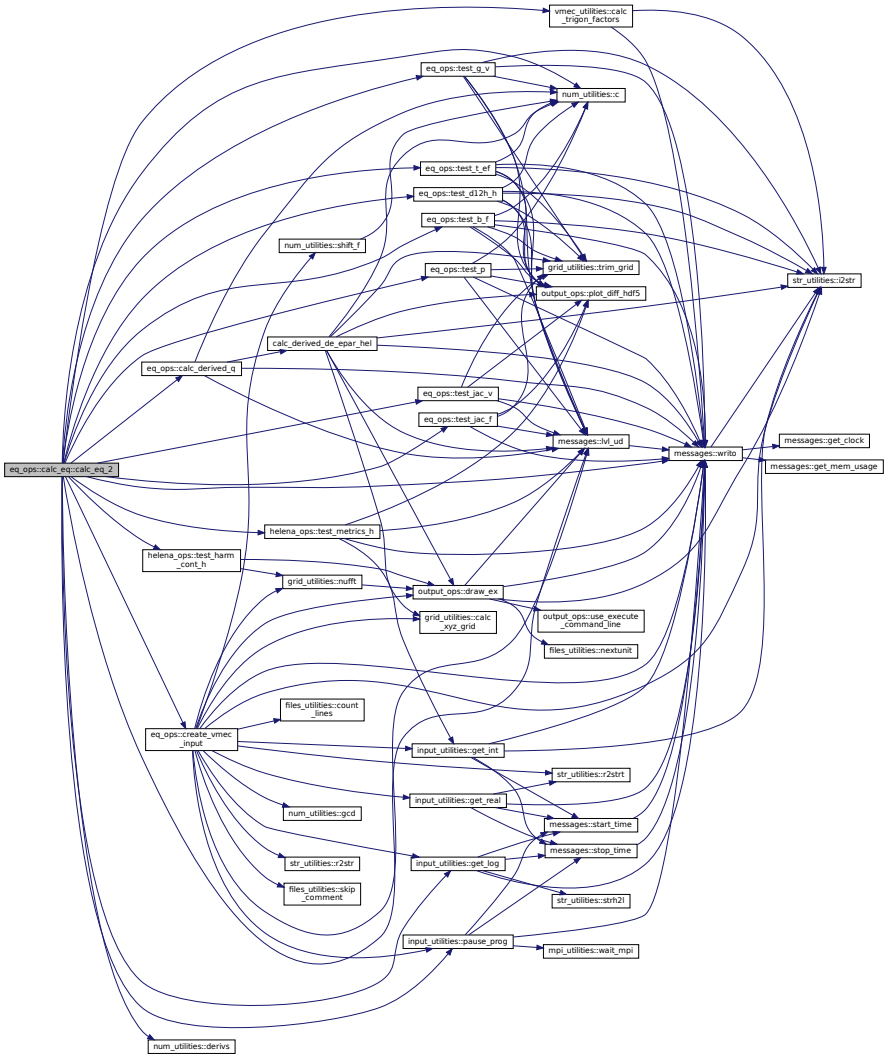
metric version

Parameters

in,out	*grid_eq*	equilibrium grid
in	*eq_1*	metric equilibrium variables
in,out	*eq_2*	metric equilibrium variables
in	*dealloc_vars*	deallocate variables on the fly after writing

Definition at line 435 of file eq_ops.f90.

Here is the call graph for this function:



C.6 grid_utilities::calc_eqd_grid Interface Reference

Calculate grid of equidistant points, where optionally the last point can be excluded.

Public Member Functions

- integer function calc_eqd_grid_1d (var, min_grid, max_grid, excl_last)
  1-D version
- integer function calc_eqd_grid_3d (var, min_grid, max_grid, grid_dim, excl_last)
  3-D version

C.6.1 Detailed Description

Calculate grid of equidistant points, where optionally the last point can be excluded.

Definition at line 75 of file grid_utilities.f90.

C.6.2 Member Function/Subroutine Documentation

C.6.2.1 calc_eqd_grid_1d()

```
integer function grid_utilities::calc_eqd_grid::calc_eqd_grid_1d (
               real(dp), dimension(:), intent(inout) var,
               real(dp), intent(in) min_grid,
               real(dp), intent(in) max_grid,
               logical, intent(in), optional excl_last )
```

1-D version

Parameters

in,out	*var*	output
in	*min_grid*	min. of angles [π]
in	*max_grid*	max. of angles [π]
in	*excl_last*	.true. if last point excluded

Definition at line 605 of file grid_utilities.f90.

C.6.2.2 calc_eqd_grid_3d()

```
integer function grid_utilities::calc_eqd_grid::calc_eqd_grid_3d (
               real(dp), dimension(:,:,:), intent(inout) var,
               real(dp), intent(in) min_grid,
               real(dp), intent(in) max_grid,
               integer, intent(in) grid_dim,
               logical, intent(in), optional excl_last )
```

3-D version

Parameters

in,out	*var*	output
in	*min_grid*	min. of angles [π]
in	*max_grid*	max. of angles [π]
in	*grid_dim*	in which dimension to create the grid
in	*excl_last*	.true. if last point excluded

Definition at line 529 of file grid_utilities.f90.

C.7 eq_utilities::calc_f_derivs Interface Reference

Transforms derivatives of the equilibrium quantities in E coordinates to derivatives in the F coordinates.

Public Member Functions

- integer function calc_f_derivs_1 (grid_eq, eq)
  flux version
- integer function calc_f_derivs_2 (eq)
  metric version

C.7.1 Detailed Description

Transforms derivatives of the equilibrium quantities in E coordinates to derivatives in the F coordinates.

Returns

ierr

Definition at line 36 of file eq_utilities.f90.

C.7.2 Member Function/Subroutine Documentation

C.7.2.1 calc_f_derivs_1()

```
integer function eq_utilities::calc_f_derivs::calc_f_derivs_1 (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(inout) eq )
```

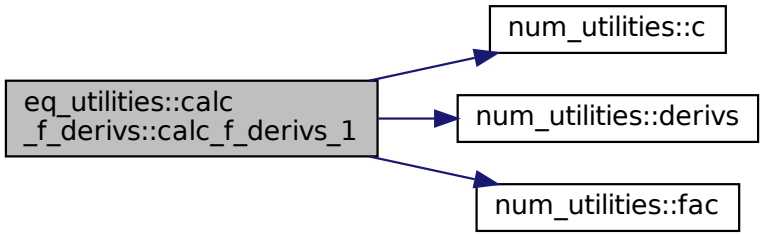
flux version

Parameters

in	*grid_eq*	equilibrium grid
in,out	*eq*	flux equilibrium variables

Definition at line 755 of file eq_utilities.f90.

Here is the call graph for this function:



C.7.2.2 calc_f_derivs_2()

```
integer function eq_utilities::calc_f_derivs::calc_f_derivs_2 (
                type(eq_2_type), intent(inout) eq )
```

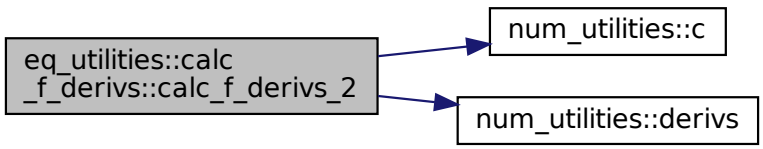
metric version

Parameters

in,out	*eq*	metric equilibrium variables

Definition at line 838 of file eq_utilities.f90.

Here is the call graph for this function:



C.8 eq_ops::calc_g_c Interface Reference

Calculate the lower metric elements in the C(ylindrical) coordinate system.

Public Member Functions

- integer function calc_g_c_ind (eq, deriv)
  individual version
- integer function calc_g_c_arr (eq, deriv)
  array version

C.8.1 Detailed Description

Calculate the lower metric elements in the C(ylindrical) coordinate system.

This is done directly using the formula's in [15]

Returns

ierr

Definition at line 139 of file eq_ops.f90.

C.8.2 Member Function/Subroutine Documentation

C.8.2.1 calc_g_c_arr()

```
integer function eq_ops::calc_g_c::calc_g_c_arr (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2858 of file eq_ops.f90.

C.8.2.2 calc_g_c_ind()

```
integer function eq_ops::calc_g_c::calc_g_c_ind (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:), intent(in) deriv )
```

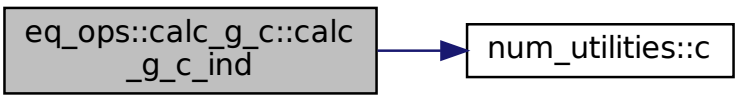
individual version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2827 of file eq_ops.f90.

Here is the call graph for this function:



C.9 eq_ops::calc_g_f Interface Reference

Calculate the metric coefficients in the F(lux) coordinate system.

Public Member Functions

- integer function calc_g_f_ind (eq, deriv)
  individual version
- integer function calc_g_f_arr (eq, deriv)
  array version

C.9.1 Detailed Description

Calculate the metric coefficients in the F(lux) coordinate system.

This is done using the metric coefficients in the equilibrium coordinate system and the transformation matrices.

Returns

ierr

Definition at line 183 of file eq_ops.f90.

C.9.2 Member Function/Subroutine Documentation

C.9.2.1 calc_g_f_arr()

```
integer function eq_ops::calc_g_f::calc_g_f_arr (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3114 of file eq_ops.f90.

C.9.2.2 calc_g_f_ind()

```
integer function eq_ops::calc_g_f::calc_g_f_ind (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:), intent(in) deriv )
```

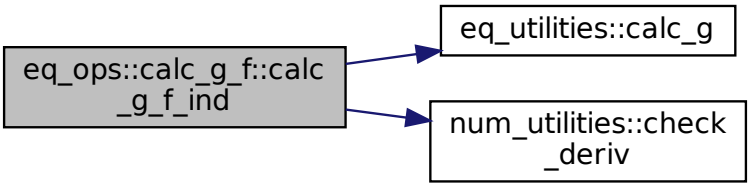
individual version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3091 of file eq_ops.f90.

Here is the call graph for this function:



C.10 eq_ops::calc_g_h Interface Reference

Calculate the lower metric coefficients in the equilibrium H(ELENA) coordinate system.

Public Member Functions

- integer function calc_g_h_ind (grid_eq, eq, deriv)
  individual version
- integer function calc_g_h_arr (grid_eq, eq, deriv)
  array version

C.10.1 Detailed Description

Calculate the lower metric coefficients in the equilibrium H(ELENA) coordinate system.

This is done using the HELENA output

Returns

ierr

Definition at line 169 of file eq_ops.f90.

C.10.2 Member Function/Subroutine Documentation

C.10.2.1 calc_g_h_arr()

```
integer function eq_ops::calc_g_h::calc_g_h_arr (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in	*grid_eq*	equilibirum grid
in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3070 of file eq_ops.f90.

C.10.2.2 calc_g_h_ind()

```
integer function eq_ops::calc_g_h::calc_g_h_ind (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:), intent(in) deriv )
```

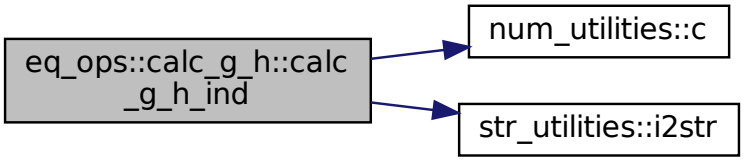
individual version

Parameters

in	*grid_eq*	equilibrium grid
in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2920 of file eq_ops.f90.

Here is the call graph for this function:



C.11 eq_ops::calc_g_v Interface Reference

Calculate the lower metric coefficients in the equilibrium V(MEC) coordinate system.

Public Member Functions

- integer function calc_g_v_ind (eq, deriv)
  individual version
- integer function calc_g_v_arr (eq, deriv)
  array version

C.11.1 Detailed Description

Calculate the lower metric coefficients in the equilibrium V(MEC) coordinate system.

This is done using the metric coefficients in the C(ylindrical) coordinate system and the transformation matrices

Note

It is assumed that the lower order derivatives have been calculated already. If not, the results will be incorrect.

Returns

ierr

Definition at line 156 of file eq_ops.f90.

C.11.2 Member Function/Subroutine Documentation

C.11.2.1 calc_g_v_arr()

```
integer function eq_ops::calc_g_v::calc_g_v_arr (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2900 of file eq_ops.f90.

C.11.2.2 calc_g_v_ind()

```
integer function eq_ops::calc_g_v::calc_g_v_ind (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:), intent(in) deriv )
```

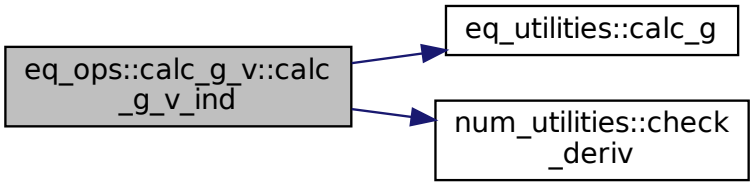
individual version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2878 of file eq_ops.f90.

Here is the call graph for this function:



C.12 num_utilities::calc_int Interface Reference

Integrates a function using the trapezoidal rule.

Public Member Functions

- integer function calc_int_eqd (var, step_size, var_int)
  equidistant version
- integer function calc_int_reg (var, x, var_int)
  regular version

C.12.1 Detailed Description

Integrates a function using the trapezoidal rule.

This function can be defined on an equidistant grid or a regular one:

- For an equidistant grid,
$$\int_1^n f(x)dx = \sum_{k=1}^{n-1} \left(f(k+1) + f(k)\right) \frac{\Delta_x}{2},$$
is used, with n the number of points, which are assumed to be equidistant with a given step size $Delta_x$.
- For a regular grid,
$$\int_1^n f(x)dx = \sum_{k=1}^{n-1} \left(f(k+1) + f(k)\right) \frac{x(k+1) - x(k)}{2},$$
is used, with n the number of points.
  - Therefore, n points have to be specified as well as n values for the function to be integrated.
  - They have to be given in ascending order but the step size does not have to be constant.
  - This yields the following difference formula:
$$\int_1^n f(x)dx = \int_1^{n-1} f(x)dx + \left(f(n) + f(n-1)\right) \frac{x(n) - x(n-1)}{2},$$
which is used here

Note

For periodic function, the trapezoidal rule works well only if the last point of the grid is included, i.e. the point where the function is equal to the first point.

Returns

ierr

Definition at line 160 of file num_utilities.f90.

C.12.2 Member Function/Subroutine Documentation

C.12.2.1 calc_int_eqd()

```
integer function num_utilities::calc_int::calc_int_eqd (
               real(dp), dimension(:), intent(in) var,
               real(dp), intent(in) step_size,
               real(dp), dimension(:), intent(inout) var_int )
```

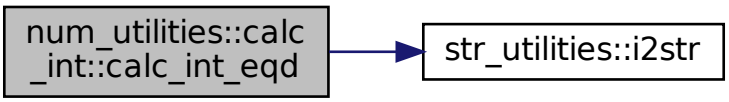
equidistant version

Parameters

in,out	*var_int*	integrated variable
in	*var*	variable to be integrated
in	*step_size*	step size of abscissa

Definition at line 327 of file num_utilities.f90.

Here is the call graph for this function:



C.12.2.2 calc_int_reg()

```
integer function num_utilities::calc_int::calc_int_reg (
               real(dp), dimension(:), intent(in) var,
               real(dp), dimension(:), intent(in) x,
               real(dp), dimension(:), intent(inout) var_int )
```

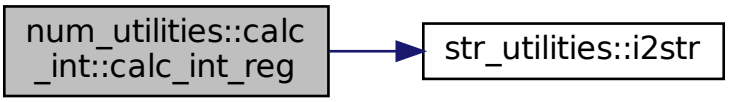
regular version

Parameters

in,out	*var_int*	integrated variable
in	*var*	variable to be integrated
in	*x*	abscissa

Definition at line 286 of file num_utilities.f90.

Here is the call graph for this function:



C.13 num_utilities::calc_inv Interface Reference

Calculate inverse of square matrix A.

Public Member Functions

- integer function calc_inv_0d (inv_0D, A)

  private constant version
- integer function calc_inv_3d (inv_3D, A, n)

  array version

C.13.1 Detailed Description

Calculate inverse of square matrix A.

This matrix can be defined on a 3-D grid or constant. The storage convention described in eq_vars.eq_2_type is used.

In the former case the size of the matrix (last two indices) should be small, as direct inversion is performed using Cramer's rule.

In the latter case, lapack routines are used.

Returns

ierr

Definition at line 81 of file num_utilities.f90.

C.13.2 Member Function/Subroutine Documentation

C.13.2.1 calc_inv_0d()

```
integer function num_utilities::calc_inv::calc_inv_0d (
               real(dp), dimension(:,:), intent(inout) inv_0D,
               real(dp), dimension(:,:), intent(in) A )
```

private constant version

Parameters

in,out	*inv_0d*	output
in	*a*	input

Definition at line 772 of file num_utilities.f90.

C.13.2.2 calc_inv_3d()

```
integer function num_utilities::calc_inv::calc_inv_3d (
               real(dp), dimension(:,:,:,:), intent(inout) inv_3D,
               real(dp), dimension(:,:,:,:), intent(in) A,
               integer, intent(in) n )
```

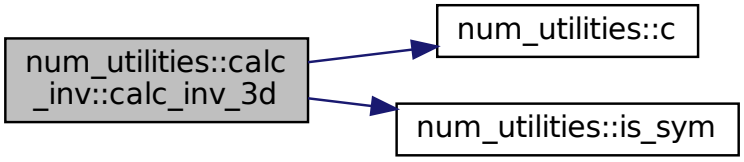
array version

Parameters

in,out	*inv_3d*	output
in	*a*	input
in	*n*	size of matrix

Definition at line 676 of file num_utilities.f90.

Here is the call graph for this function:



C.14 eq_utilities::calc_inv_met Interface Reference

Calculate $D_1^{m_1} D_2^{m_2} D_3^{m_3} X$ from $D_1^{i_1} D_2^{i_2} D_3^{i_3} X$ and $D_1^{j_1} D_2^{j_2} D_3^{j_3} Y$ where $XY = 1$ and $i, j = 0 \ldots m$, according to [15].

Public Member Functions

- integer function calc_inv_met_ind (X, Y, deriv)
  individual matrix version
- integer function calc_inv_met_arr (X, Y, deriv)
- integer function calc_inv_met_ind_0d (X, Y, deriv)
  individual scalar version
- integer function calc_inv_met_arr_0d (X, Y, deriv)

C.14.1 Detailed Description

Calculate $D_1^{m_1} D_2^{m_2} D_3^{m_3} X$ from $D_1^{i_1} D_2^{i_2} D_3^{i_3} X$ and $D_1^{j_1} D_2^{j_2} D_3^{j_3} Y$ where $XY = 1$ and $i, j = 0 \ldots m$, according to [15].

$D_1^{m_1} D_2^{m_2} D_3^{m_3}$ is defined as $\left(\frac{\partial}{\partial u^1}\right)^{m_1} \left(\frac{\partial}{\partial u^2}\right)^{m_2} \left(\frac{\partial}{\partial u^3}\right)^{m_3}$

Note

It is assumed that the lower order derivatives have been calculated already. If not, the results will be incorrect.

Returns

ierr

Definition at line 61 of file eq_utilities.f90.

C.14.2 Member Function/Subroutine Documentation

C.14.2.1 calc_inv_met_arr()

```
integer function eq_utilities::calc_inv_met::calc_inv_met_arr (
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(inout) X,
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(in) Y,
               integer, dimension(:,:), intent(in) deriv )
```

Parameters

in	*deriv*	derivatives

Definition at line 322 of file eq_utilities.f90.

C.14.2.2 calc_inv_met_arr_0d()

```
integer function eq_utilities::calc_inv_met::calc_inv_met_arr_0d (
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(inout) X,
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) Y,
               integer, dimension(:,:), intent(in) deriv )
```

Parameters

in	*deriv*	derivatives

Definition at line 342 of file eq_utilities.f90.

C.14.2.3 calc_inv_met_ind()

```
integer function eq_utilities::calc_inv_met::calc_inv_met_ind (
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(inout) X,
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(in) Y,
               integer, dimension(:), intent(in) deriv )
```

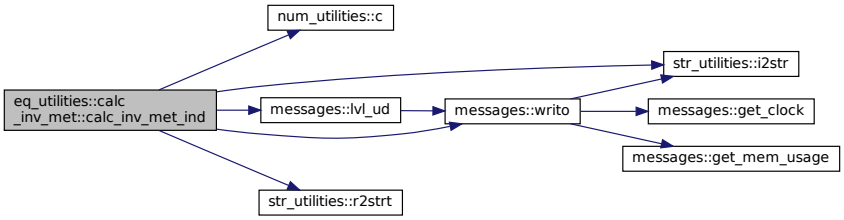
individual matrix version

Parameters

in	*deriv*	derivatives

Definition at line 132 of file eq_utilities.f90.

Here is the call graph for this function:



C.14.2.4 calc_inv_met_ind_0d()

```
integer function eq_utilities::calc_inv_met::calc_inv_met_ind_0d (
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(inout) X,
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) Y,
               integer, dimension(:), intent(in) deriv )
```

individual scalar version

Parameters

in	*deriv*	derivatives

Definition at line 261 of file eq_utilities.f90.

C.15 eq_ops::calc_jac_f Interface Reference

Calculate $\mathcal{J}_F$, the jacobian in Flux coordinates.

Public Member Functions

- integer function calc_jac_f_ind (eq, deriv)
  individual version
- integer function calc_jac_f_arr (eq, deriv)
  array version

C.15.1 Detailed Description

Calculate $\mathcal{J}_F$, the jacobian in Flux coordinates.

This is done directly from

$$\mathcal{J}_F = \det\left(\overline{T}_F^E\right)\mathcal{J}_E$$

Note

It is assumed that the lower order derivatives have been calculated already. If not, the results will be incorrect.

Returns

ierr

Definition at line 232 of file eq_ops.f90.

C.15.2 Member Function/Subroutine Documentation

C.15.2.1 calc_jac_f_arr()

```
integer function eq_ops::calc_jac_f::calc_jac_f_arr (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3331 of file eq_ops.f90.

C.15.2.2 calc_jac_f_ind()

```
integer function eq_ops::calc_jac_f::calc_jac_f_ind (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:), intent(in) deriv )
```

individual version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3304 of file eq_ops.f90.

C.16 eq_ops::calc_jac_h Interface Reference

Calculate $\mathcal{J}_{\mathrm{H}}$, the jacobian in HELENA coordinates.

Public Member Functions

- integer function calc_jac_h_ind (grid_eq, eq_1, eq_2, deriv)
  individual version
- integer function calc_jac_h_arr (grid_eq, eq_1, eq_2, deriv)
  array version

C.16.1 Detailed Description

Calculate $\mathcal{J}_{\mathrm{H}}$, the jacobian in HELENA coordinates.

This is done directly from

$$\mathcal{J}_{\mathrm{H}} = q\frac{R^2}{F}$$

Note

It is assumed that the lower order derivatives have been calculated already. If not, the results will be incorrect.

Returns

ierr

Definition at line 213 of file eq_ops.f90.

C.16.2 Member Function/Subroutine Documentation

C.16.2.1 calc_jac_h_arr()

```
integer function eq_ops::calc_jac_h::calc_jac_h_arr (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout) eq_2,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in	*grid_eq*	equilibrim grid
in	*eq_1*	flux equilibrium
in,out	*eq_2*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3282 of file eq_ops.f90.

C.16.2.2 calc_jac_h_ind()

```
integer function eq_ops::calc_jac_h::calc_jac_h_ind (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout) eq_2,
               integer, dimension(:), intent(in) deriv )
```

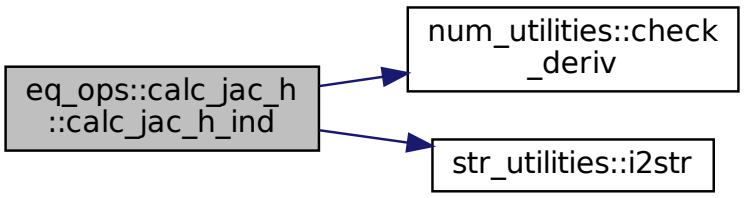
individual version

Parameters

in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in,out	*eq_2*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3186 of file eq_ops.f90.

Here is the call graph for this function:



C.17 eq_ops::calc_jac_v Interface Reference

Calculate $\mathcal{J}_V$, the jacobian in V(MEC) coordinates.

Public Member Functions

- integer function calc_jac_v_ind (grid, eq, deriv)
  individual version
- integer function calc_jac_v_arr (grid, eq, deriv)
  array version

C.17.1 Detailed Description

Calculate $\mathcal{J}_V$, the jacobian in V(MEC) coordinates.

This is done using VMEC output.

Returns

ierr

Definition at line 196 of file eq_ops.f90.

C.17.2 Member Function/Subroutine Documentation

C.17.2.1 calc_jac_v_arr()

```
integer function eq_ops::calc_jac_v::calc_jac_v_arr (
               type(grid_type), intent(in) grid,
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in	*grid*	grid for which to calculate Jacobian
in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3165 of file eq_ops.f90.

C.17.2.2 calc_jac_v_ind()

```
integer function eq_ops::calc_jac_v::calc_jac_v_ind (
               type(grid_type), intent(in) grid,
```

```
type(eq_2_type), intent(inout) eq,
integer, dimension(:), intent(in) deriv )
```

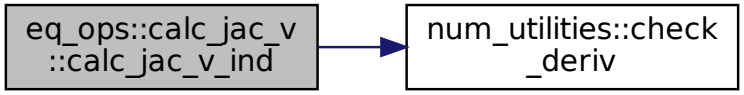
individual version

Parameters

in	*grid*	grid for which to calculate Jacobian
in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3134 of file eq_ops.f90.

Here is the call graph for this function:



C.18 num_utilities::calc_mult Interface Reference

Calculate matrix multiplication of two square matrices $\overline{AB} = \overline{A}\,\overline{B}$.

Public Member Functions

- integer function calc_mult_0d_real (A, B, AB, n, transp)
  real constant version
- integer function calc_mult_3d_real (A, B, AB, n, transp)
  real array version
- integer function calc_mult_3d_complex (A, B, AB, n, transp)
  complex array version

C.18.1 Detailed Description

Calculate matrix multiplication of two square matrices $\overline{AB} = \overline{A}\,\overline{B}$.

This matrix can be defined on a 3-D grid or constant. The storage convention described in eq_vars.eq_2_type is used.

Returns

ierr

Definition at line 95 of file num_utilities.f90.

C.18.2 Member Function/Subroutine Documentation

C.18.2.1 calc_mult_0d_real()

```
integer function num_utilities::calc_mult::calc_mult_0d_real (
               real(dp), dimension(:), intent(in) A,
               real(dp), dimension(:), intent(in) B,
               real(dp), dimension(:), intent(inout) AB,
               integer, intent(in) n,
               logical, dimension(2), intent(in), optional transp )
```

real constant version

Parameters

in	*a*	input A
in	*b*	input B
in,out	*ab*	output A B
in	*n*	size of matrix
in	*transp*	whether A and/or B transposed

Definition at line 984 of file num_utilities.f90.

C.18.2.2 calc_mult_3d_complex()

```
integer function num_utilities::calc_mult::calc_mult_3d_complex (
               complex(dp), dimension(:,:,:,:), intent(in) A,
               complex(dp), dimension(:,:,:,:), intent(in) B,
               complex(dp), dimension(:,:,:,:), intent(inout) AB,
               integer, intent(in) n,
               logical, dimension(2), intent(in), optional transp )
```

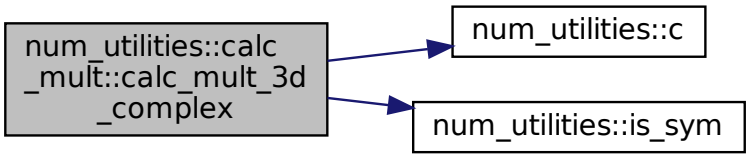
complex array version

Parameters

in	*a*	input A
in	*b*	input B
in,out	*ab*	output A B
in	*n*	size of matrix
in	*transp*	whether A and/or B transposed

Definition at line 898 of file num_utilities.f90.

Here is the call graph for this function:



C.18.2.3 calc_mult_3d_real()

```
integer function num_utilities::calc_mult::calc_mult_3d_real (
               real(dp), dimension(:,:,:,:), intent(in) A,
               real(dp), dimension(:,:,:,:), intent(in) B,
               real(dp), dimension(:,:,:,:), intent(inout) AB,
               integer, intent(in) n,
               logical, dimension(2), intent(in), optional transp )
```

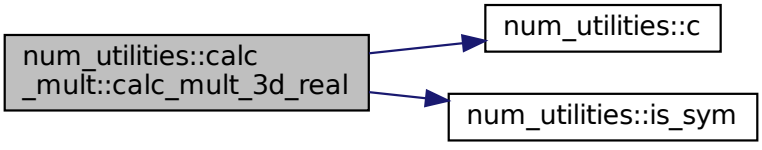
real array version

Parameters

in	*a*	input A
in	*b*	input B
in,out	*ab*	output A B
in	*n*	size of matrix
in	*transp*	whether A and/or B transposed

Definition at line 817 of file num_utilities.f90.

Here is the call graph for this function:



C.19 eq_ops::calc_rzl Interface Reference

Calculate R, Z & λ and derivatives in VMEC coordinates.

Public Member Functions

- integer function calc_rzl_ind (grid_eq, eq, deriv)
  individual version
- integer function calc_rzl_arr (grid_eq, eq, deriv)
  array version

C.19.1 Detailed Description

Calculate R, Z & λ and derivatives in VMEC coordinates.

This is done at the grid points given by the variables `theta_E` and `zeta_E` (contained in `trigon_factors`) and at every normal point.

The derivatives are indicated by the variable `deriv` which has 3 indices

Note

There is no HELENA equivalent because for HELENA simulations, R and Z are not necessary for calculation of the metric coefficients, and λ does not exist.

See also

calc_trigon_factors()

Returns

ierr

Definition at line 126 of file eq_ops.f90.

C.19.2 Member Function/Subroutine Documentation

C.19.2.1 calc_rzl_arr()

```
integer function eq_ops::calc_rzl::calc_rzl_arr (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in	*grid_eq*	equilibrium grid
in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2806 of file eq_ops.f90.

C.19.2.2 calc_rzl_ind()

```
integer function eq_ops::calc_rzl::calc_rzl_ind (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(inout) eq,
               integer, dimension(3), intent(in) deriv )
```

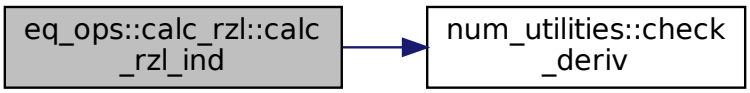
individual version

Parameters

in	*grid_eq*	equilibrium grid
in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 2768 of file eq_ops.f90.

Here is the call graph for this function:



C.20 eq_ops::calc_t_hf Interface Reference

Calculate $\overline{T}^{F}_{H}$, the transformation matrix between H(ELENA) and F(lux) coordinate systems.

Public Member Functions

- integer function calc_t_hf_ind (grid_eq, eq_1, eq_2, deriv)
  individual version
- integer function calc_t_hf_arr (grid_eq, eq_1, eq_2, deriv)
  array version

C.20.1 Detailed Description

Calculate $\bar{\bar{T}}^{F}_{H}$, the transformation matrix between H(ELENA) and F(lux) coordinate systems.

This is done directly using the formula's in [15]

Returns

ierr

Definition at line 271 of file eq_ops.f90.

C.20.2 Member Function/Subroutine Documentation

C.20.2.1 calc_t_hf_arr()

```
integer function eq_ops::calc_t_hf::calc_t_hf_arr (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout) eq_2,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in,out	*eq_2*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3780 of file eq_ops.f90.

C.20.2.2 calc_t_hf_ind()

```
integer function eq_ops::calc_t_hf::calc_t_hf_ind (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout) eq_2,
               integer, dimension(:), intent(in) deriv )
```

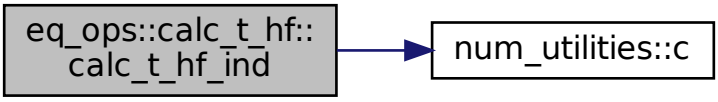
individual version

Parameters

in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in,out	*eq_2*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3607 of file eq_ops.f90.

Here is the call graph for this function:



C.21 eq_ops::calc_t_vc Interface Reference

Calculate $\overline{\overline{T}}^{V}_{C}$, the transformation matrix between C(ylindrical) and V(mec) coordinate system.

Public Member Functions

- integer function calc_t_vc_ind (eq, deriv)
  individual version
- integer function calc_t_vc_arr (eq, deriv)
  array version

C.21.1 Detailed Description

Calculate $\overline{\overline{T}}^{V}_{C}$, the transformation matrix between C(ylindrical) and V(mec) coordinate system.

This is done directly using the formula's in [15]

Returns

ierr

Definition at line 245 of file eq_ops.f90.

C.21.2 Member Function/Subroutine Documentation

C.21.2.1 calc_t_vc_arr()

```
integer function eq_ops::calc_t_vc::calc_t_vc_arr (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3394 of file eq_ops.f90.

C.21.2.2 calc_t_vc_ind()

```
integer function eq_ops::calc_t_vc::calc_t_vc_ind (
               type(eq_2_type), intent(inout) eq,
               integer, dimension(:), intent(in) deriv )
```

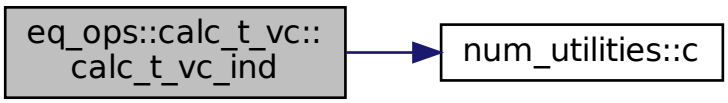
individual version

Parameters

in,out	*eq*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3351 of file eq_ops.f90.

Here is the call graph for this function:



C.22 eq_ops::calc_t_vf Interface Reference

Calculate $\bar{\bar{T}}^F_V$, the transformation matrix between V(MEC) and F(lux) coordinate systems.

Public Member Functions

- integer function calc_t_vf_ind (grid_eq, eq_1, eq_2, deriv)
  individual version
- integer function calc_t_vf_arr (grid_eq, eq_1, eq_2, deriv)
  array version

C.22.1 Detailed Description

Calculate $\bar{\bar{T}}^F_V$, the transformation matrix between V(MEC) and F(lux) coordinate systems.

This is done directly using the formula's in [15]

Returns

ierr

Definition at line 258 of file eq_ops.f90.

C.22.2 Member Function/Subroutine Documentation

C.22.2.1 calc_t_vf_arr()

```
integer function eq_ops::calc_t_vf::calc_t_vf_arr (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout) eq_2,
               integer, dimension(:,:), intent(in) deriv )
```

array version

Parameters

in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in,out	*eq_2*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3585 of file eq_ops.f90.

C.22.2.2 calc_t_vf_ind()

```
integer function eq_ops::calc_t_vf::calc_t_vf_ind (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(inout) eq_2,
               integer, dimension(:), intent(in) deriv )
```

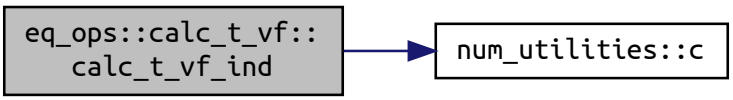
individual version

Parameters

in	*grid_eq*	equilibrium grid
in	*eq_1*	flux equilibrium
in,out	*eq_2*	metric equilibrium
in	*deriv*	derivatives

Definition at line 3414 of file eq_ops.f90.

Here is the call graph for this function:



C.23 grid_utilities::calc_tor_diff Interface Reference

Calculates the toroidal difference for a magnitude calculated on three toroidal points: two extremities and one in the middle.

Public Member Functions

- integer function calc_tor_diff_0d (v_mag, theta, norm_disc_prec, absolute, r)
  0-D version
- integer function calc_tor_diff_2d (v_com, theta, norm_disc_prec, absolute, r)
  2-D version

C.23.1 Detailed Description

Calculates the toroidal difference for a magnitude calculated on three toroidal points: two extremities and one in the middle.

This is done using the formula

$$\frac{b-a}{b+a},$$

for the relative difference between a and b. This is useful when it is used to calculate a toroidal ripple and these are the extreme points.

The procedure also needs the map between the flux poloidal angle and the geometrical poloidal angle.

In a first step the quantity is interpolated on an equidistant grid in the geometrical poloidal angle.

The difference is then calculated for values of constant geometric poloidal angle.

Finally, this result is transformed back to the Flux coordinates.

Note

1. The theta map should have first and last point equal.
2. This routine should be used only for periodic quantities (so not for some of the Flux quantities that are defined along generally non-rational magnetic field lines).

Returns

ierr

Definition at line 109 of file grid_utilities.f90.

C.23.2 Member Function/Subroutine Documentation

C.23.2.1 calc_tor_diff_0d()

```
integer function grid_utilities::calc_tor_diff::calc_tor_diff_0d (
               real(dp), dimension(:,:,:), intent(inout) v_mag,
               real(dp), dimension(:,:,:), intent(in) theta,
               integer, intent(in) norm_disc_prec,
               logical, intent(in), optional absolute,
               real(dp), dimension(:), intent(in), optional r )
```

0-D version

Parameters

in,out	*v_mag*	magnitude (dim1,dim2,dim3)
in	*theta*	geometric poloidal angle
in	*norm_disc_prec*	precision for normal derivatives
in	*absolute*	calculate absolute, not relative, difference
in	*r*	normal positions for theta

Definition at line 746 of file grid_utilities.f90.

C.23.2.2 calc_tor_diff_2d()

```
integer function grid_utilities::calc_tor_diff::calc_tor_diff_2d (
               real(dp), dimension(:,:,:,:,:), intent(inout) v_com,
               real(dp), dimension(:,:,:), intent(in) theta,
               integer, intent(in) norm_disc_prec,
               logical, intent(in), optional absolute,
               real(dp), dimension(:), intent(in), optional r )
```

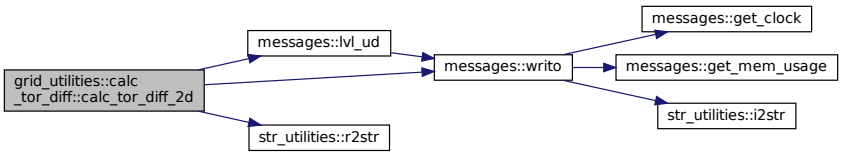
2-D version

Parameters

in,out	*v_com*	covariant and contravariant components (dim1,dim2,dim3,3,2)
in	*theta*	geometric poloidal angle
in	*norm_disc_prec*	precision for normal derivatives
in	*absolute*	calculate absolute, not relative, difference
in	*r*	normal positions for theta

Definition at line 633 of file grid_utilities.f90.

Here is the call graph for this function:



C.24 x_ops::calc_x Interface Reference

Calculates either vectorial or tensorial perturbation variables.

Public Member Functions

- integer function calc_x_1 (mds, grid_eq, grid_X, eq_1, eq_2, X, lim_sec_X)
  vectorial version
- integer function calc_x_2 (mds, grid_eq, grid_X, eq_1, eq_2, X_a, X_b, X, lim_sec_X)
  tensorial version

C.24.1 Detailed Description

Calculates either vectorial or tensorial perturbation variables.

Optionally, the secondary mode number can be specified (`m` if poloidal flux is used as normal coordinate and c n if toroidal flux). By default, they are taken from the global `X_vars` variables.

Returns

ierr

Definition at line 39 of file X_ops.f90.

C.24.2 Member Function/Subroutine Documentation

C.24.2.1 calc_x_1()

```
integer function x_ops::calc_x::calc_x_1 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               type(x_1_type), intent(inout) X,
               integer, dimension(2), intent(in), optional lim_sec_X )
```

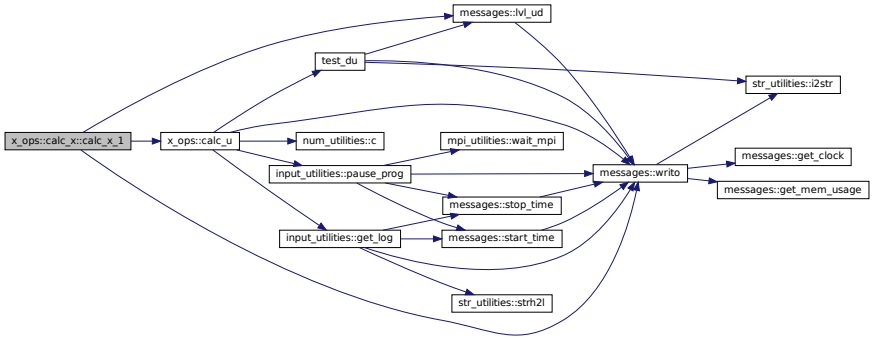
vectorial version

Parameters

in	*mds*	general modes variables
in	*grid_eq*	equilibrium grid variables
in	*grid_x*	perturbation grid variables
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in,out	*x*	vectorial perturbation variables
in	*lim_↩ sec_x*	limits of m_X (pol. flux) or \ n_X (tor. flux)

Definition at line 96 of file X_ops.f90.

Here is the call graph for this function:



C.24.2.2 calc_x_2()

```
integer function x_ops::calc_x::calc_x_2 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_eq,
               type(grid_type), intent(in) grid_X,
               type(eq_1_type), intent(in) eq_1,
               type(eq_2_type), intent(in) eq_2,
               type(x_1_type), intent(inout) X_a,
               type(x_1_type), intent(inout) X_b,
               type(x_2_type), intent(inout) X,
               integer, dimension(2,2), intent(in), optional lim_sec_X )
```

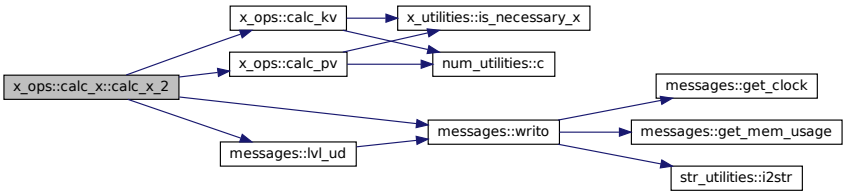
tensorial version

Parameters

in	*mds*	general modes variables
in	*grid_eq*	equilibrium grid variables
in	*grid_x*	perturbation grid variables
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in,out	*x_a*	vectorial perturbation variables (dimension 1)
in,out	*x_b*	vectorial perturbation variables (dimension 2)
in,out	*x*	tensorial perturbation variables
in	*lim_sec_x*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions

Definition at line 131 of file X_ops.f90.

Here is the call graph for this function:



C.25 sol_utilities::calc_xuq Interface Reference

Calculates the normal ($\cdot_n$) or geodesic ($\cdot_g$) components of the plasma perturbation $\vec{\xi}$ or the magnetic perturbation $\vec{Q} = \nabla \times \left(\vec{x} \times \vec{B} \right)$.

Public Member Functions

- integer function calc_xuq_ind (mds_X, mds_sol, grid_eq, grid_X, grid_sol, eq_1, eq_2, X, sol, X_id, XU↩ Q_style, time, XUQ, deriv)
  (time) individual version
- integer function calc_xuq_arr (mds_X, mds_sol, grid_eq, grid_X, grid_sol, eq_1, eq_2, X, sol, X_id, XU↩ Q_style, time, XUQ, deriv)
  (time) array version

C.25.1 Detailed Description

Calculates the normal ($\cdot_n$) or geodesic ($\cdot_g$) components of the plasma perturbation $\vec{\xi}$ or the magnetic perturbation $\vec{Q} = \nabla \times \left(\vec{x} \times \vec{B} \right)$.

Which variables are plot depends on `XUQ_style` [15] :

- `XUQ_style` = 1: X (supports parallel derivative)
- `XUQ_style` = 2: U (supports parallel derivative)
- `XUQ_style` = 3: Q_n
- `XUQ_style` = 4: Q_g

X and U support the calculation of the parallel derivative.

For Q_n and Q_g, the metric variables have to be provided as well.

The perturbation grid is assumed to have the same angular coordinates as the equilibrium grid, and the normal coordinates correspond to either the equilibrium grid (`X_grid_style` 1), the solution grid (`X_grid↩ _style` 2) or the enriched equilibrium grid (`X_grid_style` 3).

The output grid, furthermore, has the angular part corresponding to the equilibrium grid, and the normal part to the solution grid.

Returns

ierr

Definition at line 56 of file sol_utilities.f90.

C.25.2 Member Function/Subroutine Documentation

C.25.2.1 calc_xuq_arr()

```
integer function sol_utilities::calc_xuq::calc_xuq_arr (
                type(modes_type), intent(in), target mds_X,
                type(modes_type), intent(in), target mds_sol,
                type(grid_type), intent(in) grid_eq,
                type(grid_type), intent(in) grid_X,
                type(grid_type), intent(in) grid_sol,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(in) eq_2,
                type(x_1_type), intent(in), target X,
                type(sol_type), intent(in) sol,
                integer, intent(in) X_id,
                integer, intent(in) XUQ_style,
                real(dp), dimension(:), intent(in) time,
                complex(dp), dimension(:,:,:,:), intent(inout) XUQ,
                logical, intent(in), optional deriv )
```

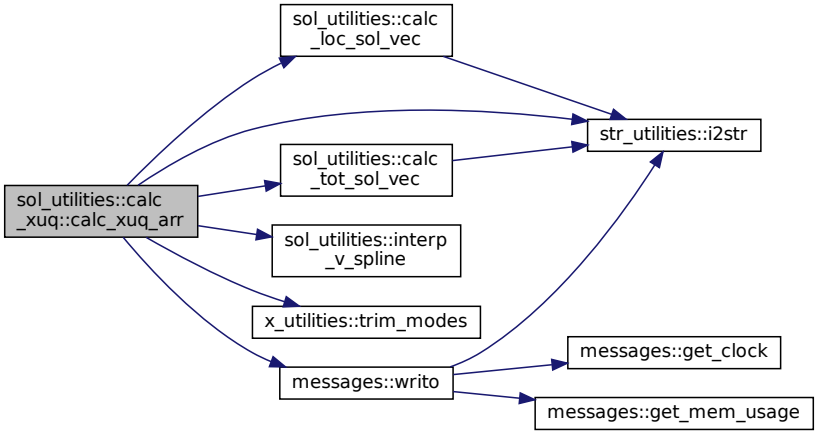
(time) array version

Parameters

in	*mds_x*	general modes variables in perturbation grid
in	*mds_sol*	general modes variables in solution grid
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*grid_sol*	solution grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x*	perturbation variables
in	*sol*	solution variables
in	*x_id*	nr. of Eigenvalue
in	*xuq_style*	whether to calculate X, U, Q_n or Q_g
in	*time*	time range
in,out	*xuq*	X, U, Q_n or Q_g
in	*deriv*	return parallel derivative

Definition at line 67 of file sol_utilities.f90.

Here is the call graph for this function:



C.25.2.2 calc_xuq_ind()

```
integer function sol_utilities::calc_xuq::calc_xuq_ind (
                type(modes_type), intent(in) mds_X,
                type(modes_type), intent(in) mds_sol,
                type(grid_type), intent(in) grid_eq,
                type(grid_type), intent(in) grid_X,
                type(grid_type), intent(in) grid_sol,
                type(eq_1_type), intent(in) eq_1,
                type(eq_2_type), intent(in) eq_2,
                type(x_1_type), intent(in) X,
                type(sol_type), intent(in) sol,
                integer, intent(in) X_id,
                integer, intent(in) XUQ_style,
                real(dp), intent(in) time,
                complex(dp), dimension(:,:,:), intent(inout) XUQ,
                logical, intent(in), optional deriv )
```

(time) individual version

Parameters

in	*mds_x*	general modes variables in perturbation grid
in	*mds_sol*	general modes variables in solution grid
in	*grid_eq*	equilibrium grid
in	*grid_x*	perturbation grid
in	*grid_sol*	solution grid
in	*eq_1*	flux equilibrium
in	*eq_2*	metric equilibrium
in	*x*	perturbation variables

Parameters

in	*sol*	solution variables
in	*x_id*	nr. of Eigenvalue
in	*xuq_style*	whether to calculate X, U, Qn or Qg
in,out	*xuq*	normal component of perturbation
in	*deriv*	return parallel derivative

Definition at line 520 of file sol_utilities.f90.

C.26 num_ops::calc_zero_hh Interface Reference

Finds the zero of a function using Householder iteration.

Public Member Functions

- character(len=max_str_ln) function calc_zero_hh_0d (zero, fun, ord, guess, max_nr_backtracks, output)
  0-D version
- character(len=max_str_ln) function calc_zero_hh_3d (dims, zero, fun, ord, guess, max_nr_backtracks, output)
  3-D version

C.26.1 Detailed Description

Finds the zero of a function using Householder iteration.

If something goes wrong, by default multiple tries can be attempted, by backtracking on the correction by multiplying it by a relaxation factor. This can be done `max_nr_backtracks` times.

If still nothing is achieved, an error message is returned, that is empty otherwise.

Definition at line 35 of file num_ops.f90.

C.26.2 Member Function/Subroutine Documentation

C.26.2.1 calc_zero_hh_0d()

```
character(len=max_str_ln) function num_ops::calc_zero_hh::calc_zero_hh_0d (
               real(dp), intent(inout) zero,
                fun,
               integer, intent(in) ord,
               real(dp), intent(in) guess,
               integer, intent(in), optional max_nr_backtracks,
               logical, intent(in), optional output )
```

0-D version

Parameters

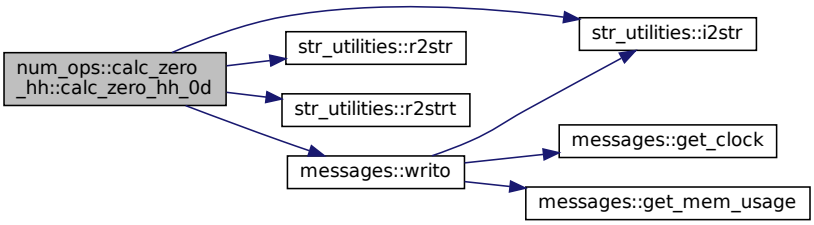
in,out	*fun*	fun(x,ord) with • x abscissa • ord order of derivative • fun ordinate
in,out	*zero*	output
in	*ord*	order of solution
in	*guess*	first guess
in	*max_nr_backtracks*	max nr. of tries with different relaxation factors
in	*output*	give output on convergence

Returns

possible error message

Definition at line 50 of file num_ops.f90.

Here is the call graph for this function:



C.26.2.2 calc_zero_hh_3d()

```
character(len=max_str_ln) function num_ops::calc_zero_hh::calc_zero_hh_3d (
               integer, dimension(3), intent(in) dims,
               real(dp), dimension(dims(1),dims(2),dims(3)), intent(inout) zero,
                fun,
               integer, intent(in) ord,
               real(dp), dimension(dims(1),dims(2),dims(3)), intent(in) guess,
               integer, intent(in), optional max_nr_backtracks,
               logical, intent(in), optional output )
```

3-D version

Parameters

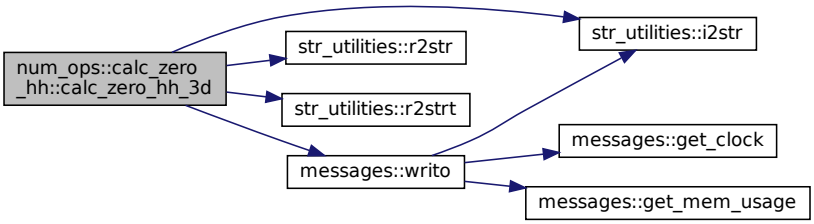
`in,out`	*fun*	`fun(dims,x,ord)` with • `dims(3)` dimension of abscissa and ordinate • `x(dims(1),dims(2),dims(3))` abscissa • `ord` order of derivative • `fun(dims(1),dims(2),dims(3))` ordinate
`in`	*dims*	dimensions of the problem
`in,out`	*zero*	output
`in`	*ord*	order of solution
`in`	*guess*	first guess
`in`	*max_nr_backtracks*	max nr. of backtracks
`in`	*output*	give output on convergence

Returns

possible error message

Definition at line 243 of file num_ops.f90.

Here is the call graph for this function:



C.27 num_utilities::con Interface Reference

Either takes the complex conjugate of a square matrix element A, defined on a 3-D grid, or not.

Public Member Functions

- complex(dp) function, dimension(d(1), d(2), d(3)) con_3d (A, c, sym, d)
  3-D version
- complex(dp) function, dimension(d(1), d(2)) con_2d (A, c, sym, d)
  2-D version
- complex(dp) function, dimension(d(1)) con_1d (A, c, sym, d)
  1-D version
- complex(dp) function con_0d (A, c, sym)
  0-D version

C.27.1 Detailed Description

Either takes the complex conjugate of a square matrix element A, defined on a 3-D grid, or not.

This is done depending on whether the indices of the matrix element correspond to the upper (conjugate) or lower (no conjugate) triangular part.

Note

Only for symmetric matrices does this have to be applied.

Definition at line 221 of file num_utilities.f90.

C.27.2 Member Function/Subroutine Documentation

C.27.2.1 con_0d()

```
complex(dp) function num_utilities::con::con_0d (
               complex(dp), intent(in) A,
               integer, dimension(2), intent(in) c,
               logical, intent(in) sym )
```

0-D version

Parameters

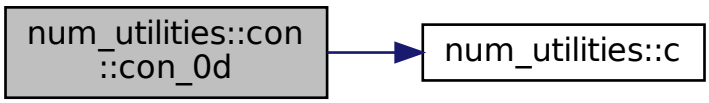
in	*a*	input A
in	*c*	indices in square matrix
in	*sym*	if A is symmetric

Returns

output B

Definition at line 1490 of file num_utilities.f90.

Here is the call graph for this function:



C.27.2.2 con_1d()

```
complex(dp) function, dimension(d(1)) num_utilities::con::con_1d (
               complex(dp), dimension(:), intent(in) A,
               integer, dimension(2), intent(in) c,
               logical, intent(in) sym,
               integer, dimension(1), intent(in) d )
```

1-D version

Parameters

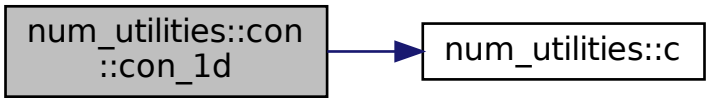
in	*a*	input A
in	*c*	indices in square matrix
in	*sym*	if A is symmetric
in	*d*	dimensions of matrix A

Returns

output B

Definition at line 1475 of file num_utilities.f90.

Here is the call graph for this function:



C.27.2.3 con_2d()

```
complex(dp) function, dimension(d(1),d(2)) num_utilities::con::con_2d (
               complex(dp), dimension(:,:), intent(in) A,
               integer, dimension(2), intent(in) c,
               logical, intent(in) sym,
               integer, dimension(2), intent(in) d )
```

2-D version

Parameters

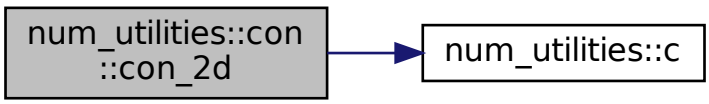
in	*a*	input A
in	*c*	indices in square matrix
in	*sym*	if A is symmetric
in	*d*	dimensions of matrix A

Returns

output B

Definition at line 1460 of file num_utilities.f90.

Here is the call graph for this function:



C.27.2.4 con_3d()

```
complex(dp) function, dimension(d(1),d(2),d(3)) num_utilities::con::con_3d (
               complex(dp), dimension(:,:,:), intent(in) A,
               integer, dimension(2), intent(in) c,
               logical, intent(in) sym,
               integer, dimension(3), intent(in) d )
```

3-D version

Parameters

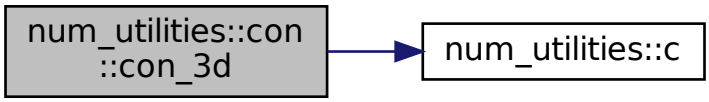
in	*a*	input A
in	*c*	indices in square matrix
in	*sym*	if A is symmetric
in	*d*	dimensions of matrix A

Returns

output B

Definition at line 1445 of file num_utilities.f90.

Here is the call graph for this function:



C.28 num_utilities::con2dis Interface Reference

Convert between points from a continuous grid to a discrete grid.

Public Member Functions

- integer function con2dis_eqd (pt_c, pt_d, lim_c, lim_d)
  equidistant version
- integer function con2dis_reg (pt_c, pt_d, var_c)
  regular version

C.28.1 Detailed Description

Convert between points from a continuous grid to a discrete grid.

This is done by providing either the the limits on the grid (`lim_c` and `lim_d`), in which case the grid is assumed to be equidistant, or the grid values themselves, in which case the grid just has to be regular.

The output is a real value where the floored integer is the index in the discrete grid and the remainder corresponds to the fraction towards the next index. If no solution is found, a negative value is outputted, as well as a message.

Returns

ierr

Definition at line 188 of file num_utilities.f90.

C.28.2 Member Function/Subroutine Documentation

C.28.2.1 con2dis_eqd()

```
integer function num_utilities::con2dis::con2dis_eqd (
               real(dp), intent(in) pt_c,
               real(dp), intent(inout) pt_d,
               real(dp), dimension(2), intent(in) lim_c,
               integer, dimension(2), intent(in) lim_d )
```

equidistant version

Parameters

in	*pt_c*	point on continous grid
in,out	*pt_d*	point on discrete grid
in	*lim_c*	[min_c,max_c]
in	*lim_d*	[min_d,max_d]

Definition at line 1206 of file num_utilities.f90.

C.28.2.2 con2dis_reg()

```
integer function num_utilities::con2dis::con2dis_reg (
               real(dp), intent(in) pt_c,
               real(dp), intent(inout) pt_d,
               real(dp), dimension(:), intent(in) var_c )
```

regular version

Parameters

in	*pt_c*	point on continous grid
in,out	*pt_d*	point on discrete grid
in	*var_c*	continous grid values

Definition at line 1226 of file num_utilities.f90.

C.29 pb3d_utilities::conv_1d2nd Interface Reference

Converts 1-D to n-D variables.

Public Member Functions

- subroutine conv_1d2nd_1d (var_in, var_out)
  1-D version
- subroutine conv_1d2nd_2d (var_in, var_out)
  2-D version
- subroutine conv_1d2nd_3d (var_in, var_out)
  3-D version

- subroutine conv_1d2nd_4d (var_in, var_out)
  4-D version
- subroutine conv_1d2nd_6d (var_in, var_out)
  6-D version
- subroutine conv_1d2nd_7d (var_in, var_out)
  7-D version

C.29.1 Detailed Description

Converts 1-D to n-D variables.

The 1-D variables are used for internal storage in HDF5, whereas the n_D variables correspond to the ones in PB3D.

Definition at line 21 of file PB3D_utilities.f90.

C.29.2 Member Function/Subroutine Documentation

C.29.2.1 conv_1d2nd_1d()

```
subroutine pb3d_utilities::conv_1d2nd::conv_1d2nd_1d (
               type(var_1d_type), intent(in) var_in,
               real(dp), dimension(:), intent(inout), allocatable var_out )
```

1-D version

Parameters

in	*var_in*	1D variable
in,out	*var_out*	output variable

Definition at line 39 of file PB3D_utilities.f90.

C.29.2.2 conv_1d2nd_2d()

```
subroutine pb3d_utilities::conv_1d2nd::conv_1d2nd_2d (
               type(var_1d_type), intent(in) var_in,
               real(dp), dimension(:,:), intent(inout), allocatable var_out )
```

2-D version

Parameters

in	*var_in*	1D variable
in,out	*var_out*	output variable

Definition at line 50 of file PB3D_utilities.f90.

C.29.2.3 conv_1d2nd_3d()

```
subroutine pb3d_utilities::conv_1d2nd::conv_1d2nd_3d (
               type(var_1d_type), intent(in) var_in,
               real(dp), dimension(:,:,:), intent(inout), allocatable var_out )
```

3-D version

Parameters

in	*var_in*	1D variable
in,out	*var_out*	output variable

Definition at line 65 of file PB3D_utilities.f90.

C.29.2.4 conv_1d2nd_4d()

```
subroutine pb3d_utilities::conv_1d2nd::conv_1d2nd_4d (
               type(var_1d_type), intent(in) var_in,
               real(dp), dimension(:,:,:,:), intent(inout), allocatable var_out )
```

4-D version

Parameters

in	*var_in*	1D variable
in,out	*var_out*	output variable

Definition at line 79 of file PB3D_utilities.f90.

C.29.2.5 conv_1d2nd_6d()

```
subroutine pb3d_utilities::conv_1d2nd::conv_1d2nd_6d (
```

```
type(var_1d_type), intent(in) var_in,
real(dp), dimension(:,:,:,:,:,:), intent(inout), allocatable var_out )
```

6-D version

Parameters

in	*var_in*	1D variable
in,out	*var_out*	output variable

Definition at line 94 of file PB3D_utilities.f90.

C.29.2.6 conv_1d2nd_7d()

```
subroutine pb3d_utilities::conv_1d2nd::conv_1d2nd_7d (
type(var_1d_type), intent(in) var_in,
real(dp), dimension(:,:,:,:,:,:,:), intent(inout), allocatable var_out )
```

7-D version

Parameters

in	*var_in*	1D variable
in,out	*var_out*	output variable

Definition at line 111 of file PB3D_utilities.f90.

C.30 num_utilities::conv_mat Interface Reference

Converts a (symmetric) matrix A with the storage convention described in eq_vars.eq_2_type.

Public Member Functions

- integer function conv_mat_3d (A, B, n, transp)
  version defined on 3-D grid
- integer function conv_mat_3d_complex (A, B, n, transp)
  complex version defined on 3D grid
- integer function conv_mat_0d (A, B, n, transp)
  scalar version
- integer function conv_mat_0d_complex (A, B, n, transp)
  complex scalar version

C.30.1 Detailed Description

Converts a (symmetric) matrix A with the storage convention described in eq_vars.eq_2_type.

This matric can have elements depending on a 3-D grid or be constant.

- If the matrix is stored with n^2 numbers, only the lower diagonal elements are kept in matrix B.
- If only the lower diagonal elements are stored, they are copied to the upper diagonal ones of the matrix B as well.

Optionally, the transpose can be calculated.

Note

1. The routine does not check whether the matrix is indeed symmetric and that information may thus be lost after conversion.
2. This routine makes a copy of A so that by providing A as input argument for both A and B overwrites A.

Returns

ierr

Definition at line 123 of file num_utilities.f90.

C.30.2 Member Function/Subroutine Documentation

C.30.2.1 conv_mat_0d()

```
integer function num_utilities::conv_mat::conv_mat_0d (
               real(dp), dimension(:), intent(in) A,
               real(dp), dimension(:), intent(inout) B,
               integer, intent(in) n,
               logical, intent(in), optional transp )
```

scalar version

Parameters

in	*a*	matrix to be converted
in,out	*b*	converted matrix
in	*n*	size of matrix
in	*transp*	transpose

Definition at line 1084 of file num_utilities.f90.

C.30.2.2 conv_mat_0d_complex()

```
integer function num_utilities::conv_mat::conv_mat_0d_complex (
               complex(dp), dimension(:), intent(in) A,
               complex(dp), dimension(:), intent(inout) B,
               integer, intent(in) n,
               logical, intent(in), optional transp )
```

complex scalar version

Parameters

in	*a*	matrix to be converted
in,out	*b*	converted matrix
in	*n*	size of matrix
in	*transp*	transpose

Definition at line 1179 of file num_utilities.f90.

C.30.2.3 conv_mat_3d()

```
integer function num_utilities::conv_mat::conv_mat_3d (
               real(dp), dimension(:,:,:,:), intent(in) A,
               real(dp), dimension(:,:,:,:), intent(inout) B,
               integer, intent(in) n,
               logical, intent(in), optional transp )
```

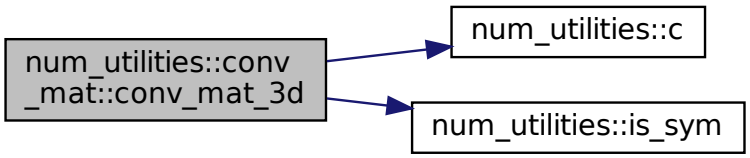
version defined on 3-D grid

Parameters

in	*a*	matrix to be converted
in,out	*b*	converted matrix
in	*n*	size of matrix
in	*transp*	transpose

Definition at line 1015 of file num_utilities.f90.

Here is the call graph for this function:



C.30.2.4 conv_mat_3d_complex()

```
integer function num_utilities::conv_mat::conv_mat_3d_complex (
               complex(dp), dimension(:,:,:,:), intent(in) A,
               complex(dp), dimension(:,:,:,:), intent(inout) B,
               integer, intent(in) n,
               logical, intent(in), optional transp )
```

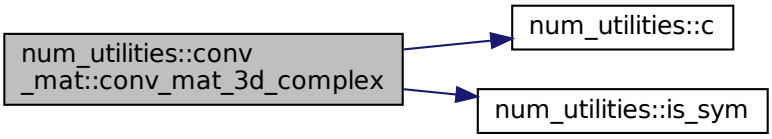
complex version defined on 3D grid

Parameters

in	*a*	matrix to be converted
in,out	*b*	converted matrix
in	*n*	size of matrix
in	*transp*	transpose

Definition at line 1110 of file num_utilities.f90.

Here is the call graph for this function:



C.31 grid_utilities::coord_e2f Interface Reference

Converts Equilibrium coordinates $(r,\theta,\zeta)_{\mathrm{E}}$. to Flux coordinates $(r,\theta,\zeta)_{\mathrm{F}}$.

Public Member Functions

- integer function coord_e2f_r (grid_eq, r_E, r_F, r_E_array, r_F_array)
  version with only r
- integer function coord_e2f_rtz (grid_eq, r_E, theta_E, zeta_E, r_F, theta_F, zeta_F, r_E_array, r_F_array)
  version with r, theta and zeta

C.31.1 Detailed Description

Converts Equilibrium coordinates $(r,\theta,\zeta)_{\mathrm{E}}$. to Flux coordinates $(r,\theta,\zeta)_{\mathrm{F}}$.

Optionally, two arrays `r_E_array` and `r_F_array` can be provided, that define the mapping between the both coordinate system.

Standard, for E, the poloidal or toroidal normalized flux is used and for F, the poloidal or toroidal flux in F coordinates, divided by 2π.

Returns

ierr

Definition at line 66 of file grid_utilities.f90.

C.31.2 Member Function/Subroutine Documentation

C.31.2.1 coord_e2f_r()

```
integer function grid_utilities::coord_e2f::coord_e2f_r (
               type(grid_type), intent(in) grid_eq,
               real(dp), dimension(:), intent(in) r_E,
               real(dp), dimension(:), intent(inout) r_F,
               real(dp), dimension(:), intent(in), optional, target r_E_array,
               real(dp), dimension(:), intent(in), optional, target r_F_array )
```

version with only r

Parameters

in	*grid_eq*	equilibrium grid (for normal local limits)
in	*r_e*	(local) r_{E}
in,out	*r_f*	(local) r_{F}
in	*r_e_array*	optional array that defines mapping between two coord. systems
in	*r_f_array*	optional array that defines mapping between two coord. systems

Definition at line 473 of file grid_utilities.f90.

C.31.2.2 coord_e2f_rtz()

```
integer function grid_utilities::coord_e2f::coord_e2f_rtz (
               type(grid_type), intent(in) grid_eq,
               real(dp), dimension(:), intent(in) r_E,
               real(dp), dimension(:,:,:), intent(in) theta_E,
               real(dp), dimension(:,:,:), intent(in) zeta_E,
               real(dp), dimension(:), intent(inout) r_F,
               real(dp), dimension(:,:,:), intent(inout) theta_F,
               real(dp), dimension(:,:,:), intent(inout) zeta_F,
               real(dp), dimension(:), intent(in), optional, target r_E_array,
               real(dp), dimension(:), intent(in), optional, target r_F_array )
```

version with r, theta and zeta

Parameters

in	*grid_eq*	equilibrium grid (for normal local limits)
in	*r_e*	(local) r_E
in	*theta_e*	θ_E
in	*zeta_e*	ζ_E
in,out	*r_f*	(local) r_F
in,out	*theta_f*	θ_F
in,out	*zeta_f*	ζ_F
in	*r_e_array*	optional array that defines mapping between two coord. systems
in	*r_f_array*	optional array that defines mapping between two coord. systems

Definition at line 359 of file grid_utilities.f90.

C.32 grid_utilities::coord_f2e Interface Reference

Converts Flux coordinates $(r, \theta, \zeta)_F$ to Equilibrium coordinates $(r, \theta, \zeta)_E$.

Public Member Functions

- integer function coord_f2e_r (grid_eq, r_F, r_E, r_F_array, r_E_array)

  version with only r
- integer function coord_f2e_rtz (grid_eq, r_F, theta_F, zeta_F, r_E, theta_E, zeta_E, r_F_array, r_E_array, ord)

  version withi r, theta and zeta

C.32.1 Detailed Description

Converts Flux coordinates $(r, \theta, \zeta)_F$ to Equilibrium coordinates $(r, \theta, \zeta)_E$.

Optionally, two arrays `r_F_array` and `r_E_array` can be provided, that define the mapping between the both coordinate system.

Standard, for E, the poloidal or toroidal normalized flux is used and for F, the poloidal or toroidal flux in F coordinates, divided by 2π.

Note

For VMEC, it can be slow, as the zero of a non-linear expression must be sought. This is done currently using num_ops.calc_zero_hh().

Returns

ierr

Definition at line 47 of file grid_utilities.f90.

C.32.2 Member Function/Subroutine Documentation

C.32.2.1 coord_f2e_r()

```
integer function grid_utilities::coord_f2e::coord_f2e_r (
               type(grid_type), intent(in) grid_eq,
               real(dp), dimension(:), intent(in) r_F,
               real(dp), dimension(:), intent(inout) r_E,
               real(dp), dimension(:), intent(in), optional, target r_F_array,
               real(dp), dimension(:), intent(in), optional, target r_E_array )
```

version with only r

Parameters

in	*grid_eq*	equilibrium grid (for possibly loc_r)
in	*r_f*	(local) r_F
in,out	*r_e*	(local) r_E
in	*r_f_array*	optional array that defines mapping between two coord. systems
in	*r_e_array*	optional array that defines mapping between two coord. systems

Definition at line 303 of file grid_utilities.f90.

C.32.2.2 coord_f2e_rtz()

```
integer function grid_utilities::coord_f2e::coord_f2e_rtz (
                type(grid_type), intent(in) grid_eq,
                real(dp), dimension(:), intent(in) r_F,
                real(dp), dimension(:,:,:), intent(in) theta_F,
                real(dp), dimension(:,:,:), intent(in) zeta_F,
                real(dp), dimension(:), intent(inout) r_E,
                real(dp), dimension(:,:,:), intent(inout) theta_E,
                real(dp), dimension(:,:,:), intent(inout) zeta_E,
                real(dp), dimension(:), intent(in), optional, target r_F_array,
                real(dp), dimension(:), intent(in), optional, target r_E_array,
                integer, intent(in), optional ord )
```

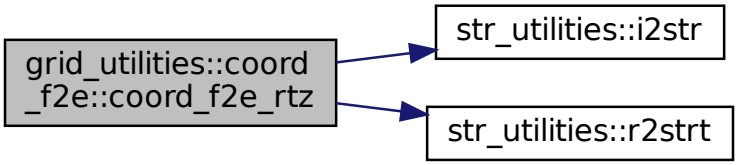
version withi r, theta and zeta

Parameters

in	*grid_eq*	equilibrium grid (for normal local limits and possibly loc_r)
in	*r_f*	(local) r_F
in	*theta_f*	θ_F
in	*zeta_f*	ζ_F
in,out	*r_e*	(local) r_E
in,out	*theta_e*	θ_E
in,out	*zeta_e*	ζ_E
in	*r_f_array*	optional array that defines mapping between two coord. systems
in	*r_e_array*	optional array that defines mapping between two coord. systems
in	*ord*	order (only used for eq_style 1)

Definition at line 120 of file grid_utilities.f90.

Here is the call graph for this function:



C.33 hdf5_vars::dealloc_var_1d Interface Reference

Deallocates 1D variable.

Public Member Functions

- subroutine dealloc_var_1d_ind (var_1D)
  individual version
- subroutine dealloc_var_1d_arr (var_1D)
  array version
- subroutine dealloc_var_1d_arr_2 (var_1D)
  rank 2 array version

C.33.1 Detailed Description

Deallocates 1D variable.

Definition at line 68 of file HDF5_vars.f90.

C.33.2 Member Function/Subroutine Documentation

C.33.2.1 dealloc_var_1d_arr()

```
subroutine hdf5_vars::dealloc_var_1d::dealloc_var_1d_arr (
               type(var_1d_type), dimension(:), intent(inout), allocatable var_1D )
```

array version

Parameters

in,out	*var_1d*	array of 1D variables to be deallocated

Definition at line 154 of file HDF5_vars.f90.

C.33.2.2 dealloc_var_1d_arr_2()

```
subroutine hdf5_vars::dealloc_var_1d::dealloc_var_1d_arr_2 (
               type(var_1d_type), dimension(:,:), intent(inout), allocatable var_1D )
```

rank 2 array version

Parameters

in,out	*var_1d*	array of 1D variables to be deallocated

Definition at line 136 of file HDF5_vars.f90.

C.33.2.3 dealloc_var_1d_ind()

```
subroutine hdf5_vars::dealloc_var_1d::dealloc_var_1d_ind (
               type(var_1d_type), intent(out) var_1D )
```

individual version

Parameters

out	*var_1d*	1D variable to be deallocated

Definition at line 170 of file HDF5_vars.f90.

C.34 hdf5_vars::dealloc_xml_str Interface Reference

Deallocates XML_str_type.

Public Member Functions

- subroutine dealloc_xml_str_ind (XML_str)
  individual version
- subroutine dealloc_xml_str_arr (XML_str)
  array version

C.34.1 Detailed Description

Deallocates XML_str_type.

Definition at line 60 of file HDF5_vars.f90.

C.34.2 Member Function/Subroutine Documentation

C.34.2.1 dealloc_xml_str_arr()

```
subroutine hdf5_vars::dealloc_xml_str::dealloc_xml_str_arr (
               type(xml_str_type), dimension(:), intent(inout), allocatable XML_str )
```

array version

Definition at line 114 of file HDF5_vars.f90.

C.34.2.2 dealloc_xml_str_ind()

```
subroutine hdf5_vars::dealloc_xml_str::dealloc_xml_str_ind (
               type(xml_str_type), intent(out) XML_str )
```

individual version

Parameters

out	*xml_str*	XML string to be deallocated

Definition at line 130 of file HDF5_vars.f90.

C.35 num_utilities::dis2con Interface Reference

Convert between points from a discrete grid to a continuous grid.

Public Member Functions

- integer function dis2con_eqd (pt_d, pt_c, lim_d, lim_c)
  equidistant version
- integer function dis2con_reg (pt_d, pt_c, var_c)
  regular version

C.35.1 Detailed Description

Convert between points from a discrete grid to a continuous grid.

This is done by providing either the the limits on the grid (`lim_c` and `lim_d`), in which case the grid is assumed to be equidistant, or the grid values themselves, in which case the grid just has to be regular.

The output is a real value. If the discrete value lies outside the range, in the case of a regular grid, a negative value is outputted, as well as a message.

Returns

ierr

Definition at line 206 of file num_utilities.f90.

C.35.2 Member Function/Subroutine Documentation

C.35.2.1 dis2con_eqd()

```
integer function num_utilities::dis2con::dis2con_eqd (
               integer, intent(in) pt_d,
               real(dp), intent(inout) pt_c,
               integer, dimension(2), intent(in) lim_d,
               real(dp), dimension(2), intent(in) lim_c )
```

equidistant version

Parameters

in	*pt_d*	point on discrete grid
in,out	*pt_c*	point on continous grid
in	*lim_d*	[min_d,max_d]
in	*lim_c*	[min_c,max_c]

Definition at line 1336 of file num_utilities.f90.

C.35.2.2 dis2con_reg()

```
integer function num_utilities::dis2con::dis2con_reg (
               integer, intent(in) pt_d,
               real(dp), intent(inout) pt_c,
               real(dp), dimension(:), intent(in) var_c )
```

regular version

Parameters

in	*pt_d*	point on discrete grid
in,out	*pt_c*	point on continous grid
in	*var_c*	continous grid values

Definition at line 1356 of file num_utilities.f90.

C.36 eq_vars::eq_1_type Type Reference

flux equilibrium type

Public Member Functions

- procedure init => init_eq_1
  initialize
- procedure copy => copy_eq_1
  copy
- procedure dealloc => dealloc_eq_1
  deallocate

Public Attributes

- real(dp), dimension(:,:), allocatable pres_e
  pressure, and norm. deriv.
- real(dp), dimension(:,:), allocatable q_saf_e
  safety factor
- real(dp), dimension(:,:), allocatable rot_t_e
  rot. transform
- real(dp), dimension(:,:), allocatable flux_p_e
  poloidal flux and norm. deriv.
- real(dp), dimension(:,:), allocatable flux_t_e
  toroidal flux and norm. deriv.
- real(dp), dimension(:,:), allocatable pres_fd
  pressure, and norm. deriv.
- real(dp), dimension(:,:), allocatable q_saf_fd
  safety factor
- real(dp), dimension(:,:), allocatable rot_t_fd
  rot. transform
- real(dp), dimension(:,:), allocatable flux_p_fd
  poloidal flux and norm. deriv.
- real(dp), dimension(:,:), allocatable flux_t_fd
  toroidal flux and norm. deriv.
- real(dp), dimension(:), allocatable rho
  density
- real(dp), dimension(2) estim_mem_usage
  expected memory usage

C.36.1 Detailed Description

flux equilibrium type

The arrays here are of the form

- `(r)` for variables without derivs.
- `(r,Dr)` for variables with derivs.

Definition at line 63 of file eq_vars.f90.

C.36.2 Member Function/Subroutine Documentation

C.36.2.1 copy()

```
procedure eq_vars::eq_1_type::copy
```

copy

Definition at line 82 of file eq_vars.f90.

C.36.2.2 dealloc()

```
procedure eq_vars::eq_1_type::dealloc
```

deallocate

Definition at line 84 of file eq_vars.f90.

C.36.2.3 init()

```
procedure eq_vars::eq_1_type::init
```

initialize

Definition at line 80 of file eq_vars.f90.

C.36.3 Member Data Documentation

C.36.3.1 estim_mem_usage

```
real(dp), dimension(2) eq_vars::eq_1_type::estim_mem_usage
```

expected memory usage

Note

Debug version only

Definition at line 76 of file eq_vars.f90.

C.36.3.2 flux_p_e

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::flux_p_e
```

poloidal flux and norm. deriv.

Definition at line 67 of file eq_vars.f90.

C.36.3.3 flux_p_fd

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::flux_p_fd
```

poloidal flux and norm. deriv.

Definition at line 72 of file eq_vars.f90.

C.36.3.4 flux_t_e

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::flux_t_e
```

toroidal flux and norm. deriv.

Definition at line 68 of file eq_vars.f90.

C.36.3.5 flux_t_fd

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::flux_t_fd
```

toroidal flux and norm. deriv.

Definition at line 73 of file eq_vars.f90.

C.36.3.6 pres_e

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::pres_e
```

pressure, and norm. deriv.

Definition at line 64 of file eq_vars.f90.

C.36.3.7 pres_fd

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::pres_fd
```

pressure, and norm. deriv.

Definition at line 69 of file eq_vars.f90.

C.36.3.8 q_saf_e

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::q_saf_e
```

safety factor

Definition at line 65 of file eq_vars.f90.

C.36.3.9 q_saf_fd

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::q_saf_fd
```

safety factor

Definition at line 70 of file eq_vars.f90.

C.36.3.10 rho

```
real(dp), dimension(:), allocatable eq_vars::eq_1_type::rho
```

density

Definition at line 74 of file eq_vars.f90.

C.36.3.11 rot_t_e

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::rot_t_e
```

rot. transform

Definition at line 66 of file eq_vars.f90.

C.36.3.12 rot_t_fd

```
real(dp), dimension(:,:), allocatable eq_vars::eq_1_type::rot_t_fd
```

rot. transform

Definition at line 71 of file eq_vars.f90.

C.37 eq_vars::eq_2_type Type Reference

metric equilibrium type

Public Member Functions

- procedure init => init_eq_2
  initialize
- procedure copy => copy_eq_2
  copy
- procedure dealloc => dealloc_eq_2
  deallocate

Public Attributes

- real(dp), dimension(:,:,:,:,:,:), allocatable r_e
  R in E(quilibrium) coord.
- real(dp), dimension(:,:,:,:,:,:), allocatable z_e
  Z in E(quilibrium) coords.
- real(dp), dimension(:,:,:,:,:,:), allocatable l_e
  L(ambda) in E(quilibrium) coords.
- real(dp), dimension(:,:,:,:,:,:,:), allocatable g_c
  in the C(ylindrical) coord. system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable g_e
  in the E(quilibrium) coord. system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable h_e
  in the E(quilibrium) coord. system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable g_f
  in the F(lux) coord. system with derivs. in V(MEC) system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable h_f
  in the F(lux) coord. system with derivs. in V(MEC) system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable g_fd
  in the F(lux) coord. system with derivs in F(lux) system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable h_fd
  in the F(lux) coord. system with derivs in F(lux) system
- real(dp), dimension(:,:,:,:,:,:,:), allocatable t_vc
  C(ylindrical) to V(MEC)
- real(dp), dimension(:,:,:,:,:,:,:), allocatable t_ef

E(quilibrium) to F(lux)

- real(dp), dimension(:,:,:,:,:,:,:), allocatable t_fe
  F(lux) to E(quilibrium)
- real(dp), dimension(:,:,:,:,:,:), allocatable det_t_ef
  determinant of T_EF
- real(dp), dimension(:,:,:,:,:,:), allocatable det_t_fe
  determinant of T_FE
- real(dp), dimension(:,:,:,:,:,:), allocatable jac_e
  jacobian of E(quilibrium) coord. system
- real(dp), dimension(:,:,:,:,:,:), allocatable jac_f
  jacobian of F(lux) coord. system with derivs. in V(MEC) system
- real(dp), dimension(:,:,:,:,:,:), allocatable jac_fd
  jacobian of F(lux) coord. system with derivs. in F(lux) system
- real(dp), dimension(:,:,:), allocatable s
  magnetic shear
- real(dp), dimension(:,:,:), allocatable kappa_n
  normal curvature
- real(dp), dimension(:,:,:), allocatable kappa_g
  geodesic curvature
- real(dp), dimension(:,:,:), allocatable sigma
  parallel current
- real(dp), dimension(2) estim_mem_usage
  expected memory usage

C.37.1 Detailed Description

metric equilibrium type

The arrays here are of the form

- `(angle_1,angle_2,r)` for scalars without derivs.
- `(angle_1,angle_2,r,D123)` for scalars with derivs.
- `(angle_1,angle_2,r,6/9,D1,D2,D3)` for tensors with derivs.
  where it is refered to the discussion of the grid type for an explanation of the angles `angle_1` and `angle_2`.

The last index refers to the derivatives in coordinate 1, 2 and 3, which refer to the coordinates described in [15].

- For E(quilibrium) coordinates, they are $(r, \theta, \zeta)_{\mathsf{E}}$.
- For F(lux) coordinates, they are $(\alpha, \psi, \gamma)_{\mathsf{F}}$, where
  - $\alpha = \zeta - q\theta$ and $\gamma = \theta$ for pol. flux,
  - $\alpha = -\theta + \iota\zeta$ and $\gamma = \zeta$ Nfor tor. flux.

The fourth index for tensorial variables correspond to the 9 or 6 (symmetric) different values:

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \text{ or } \begin{pmatrix} 1 & & \\ 2 & 4 & \\ 3 & 5 & 6 \end{pmatrix}$$

Note

Fortran only allows for 7 dimensions in arrays.

Definition at line 114 of file eq_vars.f90.

C.37.2 Member Function/Subroutine Documentation

C.37.2.1 copy()

```
procedure eq_vars::eq_2_type::copy
```

copy

Definition at line 150 of file eq_vars.f90.

C.37.2.2 dealloc()

```
procedure eq_vars::eq_2_type::dealloc
```

deallocate

Definition at line 152 of file eq_vars.f90.

C.37.2.3 init()

```
procedure eq_vars::eq_2_type::init
```

initialize

Definition at line 148 of file eq_vars.f90.

C.37.3 Member Data Documentation

C.37.3.1 det_t_ef

```
real(dp), dimension(:,:,:,:,:), allocatable eq_vars::eq_2_type::det_t_ef
```

determinant of T_EF

Definition at line 132 of file eq_vars.f90.

C.37.3.2 det_t_fe

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::det_t_fe
```

determinant of T_FE

Definition at line 133 of file eq_vars.f90.

C.37.3.3 estim_mem_usage

```
real(dp), dimension(2) eq_vars::eq_2_type::estim_mem_usage
```

expected memory usage

Note

Debug version only

Definition at line 144 of file eq_vars.f90.

C.37.3.4 g_c

```
real(dp), dimension(:,:,:,:,:,:,:), allocatable eq_vars::eq_2_type::g_c
```

in the C(ylindrical) coord. system

Definition at line 120 of file eq_vars.f90.

C.37.3.5 g_e

```
real(dp), dimension(:,:,:,:,:,:,:), allocatable eq_vars::eq_2_type::g_e
```

in the E(quilibrium) coord. system

Definition at line 121 of file eq_vars.f90.

C.37.3.6 g_f

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::g_f
```

in the F(lux) coord. system with derivs. in V(MEC) system

Definition at line 123 of file eq_vars.f90.

C.37.3.7 g_fd

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::g_fd
```

in the F(lux) coord. system with derivs in F(lux) system

Definition at line 125 of file eq_vars.f90.

C.37.3.8 h_e

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::h_e
```

in the E(quilibrium) coord. system

Definition at line 122 of file eq_vars.f90.

C.37.3.9 h_f

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::h_f
```

in the F(lux) coord. system with derivs. in V(MEC) system

Definition at line 124 of file eq_vars.f90.

C.37.3.10 h_fd

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::h_fd
```

in the F(lux) coord. system with derivs in F(lux) system

Definition at line 126 of file eq_vars.f90.

C.37.3.11 jac_e

```
real(dp), dimension(:,:,:,:,:), allocatable eq_vars::eq_2_type::jac_e
```

jacobian of E(quilibrium) coord. system

Definition at line 135 of file eq_vars.f90.

C.37.3.12 jac_f

```
real(dp), dimension(:,:,:,:,:), allocatable eq_vars::eq_2_type::jac_f
```

jacobian of F(lux) coord. system with derivs. in V(MEC) system

Definition at line 136 of file eq_vars.f90.

C.37.3.13 jac_fd

```
real(dp), dimension(:,:,:,:,:), allocatable eq_vars::eq_2_type::jac_fd
```

jacobian of F(lux) coord. system with derivs. in F(lux) system

Definition at line 137 of file eq_vars.f90.

C.37.3.14 kappa_g

```
real(dp), dimension(:,:,:), allocatable eq_vars::eq_2_type::kappa_g
```

geodesic curvature

Definition at line 141 of file eq_vars.f90.

C.37.3.15 kappa_n

```
real(dp), dimension(:,:,:), allocatable eq_vars::eq_2_type::kappa_n
```

normal curvature

Definition at line 140 of file eq_vars.f90.

C.37.3.16 l_e

```
real(dp), dimension(:,:,:,:,:), allocatable eq_vars::eq_2_type::l_e
```

L(ambda) in E(quilibrium) coords.

Definition at line 118 of file eq_vars.f90.

C.37.3.17 r_e

```
real(dp), dimension(:,:,:,:,:), allocatable eq_vars::eq_2_type::r_e
```

R in E(quilibrium) coord.

Definition at line 116 of file eq_vars.f90.

C.37.3.18 s

```
real(dp), dimension(:,:,:), allocatable eq_vars::eq_2_type::s
```

magnetic shear

Definition at line 139 of file eq_vars.f90.

C.37.3.19 sigma

```
real(dp), dimension(:,:,:), allocatable eq_vars::eq_2_type::sigma
```

parallel current

Definition at line 142 of file eq_vars.f90.

C.37.3.20 t_ef

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::t_ef
```

E(quilibrium) to F(lux)

Definition at line 129 of file eq_vars.f90.

C.37.3.21 t_fe

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::t_fe
```

F(lux) to E(quilibrium)

Definition at line 130 of file eq_vars.f90.

C.37.3.22 t_vc

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::t_vc
```

C(ylindrical) to V(MEC)

Definition at line 128 of file eq_vars.f90.

C.37.3.23 z_e

```
real(dp), dimension(:,:,:,:,:,:), allocatable eq_vars::eq_2_type::z_e
```

Z in E(quilibrium) coords.

Definition at line 117 of file eq_vars.f90.

C.38 vmec_utilities::fourier2real Interface Reference

Inverse Fourier transformation, from VMEC.

Public Member Functions

- integer function fourier2real_1 (varf_c, varf_s, trigon_factors, varr, sym, deriv)
  version with trigonometric factors
- integer function fourier2real_2 (varf_c, varf_s, theta, zeta, varr, sym, deriv)
  version with angles

C.38.1 Detailed Description

Inverse Fourier transformation, from VMEC.

Also calculates the poloidal or toroidal derivatives in VMEC coords., as indicated by the variable `deriv(2)`.

(The normal derivative is done on the variables in Fourier space, and should be provided here in `varf_i` if needed).

There are two variants:

1. version using trigon_factors, which is useful when the grid on which the trigonometric factors are defined is not regular and ideally when they are reused multiple times.
2. version using θ and ζ directly, which is useful for small, unique calculations.

Both these versions make use of a factor that represents angular derivatives. For `deriv = [j,k]`, this is:

- $m^j(-n)^k(-1)^{\frac{j+k+1}{2}}$ for `varf_c`,
- $m^j(-n)^k(-1)^{\frac{j+k}{2}}$ for `varf_s`,

where the divisions have to be done using integers, i.e. without remainder. The first two factors are straightforward, and the third one originates in the change of sign when deriving a cosine, but not for a sine.

Finally, depending on whether $(j+k)$ is even or odd, the correct $\cos$ or $\sin$ is chosen.

Returns

ierr

Definition at line 56 of file VMEC_utilities.f90.

C.38.2 Member Function/Subroutine Documentation

C.38.2.1 fourier2real_1()

```
integer function vmec_utilities::fourier2real::fourier2real_1 (
               real(dp), dimension(:,:), intent(in) varf_c,
               real(dp), dimension(:,:), intent(in) varf_s,
               real(dp), dimension(:,:,:,:,:), intent(in) trigon_factors,
               real(dp), dimension(:,:,:), intent(inout) varr,
               logical, dimension(2), intent(in), optional sym,
               integer, dimension(2), intent(in), optional deriv )
```

version with trigonometric factors

Parameters

in	*varf_c*	$\cos$ factor of variable in Fourier space
in	*varf_s*	$\sin$ factor of variable in Fourier space
in	*trigon_factors*	trigonometric factor cosine and sine at these angles (see calc_trigon_factors())
in,out	*varr*	variable in real space
in	*sym*	whether to use varf_c (1) and / or varf_s (2)
in	*deriv*	optional derivatives in angular coordinates

Definition at line 67 of file VMEC_utilities.f90.

C.38.2.2 fourier2real_2()

```
integer function vmec_utilities::fourier2real::fourier2real_2 (
               real(dp), dimension(:,:), intent(in) varf_c,
               real(dp), dimension(:,:), intent(in) varf_s,
               real(dp), dimension(:,:,:), intent(in) theta,
               real(dp), dimension(:,:,:), intent(in) zeta,
               real(dp), dimension(:,:,:), intent(inout) varr,
               logical, dimension(2), intent(in), optional sym,
               integer, dimension(2), intent(in), optional deriv )
```

version with angles

Parameters

in	*varf_c*	$\cos$ factor of variable in Fourier space
in	*varf_s*	$\sin$ factor of variable in Fourier space
in	*theta*	θ_E
in	*zeta*	ζ_E
in,out	*varr*	variable in real space
in	*sym*	whether to use varf_c (1) and / or varf_s (2)
in	*deriv*	optional derivatives in angular coordinates

Definition at line 165 of file VMEC_utilities.f90.

C.39 mpi_utilities::get_ghost_arr Interface Reference

Fill the ghost regions in an array.

Public Member Functions

- integer function get_ghost_arr_3d_complex (arr, size_ghost)
  3-D complex version
- integer function get_ghost_arr_3d_real (arr, size_ghost)
  3-D real version
- integer function get_ghost_arr_2d_complex (arr, size_ghost)
  2-D complex version
- integer function get_ghost_arr_1d_real (arr, size_ghost)
  1-D real version

C.39.1 Detailed Description

Fill the ghost regions in an array.

This is done by sending the first normal point of a process to the left process.

Every MPI message is identified by its sending process. The array should have the extended size, including ghost regions.

Returns

ierr

Definition at line 73 of file MPI_utilities.f90.

C.39.2 Member Function/Subroutine Documentation

C.39.2.1 get_ghost_arr_1d_real()

```
integer function mpi_utilities::get_ghost_arr::get_ghost_arr_1d_real (
               real(dp), dimension(:), intent(in) arr,
               integer, intent(in) size_ghost )
```

1-D real version

Parameters

in	*arr*	divided array
in	*size_ghost*	width of ghost region

Definition at line 555 of file MPI_utilities.f90.

C.39.2.2 get_ghost_arr_2d_complex()

```
integer function mpi_utilities::get_ghost_arr::get_ghost_arr_2d_complex (
               complex(dp), dimension(:,:), intent(inout) arr,
               integer, intent(in) size_ghost )
```

2-D complex version

Parameters

in,out	*arr*	divided array
in	*size_ghost*	width of ghost region

Definition at line 511 of file MPI_utilities.f90.

C.39.2.3 get_ghost_arr_3d_complex()

```
integer function mpi_utilities::get_ghost_arr::get_ghost_arr_3d_complex (
               complex(dp), dimension(:,:,:), intent(inout) arr,
               integer, intent(in) size_ghost )
```

3-D complex version

Parameters

in,out	*arr*	divided array
in	*size_ghost*	width of ghost region

Definition at line 422 of file MPI_utilities.f90.

C.39.2.4 get_ghost_arr_3d_real()

```
integer function mpi_utilities::get_ghost_arr::get_ghost_arr_3d_real (
               real(dp), dimension(:,:,:), intent(inout) arr,
               integer, intent(in) size_ghost )
```

3-D real version

Parameters

in,out	*arr*	divided array
in	*size_ghost*	width of ghost region

Definition at line 466 of file MPI_utilities.f90.

C.40 mpi_utilities::get_ser_var Interface Reference

Gather parallel variable in serial version on group master.

Public Member Functions

- integer function get_ser_var_complex (var, ser_var, scatter)
  complex version
- integer function get_ser_var_real (var, ser_var, scatter)
  real version
- integer function get_ser_var_int (var, ser_var, scatter)
  integer version

C.40.1 Detailed Description

Gather parallel variable in serial version on group master.

Optionally, all the processes receive the parallel variable using `scatter`.

Note

The serial variable has to be allocatable and if unallocated, it will be allocated.

Returns

ierr

Definition at line 55 of file MPI_utilities.f90.

C.40.2 Member Function/Subroutine Documentation

C.40.2.1 get_ser_var_complex()

```
integer function mpi_utilities::get_ser_var::get_ser_var_complex (
               complex(dp), dimension(:), intent(in) var,
               complex(dp), dimension(:), intent(inout), allocatable ser_var,
               logical, intent(in), optional scatter )
```

complex version

Parameters

in	*var*	parallel vector
in,out	*ser_var*	serial vector
in	*scatter*	optionally scatter the result to all the processes

Definition at line 107 of file MPI_utilities.f90.

C.40.2.2 get_ser_var_int()

```
integer function mpi_utilities::get_ser_var::get_ser_var_int (
               integer, dimension(:), intent(in) var,
               integer, dimension(:), intent(inout), allocatable ser_var,
               logical, intent(in), optional scatter )
```

integer version

Parameters

in	*var*	parallel vector
in,out	*ser_var*	serial vector
in	*scatter*	optionally scatter the result to all the processes

Definition at line 255 of file MPI_utilities.f90.

C.40.2.3 get_ser_var_real()

```
integer function mpi_utilities::get_ser_var::get_ser_var_real (
               real(dp), dimension(:), intent(in) var,
               real(dp), dimension(:), intent(inout), allocatable ser_var,
               logical, intent(in), optional scatter )
```

real version

Parameters

in	*var*	parallel vector
in,out	*ser_var*	serial vector
in	*scatter*	optionally scatter the result to all the processes

Definition at line 181 of file MPI_utilities.f90.

C.41 grid_vars::grid_type Type Reference

Type for grids.

Public Member Functions

- procedure init => init_grid
  initialize
- procedure copy => copy_grid
  copy
- procedure dealloc => dealloc_grid
  deallocate

Public Attributes

- integer, dimension(3) n
  tot nr. of points
- integer loc_n_r
  local nr. of normal points
- integer i_min
  min. normal index of this process in full arrays
- integer i_max
  max. normal index of this process in full arrays
- logical divided
  whether the grid is split up among the processes
- real(dp), dimension(:), pointer r_e => null()
  E(quilibrium) coord. values at n points.
- real(dp), dimension(:), pointer r_f => null()
  F(lux) coord. values at n points.
- real(dp), dimension(:), pointer loc_r_e => null()
  E(quilibrium) coord. values at n points.
- real(dp), dimension(:), pointer loc_r_f => null()
  F(lux) coord. values at n points.
- real(dp), dimension(:,:,:), pointer theta_e => null()
  E(quilibrium) coord. values of first angle at n points in 3-D.
- real(dp), dimension(:,:,:), pointer theta_f => null()
  F(lux) coord. values of first angle at n points in 3-D.
- real(dp), dimension(:,:,:), pointer zeta_e => null()
  E(quilibrium) coord. values of second angle at n points in 3-D.
- real(dp), dimension(:,:,:), pointer zeta_f => null()
  F(lux) coord. values of second angle at n points in 3-D.
- real(dp), dimension(:,:,:,:,:), allocatable trigon_factors
  trigonometric factor cosine for the inverse fourier transf.
- real(dp) estim_mem_usage
  estimated memory usage

C.41.1 Detailed Description

Type for grids.

The grids are saved in the following format: `(angle_1,angle_2,r)`, where `angle_1` and `angle_2` can be any angle that completely describe a flux surface.

For example, they can refer to a grid of θ and ζ values, but they can also refer to (Modified) flux coordinates with a parallel angle and a field line coordinate (α).

For field-aligned grids, `angle_1` is generally chosen equal to the parallel coordinate in the Flux coordinate system, and `angle_2` equal to the field line label (so the second dimension of the matrices is then chosen to be of size 1 for the calculations on a single field line).

At the same time, the parallel coordinate, in which integrations will have to be done, ocupies the first index. This is good for numerical efficiency.

For specific equilibrium grids, such as the case for HELENA equilibria, `angle_1` corresponds to θ and `angle_2` to the symmetry angle ζ.

It is important that this order of the coordinates in space is consistent among all the variables.

Definition at line 59 of file grid_vars.f90.

C.41.2 Member Function/Subroutine Documentation

C.41.2.1 copy()

```
procedure grid_vars::grid_type::copy
```

copy

Definition at line 81 of file grid_vars.f90.

C.41.2.2 dealloc()

```
procedure grid_vars::grid_type::dealloc
```

deallocate

Definition at line 83 of file grid_vars.f90.

C.41.2.3 init()

```
procedure grid_vars::grid_type::init
```

initialize

Definition at line 79 of file grid_vars.f90.

C.41.3 Member Data Documentation

C.41.3.1 divided

```
logical grid_vars::grid_type::divided
```

whether the grid is split up among the processes

Definition at line 64 of file grid_vars.f90.

C.41.3.2 estim_mem_usage

```
real(dp) grid_vars::grid_type::estim_mem_usage
```

estimated memory usage

Note

Debug version only

Definition at line 75 of file grid_vars.f90.

C.41.3.3 i_max

```
integer grid_vars::grid_type::i_max
```

max. normal index of this process in full arrays

Definition at line 63 of file grid_vars.f90.

C.41.3.4 i_min

```
integer grid_vars::grid_type::i_min
```

min. normal index of this process in full arrays

Definition at line 62 of file grid_vars.f90.

C.41.3.5 loc_n_r

```
integer grid_vars::grid_type::loc_n_r
```

local nr. of normal points

Definition at line 61 of file grid_vars.f90.

C.41.3.6 loc_r_e

```
real(dp), dimension(:), pointer grid_vars::grid_type::loc_r_e => null()
```

E(quilibrium) coord. values at n points.

Definition at line 67 of file grid_vars.f90.

C.41.3.7 loc_r_f

```
real(dp), dimension(:), pointer grid_vars::grid_type::loc_r_f => null()
```

F(lux) coord. values at n points.

Definition at line 68 of file grid_vars.f90.

C.41.3.8 n

```
integer, dimension(3) grid_vars::grid_type::n
```

tot nr. of points

Definition at line 60 of file grid_vars.f90.

C.41.3.9 r_e

```
real(dp), dimension(:), pointer grid_vars::grid_type::r_e => null()
```

E(quilibrium) coord. values at n points.

Definition at line 65 of file grid_vars.f90.

C.41.3.10 r_f

```
real(dp), dimension(:), pointer grid_vars::grid_type::r_f => null()
```

F(lux) coord. values at n points.

Definition at line 66 of file grid_vars.f90.

C.41.3.11 theta_e

```
real(dp), dimension(:,:,:), pointer grid_vars::grid_type::theta_e => null()
```

E(quilibrium) coord. values of first angle at n points in 3-D.

Definition at line 69 of file grid_vars.f90.

C.41.3.12 theta_f

```
real(dp), dimension(:,:,:), pointer grid_vars::grid_type::theta_f => null()
```

F(lux) coord. values of first angle at n points in 3-D.

Definition at line 70 of file grid_vars.f90.

C.41.3.13 trigon_factors

```
real(dp), dimension(:,:,:,:,:), allocatable grid_vars::grid_type::trigon_factors
```

trigonometric factor cosine for the inverse fourier transf.

Definition at line 73 of file grid_vars.f90.

C.41.3.14 zeta_e

```
real(dp), dimension(:,:,:), pointer grid_vars::grid_type::zeta_e => null()
```

E(quilibrium) coord. values of second angle at n points in 3-D.

Definition at line 71 of file grid_vars.f90.

C.41.3.15 zeta_f

```
real(dp), dimension(:,:,:), pointer grid_vars::grid_type::zeta_f => null()
```

F(lux) coord. values of second angle at n points in 3-D.

Definition at line 72 of file grid_vars.f90.

C.42 hdf5_vars::hdf5_file_type Type Reference

HDF5 data type, containing the information about HDF5 files.

Public Attributes

- integer(hid_t) hdf5_i
  HDF5 file handle.
- integer xdmf_i
  XDMF file handle.
- character(len=max_str_ln) name
  name of files (without extensions ".h5" and ".xmf")

C.42.1 Detailed Description

HDF5 data type, containing the information about HDF5 files.

Definition at line 40 of file HDF5_vars.f90.

C.42.2 Member Data Documentation

C.42.2.1 hdf5_i

```
integer(hid_t) hdf5_vars::hdf5_file_type::hdf5_i
```

HDF5 file handle.

Definition at line 41 of file HDF5_vars.f90.

C.42.2.2 name

```
character(len=max_str_ln) hdf5_vars::hdf5_file_type::name
```

name of files (without extensions ".h5" and ".xmf")

Definition at line 43 of file HDF5_vars.f90.

C.42.2.3 xdmf_i

```
integer hdf5_vars::hdf5_file_type::xdmf_i
```

XDMF file handle.

Definition at line 42 of file HDF5_vars.f90.

C.43 mpi_vars::lock_type Type Reference

lock type

Public Member Functions

- procedure init => init_lock
  initialize
- procedure dealloc => dealloc_lock
  deallocate

Public Attributes

- integer, dimension(:), allocatable wl
  waiting list
- integer wl_win
  window to waiting list
- integer wu_tag
  wakeup tag
- logical blocking
  is a normal blocking process

C.43.1 Detailed Description

lock type

There is a blocking (BL) and a nonblocking (NB) version where the former requires an exclusive lock and the latter a shared one. This is saved in the variable `blocking`.

NB processes that get the lock directly on request (meaning that there were no other processes in the queue) notify directly all the next NB processes after gaining access. It also sets their status to active. When a NB process gains the lock when notified after waiting, it does not have to check for other NB processes, as this has been done by the notifying process.

A BL process retains exclusive access upon receipt of the lock. Similarly to NB processes, if the receipt was direct on request, the status is set to active, but only of this NB process.

When returning the lock, all BL processes and NB that find themselves to be the last active NB process, scan the waiting list and pass the lock preferably to another BL process to notify. If not available, it searches for all the NB processes to notify together.

The advantage of prefering BL processes after finishing a process is that this way NB processes are accumulated, and then quickly finished afterwards.

Note

Every process in the waiting queue will eventually receive a notification.

Scheme:

	request access	gain acces	return access
BL	• add to queue • wait	if direct: • activate	• remove from queue • find next BL/NB • notify • activate all
NB	• add to queue • wait	if direct: • find next NB for all next NB: • notify • activate	always: • remove from queue if last NB: • find next BL/NB • notify • activate all

with preference BL $>$ NB(s).

See also

This is based on an extension of the ideas in `RossAtomicIO`.

Definition at line 63 of file MPI_vars.f90.

C.43.2 Member Function/Subroutine Documentation

C.43.2.1 dealloc()

```
procedure mpi_vars::lock_type::dealloc
```

deallocate

deallocate

Definition at line 72 of file MPI_vars.f90.

C.43.2.2 init()

```
procedure mpi_vars::lock_type::init
```

initialize

initialize

Definition at line 70 of file MPI_vars.f90.

C.43.3 Member Data Documentation

C.43.3.1 blocking

```
logical mpi_vars::lock_type::blocking
```

is a normal blocking process

Definition at line 67 of file MPI_vars.f90.

C.43.3.2 wl

```
integer, dimension(:), allocatable mpi_vars::lock_type::wl
```

waiting list

Definition at line 64 of file MPI_vars.f90.

C.43.3.3 wl_win

```
integer mpi_vars::lock_type::wl_win
```

window to waiting list

Definition at line 65 of file MPI_vars.f90.

C.43.3.4 wu_tag

```
integer mpi_vars::lock_type::wu_tag
```

wakeup tag

Definition at line 66 of file MPI_vars.f90.

C.44 x_vars::modes_type Type Reference

mode number type

Public Member Functions

- procedure dealloc => dealloc_mds
  deallocate

Public Attributes

- integer, dimension(:,:), allocatable n
  n *for all modes, in total grid*
- integer, dimension(:,:), allocatable m
  m *for all modes, in total grid*
- integer, dimension(:,:), allocatable sec
  m *or* n *for all possible modes, index and limits, in total grid*

C.44.1 Detailed Description

mode number type

Type containing information about mode numbers at every flux surface:

- mode numbers: `n`, `m`,
- information of secondary modes: `sec`,
  - 1: mode number
  - 2: lower limit of normal range
  - 3: upper limit of normal range
  - 4: index in tables where the number of modes can be bigger than the range if the safety factor or rotational transform is non-monotonous so that the same mode number can have multiple ranges.
- Flux normal coordinate: `r_F`.

These are set up in setup_nm_x().

Definition at line 36 of file X_vars.f90.

C.44.2 Member Function/Subroutine Documentation

C.44.2.1 dealloc()

```
procedure x_vars::modes_type::dealloc
```

deallocate

Definition at line 42 of file X_vars.f90.

C.44.3 Member Data Documentation

C.44.3.1 m

```
integer, dimension(:,:), allocatable x_vars::modes_type::m
```

m for all modes, in total grid

Definition at line 38 of file X_vars.f90.

C.44.3.2 n

```
integer, dimension(:,:), allocatable x_vars::modes_type::n
```

n for all modes, in total grid

Definition at line 37 of file X_vars.f90.

C.44.3.3 sec

```
integer, dimension(:,:), allocatable x_vars::modes_type::sec
```

m or n for all possible modes, index and limits, in total grid

Definition at line 39 of file X_vars.f90.

C.45 num_utilities::order_per_fun Interface Reference

Order a periodic function to include $0 \ldots 2\pi$ and an overlap.

Public Member Functions

- integer function order_per_fun_1 (x, y, x_out, y_out, overlap, tol)
  version with x and y separate
- integer function order_per_fun_2 (xy, xy_out, overlap, tol)
  version with x and y together

C.45.1 Detailed Description

Order a periodic function to include $0 \ldots 2\pi$ and an overlap.

Returns

ierr

Definition at line 248 of file num_utilities.f90.

C.45.2 Member Function/Subroutine Documentation

C.45.2.1 order_per_fun_1()

```
integer function num_utilities::order_per_fun::order_per_fun_1 (
               real(dp), dimension(:), intent(in) x,
               real(dp), dimension(:), intent(in) y,
               real(dp), dimension(:), intent(inout), allocatable x_out,
               real(dp), dimension(:), intent(inout), allocatable y_out,
               integer, intent(in) overlap,
               real(dp), intent(in), optional tol )
```

version with x and y separate

Parameters

in	*x*	abscissa
in	*y*	ordinate
in,out	*x_out*	ordered x
in,out	*y_out*	ordered y
in	*overlap*	overlap to include
in	*tol*	tolerance for error

Definition at line 1574 of file num_utilities.f90.

C.45.2.2 order_per_fun_2()

```
integer function num_utilities::order_per_fun::order_per_fun_2 (
               real(dp), dimension(:,:), intent(in) xy,
               real(dp), dimension(:,:), intent(inout), allocatable xy_out,
               integer, intent(in) overlap,
               real(dp), intent(in), optional tol )
```

version with x and y together

Parameters

in	*xy*	abscissa and ordinate
in,out	*xy_out*	ordered xy
in	*overlap*	overlap to include
in	*tol*	tolerance for error

Definition at line 1650 of file num_utilities.f90.

C.46 output_ops::plot_hdf5 Interface Reference

Prints variables `vars` with names `var_names` in an HDF5 file with name c file_name and accompanying X↩ DMF file.

Public Member Functions

- subroutine plot_hdf5_ind (var_name, file_name, var, tot_dim, loc_offset, X, Y, Z, cont_plot, descr)

  individual version
- subroutine plot_hdf5_arr (var_names, file_name, vars, tot_dim, loc_offset, X, Y, Z, col_id, col, sym_type, cont_plot, descr)

  array version

C.46.1 Detailed Description

Prints variables `vars` with names `var_names` in an HDF5 file with name c file_name and accompanying X↩ DMF file.

For XDMF collections ([18]), only the first value for `var_names` is used, so it should have a size of one.

The plot is generally 3-D, but if one of the dimensions provided is equal to 1, it is checked whether there is poloidal or toroidal axisymmetry and if so, the plot becomes 2-D. This can be forced using the optional input argument `sym_type`.

Optionally, the (curvilinear) grid can be provided through `X`, `Y` and `Z`. If not, the grid is assumed to be Cartesian with discrete indices where `X` corresponds to the first dimensions, `Y` to the second and `Z` to the third.

Additionally, the total grid size and local offset can be provided in `tot_dim` and `loc_offset` to run this routine in parallel automatically. Optionally, one of the dimensions (`col_id`, default 4) can be associated to a collection dimension using `col` different from 1:

- `col` = 1: no collection, just plots of different variables
- `col` = 2: time collection
- `col` = 3: spatial collection
- `col` = 4: vector field

Furthermore, using the variable `cont_plot`, a plot can be (over-)written in multiple writes. By this is meant that there should be an initial plot, with collection type 1, 2, 3 or 4, which can then be followed by an arbitrary number of additional writes. As these additional writes currently cannot modify the plot structure, nor the XDMF variables, their collection dimension should be complete from the start.

This has no implications for single plots but means that for collection types 2 to 4 all the elements in the collection have to be present, though they do not necessary need to have been completely written in the other dimensions.

Subsequent writes with `cont_plot` can then, for instance, write parts of the data that had not yet been written, or overwrite ones that had. This can be useful for post-processing where the memory requirements are large so that the work has to be split.

Note

1. For a vector field, the number of variables has to be 2 for 2-D plots or 3 for 3-D plots. This should be rewritten in the future, so that collections can be used for vectors as well.
2. In order to merge collections in their collection dimension, the XDMF files can always easily be joined.
3. If necessary, a lock system should be used when multiple processes are writing the same file, including continued writes.
4. To plot this with VisIt, use:
   - for temporal collections: pseudocolor using the variable name (other names are ignored).
   - for spatial collections: subset of blocks or pseudocolor using the variable name (other names are ignored).
   - for vector plot: Vector plot.
   - for without collections: pseudocolor using the variable names.
5. Currently all of possibly multiple processes that write data have to cover the entire range of the variables in the dimension indicated by `col_id`. (This could be implemented by changing how n↩ _plot is defined and selectively letting each processer write in the main loop at its corresponding indices.)
6. To project the data to 2-D in VisIt, use the projection tool under Operators > Transform

Definition at line 137 of file output_ops.f90.

C.46.2 Member Function/Subroutine Documentation

C.46.2.1 plot_hdf5_arr()

```
subroutine output_ops::plot_hdf5::plot_hdf5_arr (
               character(len=*), dimension(:), intent(in) var_names,
               character(len=*), intent(in) file_name,
               real(dp), dimension(:,:,:,:), intent(in), target vars,
               integer, dimension(4), intent(in), optional tot_dim,
               integer, dimension(4), intent(in), optional loc_offset,
               real(dp), dimension(:,:,:,:), intent(in), optional, target X,
               real(dp), dimension(:,:,:,:), intent(in), optional, target Y,
               real(dp), dimension(:,:,:,:), intent(in), optional, target Z,
               integer, intent(in), optional col_id,
               integer, intent(in), optional col,
               integer, intent(in), optional sym_type,
               logical, intent(in), optional cont_plot,
               character(len=*), intent(in), optional descr )
```

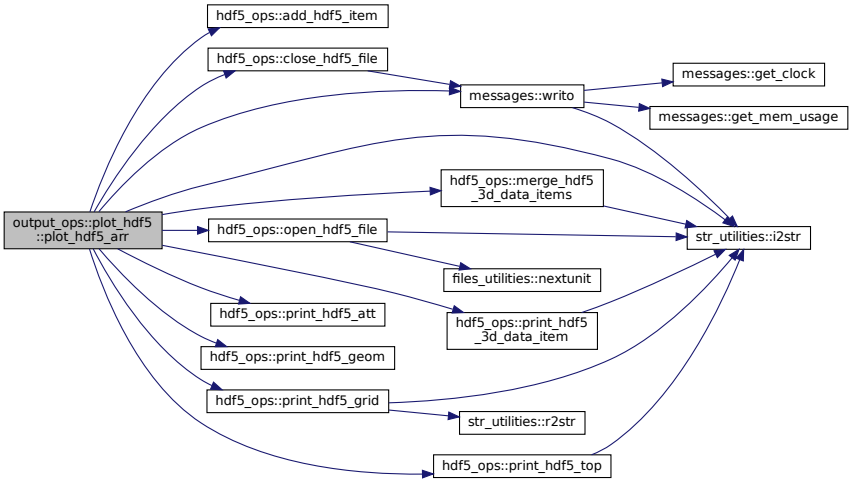
array version

Parameters

in	*var_names*	names of variable to be plot
in	*file_name*	file name
in	*vars*	variables to plot
in	*tot_dim*	total dimensions of the arrays
in	*loc_offset*	offset of local dimensions
in	*x*	curvilinear grid X points
in	*y*	curvilinear grid Y points
in	*z*	curvilinear grid Z points
in	*col_id*	index of time dimension
in	*col*	whether a collection is made
in	*sym_type*	type of symmetry (1: no symmetry, 2: toroidal, 3: poloidal)
in	*cont_plot*	continued plot
in	*descr*	description

Definition at line 435 of file output_ops.f90.

Here is the call graph for this function:



C.46.2.2 plot_hdf5_ind()

```
subroutine output_ops::plot_hdf5::plot_hdf5_ind (
               character(len=*), intent(in) var_name,
               character(len=*), intent(in) file_name,
               real(dp), dimension(:,:,:), intent(in) var,
               integer, dimension(3), intent(in), optional tot_dim,
               integer, dimension(3), intent(in), optional loc_offset,
               real(dp), dimension(:,:,:), intent(in), optional, target X,
               real(dp), dimension(:,:,:), intent(in), optional, target Y,
               real(dp), dimension(:,:,:), intent(in), optional, target Z,
               logical, intent(in), optional cont_plot,
               character(len=*), intent(in), optional descr )
```

individual version

Parameters

in	*var_name*	name of variable to be plot
in	*file_name*	file name
in	*var*	variable to plot
in	*tot_dim*	total dimensions of the arrays
in	*loc_offset*	offset of local dimensions
in	*x*	curvilinear grid X points
in	*y*	curvilinear grid Y points
in	*z*	curvilinear grid Z points
in	*cont_plot*	continued plot
in	*descr*	description

Definition at line 949 of file output_ops.f90.

C.47 output_ops::print_ex_2d Interface Reference

Print 2-D output on a file.

Public Member Functions

- subroutine print_ex_2d_ind (var_name, file_name_i, y, x, draw, persistent)
  individual version
- subroutine print_ex_2d_arr (var_names, file_name_i, y, x, draw, persistent)
  array version

C.47.1 Detailed Description

Print 2-D output on a file.

The variables `var_name` and `file_name` hold the name of the plot and of the file in which the plot data is to be saved, respectively. `y` is the an array containing the function which is stored and `x` is an optional vector with the x-values. The logical `draw` can optionally disable calling the external drawing procedure for output on screen [default], without modifying the plot file.

The first index of `y` (and `x`) contains the points of a current plot.

The second index indicates various plots (one or more)

Definition at line 47 of file output_ops.f90.

C.47.2 Member Function/Subroutine Documentation

C.47.2.1 print_ex_2d_arr()

```
subroutine output_ops::print_ex_2d::print_ex_2d_arr (
               character(len=*), dimension(:), intent(in) var_names,
               character(len=*), intent(in) file_name_i,
               real(dp), dimension(1:,1:), intent(in) y,
               real(dp), dimension(1:,1:), intent(in), optional x,
               logical, intent(in), optional draw,
               logical, intent(in), optional persistent )
```

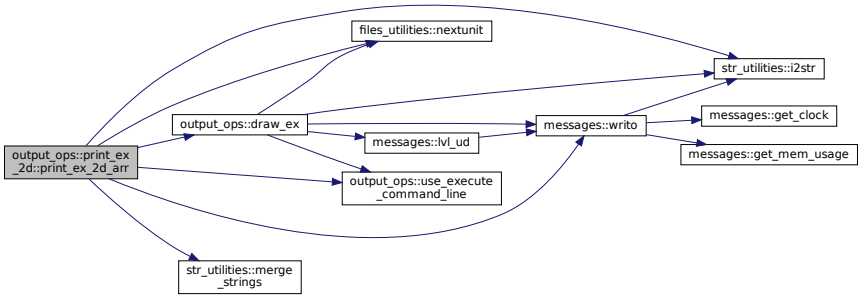
array version

Parameters

in	*var_names*	names of variables in legend
in	*file_↩ name_i*	name of input file
in	*y*	ordinate
in	*x*	absicca
in	*draw*	whether to draw the plot as well
in	*persistent*	keep on-screen plot open

Definition at line 172 of file output_ops.f90.

Here is the call graph for this function:



C.47.2.2 print_ex_2d_ind()

```
subroutine output_ops::print_ex_2d::print_ex_2d_ind (
               character(len=*), intent(in) var_name,
               character(len=*), intent(in) file_name_i,
               real(dp), dimension(1:), intent(in) y,
               real(dp), dimension(1:), intent(in), optional x,
               logical, intent(in), optional draw,
               logical, intent(in), optional persistent )
```

individual version

Parameters

in	*var_name*	name of variable in legend
in	*file_↩ name_i*	name of input file
in	*y*	ordinate
in	*x*	absicca
in	*draw*	whether to draw the plot as well
in	*persistent*	keep on-screen plot open

Definition at line 147 of file output_ops.f90.

C.48 output_ops::print_ex_3d Interface Reference

Print 3-D output on a file.

Public Member Functions

- subroutine print_ex_3d_ind (var_name, file_name_i, z, y, x, draw)
  individual version
- subroutine print_ex_3d_arr (var_names, file_name_i, z, x, y, draw)
  array version

C.48.1 Detailed Description

Print 3-D output on a file.

The variables `var_name` and `file_name` hold the name of the plot and of the file in which the plot data is to be saved, respectively. z is the an array containing the function which is stored and x andy are optional vectors with the x and y-values. The logical `draw` can optionally disable calling the external drawing procedure for output on screen [default], without modifying the plot file.

The first index of z (and x, y) contains the points of a current The plot second index indicates various plots (one or more)

Definition at line 65 of file output_ops.f90.

C.48.2 Member Function/Subroutine Documentation

C.48.2.1 print_ex_3d_arr()

```
subroutine output_ops::print_ex_3d::print_ex_3d_arr (
               character(len=*), dimension(:), intent(in) var_names,
               character(len=*), intent(in) file_name_i,
               real(dp), dimension(1:,1:,1:), intent(in) z,
               real(dp), dimension(1:,1:,1:), intent(in), optional x,
               real(dp), dimension(1:,1:,1:), intent(in), optional y,
               logical, intent(in), optional draw )
```

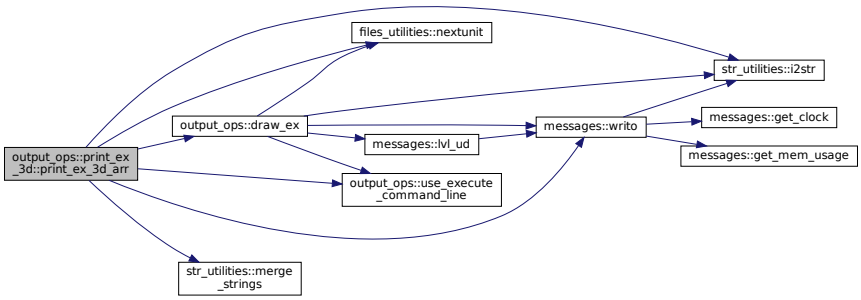
array version

Parameters

in	*var_names*	names of variables in legend
in	*file_↩ name_i*	name of input file
in	*z*	ordinate
in	*x*	absicca
in	*y*	absicca
in	*draw*	whether to draw the plot as well

Definition at line 310 of file output_ops.f90.

Here is the call graph for this function:



C.48.2.2 print_ex_3d_ind()

```
subroutine output_ops::print_ex_3d::print_ex_3d_ind (
               character(len=*), intent(in) var_name,
               character(len=*), intent(in) file_name_i,
               real(dp), dimension(1:,1:), intent(in) z,
               real(dp), dimension(1:,1:), intent(in), optional y,
               real(dp), dimension(1:,1:), intent(in), optional x,
               logical, intent(in), optional draw )
```

individual version

Parameters

in	*var_name*	name of variable in legend
in	*file_↩ name_i*	name of input file
in	*z*	ordinate
in	*x*	absicca
in	*y*	absicca
in	*draw*	whether to draw the plot as well

Definition at line 269 of file output_ops.f90.

C.49 eq_ops::print_output_eq Interface Reference

Print equilibrium quantities to an output file:

Public Member Functions

- integer function print_output_eq_1 (grid_eq, eq, data_name)
  flux version
- integer function print_output_eq_2 (grid_eq, eq, data_name, rich_lvl, par_div, dealloc_vars)
  metric version

C.49.1 Detailed Description

Print equilibrium quantities to an output file:

- flux:
  - `pres_FD`,
  - `q_saf_FD`,
  - `rot_t_FD`,
  - `flux_p_FD`,
  - `flux_t_FD`,
  - `rho`,
  - `S`,
  - `kappa_n`,
  - `kappa_g`,
  - `sigma`
- metric:
  - `g_FD`,
  - `h_FD`,
  - `jac_FD`

If `rich_lvl` is provided, `"_R_[rich_lvl]"` is appended to the data name if it is $>$`0` (only for `eq_2`).

Optionally, for `eq_2`, it can be specified that this is a divided parallel grid, corresponding to the variable `eq_jobs_lims` with index `eq_job_nr`. In this case, the total grid size is adjusted to the one specified by `eq_jobs_lims` and the grid is written as a subset.

Note

1. The equilibrium quantities are outputted in Flux coordinates.
2. The metric equilibrium quantities can be deallocated on the fly, which is useful if this routine is followed by a deallocation any way, so that memory usage does not almost double.
3. `print_output_eq_2` is only used by HELENA now, as for VMEC it is too slow since there are often multiple VMEC equilibrium jobs, while for HELENA this is explicitely forbidden.

Returns

ierr

Definition at line 90 of file eq_ops.f90.

C.49.2 Member Function/Subroutine Documentation

C.49.2.1 print_output_eq_1()

```
integer function eq_ops::print_output_eq::print_output_eq_1 (
               type(grid_type), intent(in) grid_eq,
               type(eq_1_type), intent(in) eq,
               character(len=*), intent(in) data_name )
```

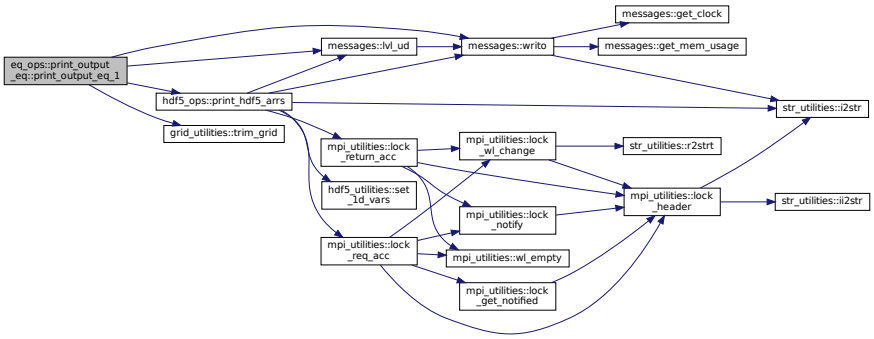
flux version

Parameters

in	*grid_eq*	equilibrium grid variables
in	*eq*	flux equilibrium variables
in	*data_name*	name under which to store

Definition at line 2275 of file eq_ops.f90.

Here is the call graph for this function:



C.49.2.2 print_output_eq_2()

```
integer function eq_ops::print_output_eq::print_output_eq_2 (
               type(grid_type), intent(in) grid_eq,
               type(eq_2_type), intent(inout) eq,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional par_div,
               logical, intent(in), optional dealloc_vars )
```

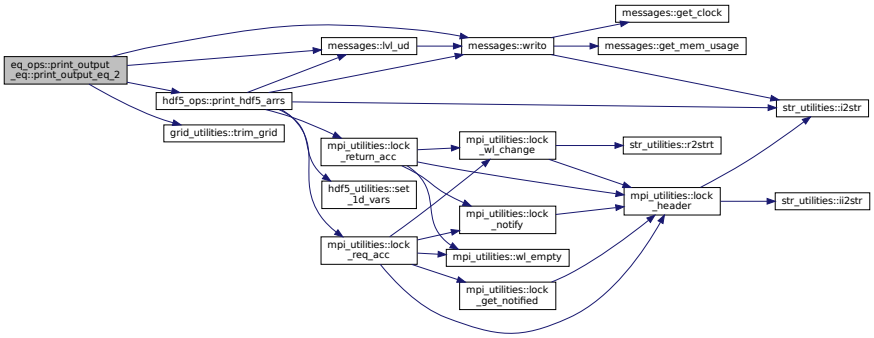
metric version

Parameters

in	*grid_eq*	equilibrium grid variables
in,out	*eq*	metric equilibrium variables
in	*data_name*	name under which to store
in	*rich_lvl*	Richardson level to print
in	*par_div*	is a parallely divided grid
in	*dealloc_vars*	deallocate variables on the fly after writing

Definition at line 2408 of file eq_ops.f90.

Here is the call graph for this function:



C.50 x_ops::print_output_x Interface Reference

Print either vectorial or tensorial perturbation quantities of a certain order to an output file.

Public Member Functions

- integer function print_output_x_1 (grid_X, X, data_name, rich_lvl, par_div, lim_sec_X)
  vectorial version.
- integer function print_output_x_2 (grid_X, X, data_name, rich_lvl, par_div, lim_sec_X, is_field_averaged)
  tensorial version

C.50.1 Detailed Description

Print either vectorial or tensorial perturbation quantities of a certain order to an output file.

- vectorial:
  - U

  - DU

- tensorial:
  - PV_int
  - KV_int

(The non-integrated variables are not saved because they are heavy and not requested.)

If `rich_lvl` is provided, `"_R_[rich_lvl]"` is appended to the data name if it is $>$`0`.

Optionally, it can be specified that this is a divided parallel grid, corresponding to the variable `eq_jobs_lims` with index `eq_job_nr`. In this case, the total grid size is adjusted to the one specified by `eq_jobs_lims` and the grid is written as a subset.

Note

1. The equilibrium quantities are outputted in Flux coordinates.
2. The tensorial perturbation type can also be used for field-aligned variables, in which case the first index is assumed to have dimension 1 only. This can be triggered using `is_field_averaged`.

Returns

ierr

Definition at line 85 of file X_ops.f90.

C.50.2 Member Function/Subroutine Documentation

C.50.2.1 print_output_x_1()

```
integer function x_ops::print_output_x::print_output_x_1 (
               type(grid_type), intent(in) grid_X,
               type(x_1_type), intent(in) X,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional par_div,
               integer, dimension(2), intent(in), optional lim_sec_X )
```

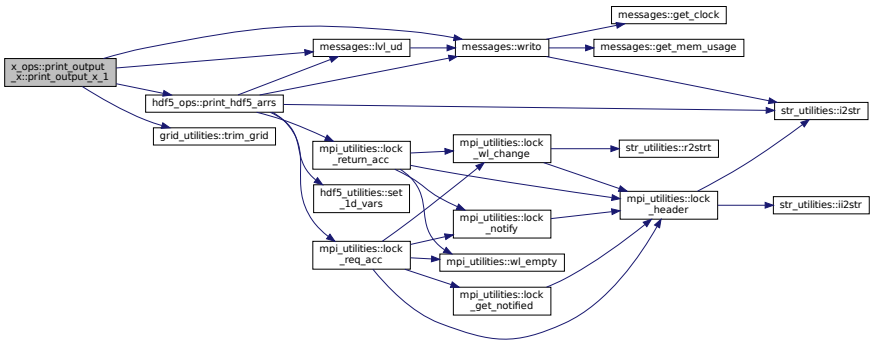
vectorial version.

Parameters

in	*grid_x*	perturbation grid variables
in	*x*	vectorial perturbation variables
in	*data_name*	name under which to store
in	*rich_lvl*	Richardson level to print
in	*par_div*	is a parallely divided grid
in	*lim_sec_x*	limits of `m_X` (pol. flux) or `n_X` (tor. flux)

Definition at line 411 of file X_ops.f90.

Here is the call graph for this function:



C.50.2.2 print_output_x_2()

```
integer function x_ops::print_output_x::print_output_x_2 (
               type(grid_type), intent(in) grid_X,
               type(x_2_type), intent(in) X,
               character(len=*), intent(in) data_name,
               integer, intent(in), optional rich_lvl,
               logical, intent(in), optional par_div,
               integer, dimension(2,2), intent(in), optional lim_sec_X,
               logical, intent(in), optional is_field_averaged )
```

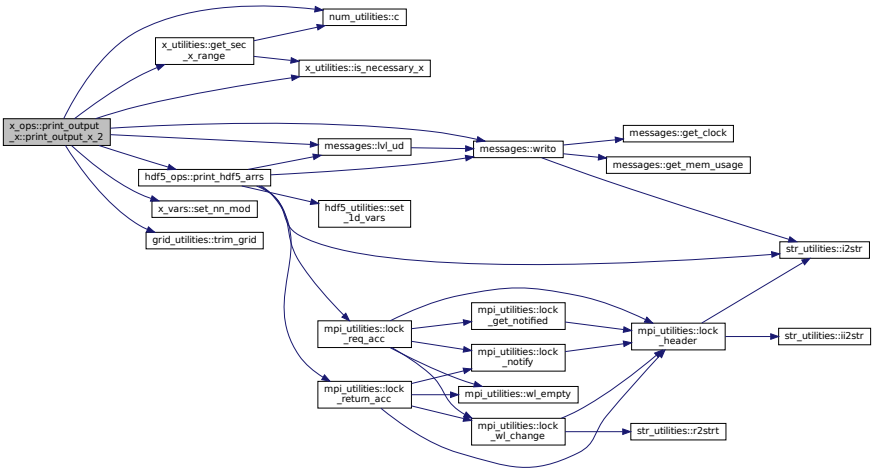
tensorial version

Parameters

in	*grid_x*	perturbation grid variables
in	*x*	tensorial perturbation variables
in	*data_name*	name under which to store
in	*rich_lvl*	Richardson level to print
in	*par_div*	is a parallely divided grid
in	*lim_sec_x*	limits of m_X (pol. flux) or n_X (tor. flux)
in	*is_field_averaged*	whether field-averaged, so only one dimension for first index

Definition at line 603 of file X_ops.f90.

Here is the call graph for this function:



C.51 eq_ops::redistribute_output_eq Interface Reference

Redistribute the equilibrium variables, but only the Flux variables are saved.

Public Member Functions

- integer function redistribute_output_eq_1 (grid, grid_out, eq, eq_out)
  flux version
- integer function redistribute_output_eq_2 (grid, grid_out, eq, eq_out)
  metric version

C.51.1 Detailed Description

Redistribute the equilibrium variables, but only the Flux variables are saved.

See also

redistribute_output_grid()

Returns

ierr

Definition at line 103 of file eq_ops.f90.

C.51.2 Member Function/Subroutine Documentation

C.51.2.1 redistribute_output_eq_1()

```
integer function eq_ops::redistribute_output_eq::redistribute_output_eq_1 (
               type(grid_type), intent(in) grid,
               type(grid_type), intent(in) grid_out,
               type(eq_1_type), intent(in) eq,
               type(eq_1_type), intent(inout) eq_out )
```

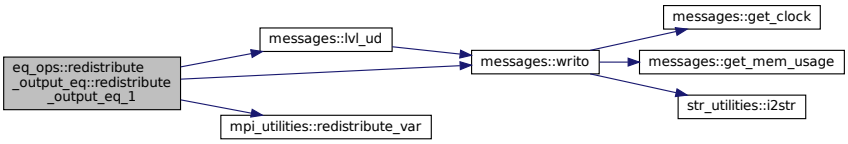
flux version

Parameters

in	*grid*	equilibrium grid variables
in	*grid_out*	redistributed equilibrium grid variables
in	*eq*	flux equilibrium variables
in,out	*eq_out*	flux equilibrium variables in redistributed grid

Definition at line 2597 of file eq_ops.f90.

Here is the call graph for this function:



C.51.2.2 redistribute_output_eq_2()

```
integer function eq_ops::redistribute_output_eq::redistribute_output_eq_2 (
               type(grid_type), intent(in) grid,
               type(grid_type), intent(in) grid_out,
               type(eq_2_type), intent(in) eq,
               type(eq_2_type), intent(inout) eq_out )
```

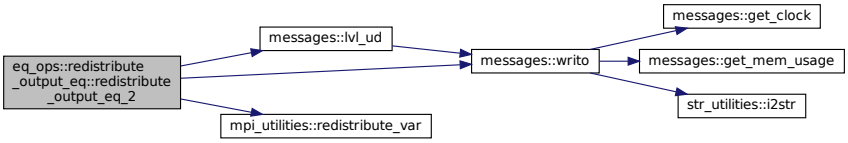
metric version

Parameters

in	*grid*	equilibrium grid variables
in	*grid_out*	redistributed equilibrium grid variables
in	*eq*	metric equilibrium variables
in,out	*eq_out*	metric equilibrium variables in redistributed grid

Definition at line 2665 of file eq_ops.f90.

Here is the call graph for this function:



C.52 x_ops::redistribute_output_x Interface Reference

Redistribute the perturbation variables.

Public Member Functions

- integer function redistribute_output_x_1 (mds, grid, grid_out, X, X_out)
  flux version
- integer function redistribute_output_x_2 (mds, grid, grid_out, X, X_out)
  metric version

C.52.1 Detailed Description

Redistribute the perturbation variables.

See also

redistribute_output_grid()

Returns

ierr

Definition at line 51 of file X_ops.f90.

C.52.2 Member Function/Subroutine Documentation

C.52.2.1 redistribute_output_x_1()

```
integer function x_ops::redistribute_output_x::redistribute_output_x_1 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid,
               type(grid_type), intent(in) grid_out,
               type(x_1_type), intent(in) X,
               type(x_1_type), intent(inout) X_out )
```

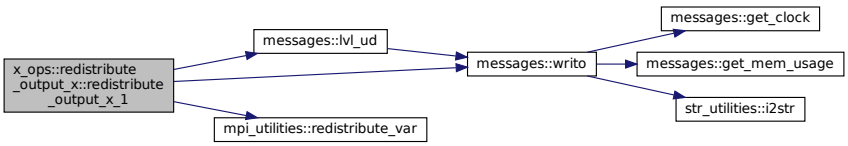
flux version

Parameters

in	*mds*	general modes variables
in	*grid*	perturbation grid variables
in	*grid_out*	redistributed perturbation grid variables
in	*x*	vectorial perturbation variables
in,out	*x_out*	vectorial perturbation variables in redistributed grid

Definition at line 178 of file X_ops.f90.

Here is the call graph for this function:



C.52.2.2 redistribute_output_x_2()

```
integer function x_ops::redistribute_output_x::redistribute_output_x_2 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid,
               type(grid_type), intent(in) grid_out,
               type(x_2_type), intent(in) X,
               type(x_2_type), intent(inout) X_out )
```

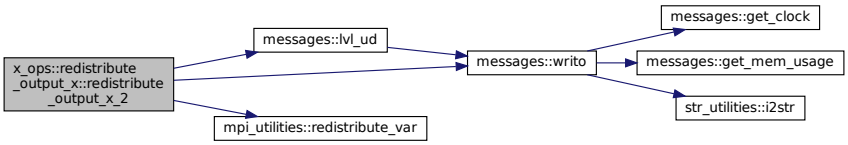
metric version

Parameters

in	*mds*	general modes variables
in	*grid*	perturbation grid variables
in	*grid_out*	redistributed perturbation grid variables
in	*x*	tensorial perturbation variables
in,out	*x_out*	tensorial perturbation variables in redistributed grid

Definition at line 277 of file X_ops.f90.

Here is the call graph for this function:



C.53 hdf5_ops::reset_hdf5_item Interface Reference

Resets an HDF5 XDMF item.

Public Member Functions

- subroutine reset_hdf5_item_ind (XDMF_item, ind_plot)
  individual version
- subroutine reset_hdf5_item_arr (XDMF_items, ind_plot)
  array version

C.53.1 Detailed Description

Resets an HDF5 XDMF item.

Note

individual version cannot make use of array version because then the deallocation does not work properly.

Definition at line 57 of file HDF5_ops.f90.

C.53.2 Member Function/Subroutine Documentation

C.53.2.1 reset_hdf5_item_arr()

```
subroutine hdf5_ops::reset_hdf5_item::reset_hdf5_item_arr (
               type(xml_str_type), dimension(:), intent(inout) XDMF_items,
               logical, intent(in), optional ind_plot )
```

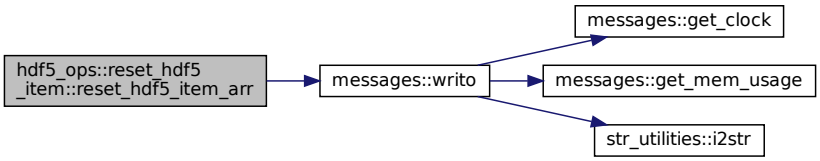
array version

Parameters

in,out	*xdmf_items*	XDMF items to reset
in	*ind_plot*	whether individual plot

Definition at line 1794 of file HDF5_ops.f90.

Here is the call graph for this function:



C.53.2.2 reset_hdf5_item_ind()

```
subroutine hdf5_ops::reset_hdf5_item::reset_hdf5_item_ind (
               type(xml_str_type), intent(inout) XDMF_item,
               logical, intent(in), optional ind_plot )
```

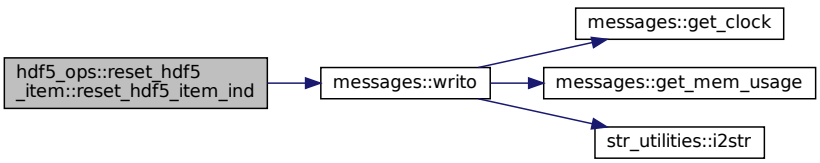
individual version

Parameters

in,out	*xdmf_item*	XDMF item to reset
in	*ind_plot*	whether individual plot

Definition at line 1834 of file HDF5_ops.f90.

Here is the call graph for this function:



C.54 num_utilities::round_with_tol Interface Reference

Rounds an arry of values to limits, with a tolerance 10^{-5} that can optionally be modified.

Public Member Functions

- integer function round_with_tol_ind (val, lim_lo, lim_hi, tol)
  individual version
- integer function round_with_tol_arr (vals, lim_lo, lim_hi, tol)
  array version

C.54.1 Detailed Description

Rounds an arry of values to limits, with a tolerance 10^{-5} that can optionally be modified.

Definition at line 169 of file num_utilities.f90.

C.54.2 Member Function/Subroutine Documentation

C.54.2.1 round_with_tol_arr()

```
integer function num_utilities::round_with_tol::round_with_tol_arr (
               real(dp), dimension(:), intent(inout) vals,
               real(dp), intent(in) lim_lo,
               real(dp), intent(in) lim_hi,
               real(dp), intent(in), optional tol )
```

array version

Parameters

in,out	*vals*	values to be rounded
in	*lim_lo*	lower limit on val
in	*lim_hi*	upper limit on val
in	*tol*	tolerance

Definition at line 1383 of file num_utilities.f90.

C.54.2.2 round_with_tol_ind()

```
integer function num_utilities::round_with_tol::round_with_tol_ind (
               real(dp), intent(inout) val,
               real(dp), intent(in) lim_lo,
               real(dp), intent(in) lim_hi,
               real(dp), intent(in), optional tol )
```

individual version

Parameters

in,out	val	value to be rounded
in	lim↩_lo	lower limit on val
in	lim↩_hi	upper limit on val
in	tol	tolerance

Definition at line 1421 of file num_utilities.f90.

C.55 x_utilities::sec_ind_loc2tot Interface Reference

Returns the sec_ind_tot used to refer to a perturbation quantity.

Public Member Functions

- integer function sec_ind_loc2tot_1 (id, lim_sec_X)
  vectorial version
- integer function, dimension(2) sec_ind_loc2tot_2 (id, jd, lim_sec_X)
  tensorial version

C.55.1 Detailed Description

Returns the sec_ind_tot used to refer to a perturbation quantity.

Definition at line 22 of file X_utilities.f90.

C.55.2 Member Function/Subroutine Documentation

C.55.2.1 sec_ind_loc2tot_1()

```
integer function x_utilities::sec_ind_loc2tot::sec_ind_loc2tot_1 (
               integer, intent(in) id,
               integer, dimension(2), intent(in), optional lim_sec_X )
```

vectorial version

Parameters

in	*id*	mode index
in	*lim_sec_x*	limits of m_X (pol. flux) or n_X (tor. flux)

Returns

output

Definition at line 32 of file X_utilities.f90.

C.55.2.2 sec_ind_loc2tot_2()

```
integer function, dimension(2) x_utilities::sec_ind_loc2tot::sec_ind_loc2tot_2 (
               integer, intent(in) id,
               integer, intent(in) jd,
               integer, dimension(2,2), intent(in), optional lim_sec_X )
```

tensorial version

Parameters

in	*id*	mode index for dimension 1
in	*jd*	mode index for dimension 1
in	*lim_sec_x*	limits of m_X (pol flux) or n_X (tor flux) for both dimensions

Returns

output

Definition at line 49 of file X_utilities.f90.

C.56 x_vars::set_nm_x Interface Reference

Sets n_X and m_X.

Public Member Functions

- subroutine set_nm_x_1 (mds, grid_X, lim_sec_X_o, n_X_loc, m_X_loc, lim_sec_X_i)
  vectorial version
- subroutine set_nm_x_2 (mds, grid_X, lim_sec_X_o, n_X_1, m_X_1, n_X_2, m_X_2, lim_sec_X_i)
  tensorial version

C.56.1 Detailed Description

Sets n_X and m_X.

By default, this is done using by default global X_vars variables but optionally different limits for the secondary mode numbers (m_X for poloidal flux or n_X for toroidal flux).

Note

n_X and m_X need to have been set up with the same limits as the grid used here. This is done in setup_nm_X().

Definition at line 116 of file X_vars.f90.

C.56.2 Member Function/Subroutine Documentation

C.56.2.1 set_nm_x_1()

```
subroutine x_vars::set_nm_x::set_nm_x_1 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               integer, dimension(2), intent(inout) lim_sec_X_o,
               integer, dimension(:,:), intent(inout), allocatable n_X_loc,
               integer, dimension(:,:), intent(inout), allocatable m_X_loc,
               integer, dimension(2), intent(in), optional lim_sec_X_i )
```

vectorial version

Parameters

in	*mds*	general modes variables
in	*grid_x*	perturbation grid
in,out	*lim_sec_x_o*	limits on secondary mode numbers
in,out	*n_x_loc*	toroidal mode numbers
in,out	*m_x_loc*	poloidal mode numbers
in	*lim_sec_x_i*	optional limits on secondary mode numbers

Definition at line 145 of file X_vars.f90.

C.56.2.2 set_nm_x_2()

```
subroutine x_vars::set_nm_x::set_nm_x_2 (
               type(modes_type), intent(in) mds,
               type(grid_type), intent(in) grid_X,
               integer, dimension(2,2), intent(inout) lim_sec_X_o,
               integer, dimension(:,:), intent(inout), allocatable n_X_1,
               integer, dimension(:,:), intent(inout), allocatable m_X_1,
               integer, dimension(:,:), intent(inout), allocatable n_X_2,
               integer, dimension(:,:), intent(inout), allocatable m_X_2,
               integer, dimension(2,2), intent(in), optional lim_sec_X_i )
```

tensorial version

Parameters

in	*mds*	general modes variables
in	*grid_x*	perturbation grid
in,out	*lim_sec_x_o*	limits on secondary mode numbers
in,out	*n_x_1*	toroidal mode numbers for dimension 1
in,out	*m_x_1*	poloidal mode numbers for dimension 1
in,out	*n_x_2*	toroidal mode numbers for dimension 2
in,out	*m_x_2*	poloidal mode numbers for dimension 2
in	*lim_sec_x_i*	optional limits on secondary mode numbers

Definition at line 169 of file X_vars.f90.

C.57 sol_vars::sol_type Type Reference

solution type

Public Member Functions

- procedure init => init_sol
  initialize
- procedure dealloc => dealloc_sol
  deallocate

Public Attributes

- integer n_mod
  size of n and m (nr. of modes)
- integer, dimension(2) lim_sec_x
  limits of m_X (pol. flux) or n_X (tor. flux)

- integer, dimension(:,:), allocatable n
  vector of toroidal mode numbers
- integer, dimension(:,:), allocatable m
  vector of poloidal mode numbers
- complex(dp), dimension(:,:,:), allocatable vec
  Eigenvector solution.
- complex(dp), dimension(:), allocatable val
  Eigenvalue solution.
- real(dp) estim_mem_usage
  estimated memory usage

C.57.1 Detailed Description

solution type

The arrays here are of the form:

- `val: (1:n_EV)`
- `vec: (1:n_mod,1:loc_n_r,1:n_EV)`

Definition at line 30 of file sol_vars.f90.

C.57.2 Member Function/Subroutine Documentation

C.57.2.1 dealloc()

```
procedure sol_vars::sol_type::dealloc
```

deallocate

Definition at line 44 of file sol_vars.f90.

C.57.2.2 init()

```
procedure sol_vars::sol_type::init
```

initialize

Definition at line 42 of file sol_vars.f90.

C.57.3 Member Data Documentation

C.57.3.1 estim_mem_usage

```
real(dp) sol_vars::sol_type::estim_mem_usage
```

estimated memory usage

Note

Debug version only

Definition at line 38 of file sol_vars.f90.

C.57.3.2 lim_sec_x

```
integer, dimension(2) sol_vars::sol_type::lim_sec_x
```

limits of `m_X` (pol. flux) or `n_X` (tor. flux)

Definition at line 32 of file sol_vars.f90.

C.57.3.3 m

```
integer, dimension(:,:), allocatable sol_vars::sol_type::m
```

vector of poloidal mode numbers

Definition at line 34 of file sol_vars.f90.

C.57.3.4 n

```
integer, dimension(:,:), allocatable sol_vars::sol_type::n
```

vector of toroidal mode numbers

Definition at line 33 of file sol_vars.f90.

C.57.3.5 n_mod

```
integer sol_vars::sol_type::n_mod
```

size of n and m (nr. of modes)

Definition at line 31 of file sol_vars.f90.

C.57.3.6 val

```
complex(dp), dimension(:), allocatable sol_vars::sol_type::val
```

Eigenvalue solution.

Definition at line 36 of file sol_vars.f90.

C.57.3.7 vec

```
complex(dp), dimension(:,:,:), allocatable sol_vars::sol_type::vec
```

Eigenvector solution.

Definition at line 35 of file sol_vars.f90.

C.58 num_utilities::spline Interface Reference

Wrapper to the pspline library, making it easier to use for 1-D applications where speed is not the main priority. If spline representations are to be reused, manually use the library.

Public Member Functions

- integer function spline_real (x, y, xnew, ynew, ord, deriv, bcs, bcs_val, extrap)
  real version
- integer function spline_complex (x, y, xnew, ynew, ord, deriv, bcs, bcs_val, extrap)
  complex version

C.58.1 Detailed Description

Wrapper to the pspline library, making it easier to use for 1-D applications where speed is not the main priority. If spline representations are to be reused, manually use the library.

Order 1 (linear), 2 (akima hermite) or 3 (cubic) possible. Boundary conditions are possible:

- -1: periodic
- 0: not-a-knot
- 1: prescribe first derivative
- 2: prescribe second derivative

However, for order 2 boundary condition 2 is not available and for order 1 none of them.

Furthermore, derivatives can be specified:

- up to 1 for order 1 and 2
- up to 2 for order 3

Finally, extrapolation can be performed as well.

Returns

ierr

Definition at line 276 of file num_utilities.f90.

C.58.2 Member Function/Subroutine Documentation

C.58.2.1 spline_complex()

```
integer function num_utilities::spline::spline_complex (
               real(dp), dimension(:), intent(in) x,
               complex(dp), dimension(:), intent(in) y,
               real(dp), dimension(:), intent(in) xnew,
               complex(dp), dimension(:), intent(out) ynew,
               integer, intent(in), optional ord,
               integer, intent(in), optional deriv,
               integer, dimension(2), intent(in), optional bcs,
               complex(dp), dimension(2), intent(in), optional bcs_val,
               logical, intent(in), optional extrap )
```

complex version

Parameters

in	*x*	coordinates
in	*y*	function value
in	*xnew*	new coordinates
out	*ynew*	new function values
in	*ord*	order [def 3]
in	*deriv*	derivative [def 0]
in	*bcs*	boundary conditions [def 0]
in	*bcs_val*	boundary conditions [no def]
in	*extrap*	whether extrapolation is allowed [def .false.]

Definition at line 2003 of file num_utilities.f90.

C.58.2.2 spline_real()

```
integer function num_utilities::spline::spline_real (
               real(dp), dimension(:), intent(in), target x,
               real(dp), dimension(:), intent(in) y,
               real(dp), dimension(:), intent(in), target xnew,
               real(dp), dimension(:), intent(out) ynew,
               integer, intent(in), optional ord,
               integer, intent(in), optional deriv,
               integer, dimension(2), intent(in), optional bcs,
               real(dp), dimension(2), intent(in), optional bcs_val,
               logical, intent(in), optional extrap )
```

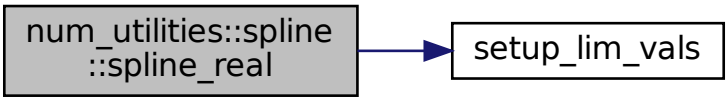
real version

Parameters

in	*x*	coordinates
in	*y*	function value
in	*xnew*	new coordinates
out	*ynew*	new function values
in	*ord*	order [def 3]
in	*deriv*	derivative [def 0]
in	*bcs*	boundary conditions [def 0]
in	*bcs_val*	boundary conditions [no def]
in	*extrap*	whether extrapolation is allowed [def .false.]

Definition at line 1676 of file num_utilities.f90.

Here is the call graph for this function:



C.59 eq_utilities::transf_deriv Interface Reference

Calculates derivatives in a coordinate system B from derivatives in a coordinates system A, making use of the transformation matrix $\overline{\mathsf{T}}^{\mathsf{A}}_{\mathsf{B}}$.

Public Member Functions

- integer recursive function transf_deriv_3_ind (X_A, T_BA, X_B, max_deriv, deriv_B, deriv_A_input)
  3-D scalar version with one derivative
- integer function transf_deriv_3_arr (X_A, T_BA, X_B, max_deriv, derivs)
  3-D scalar version with multiple derivatives
- integer function transf_deriv_3_arr_2d (X_A, T_BA, X_B, max_deriv, derivs)
  3-D matrix version with multiple derivatives
- integer recursive function transf_deriv_1_ind (X_A, T_BA, X_B, max_deriv, deriv_B, deriv_A_input)
  1-D scalar version with one derivative

C.59.1 Detailed Description

Calculates derivatives in a coordinate system B from derivatives in a coordinates system A, making use of the transformation matrix $\overline{\mathsf{T}}^{\mathsf{A}}_{\mathsf{B}}$.

The routine works by exchanging the derivatives in the coordinates B for derivatives in coordinates A using the formula

$$\mathbf{D}^m_{\mathsf{B}} X = \overline{\mathsf{T}}^{\mathsf{A}}_{\mathsf{B}} \mathbf{D}^1_{\mathsf{A}} \left(\mathbf{D}^{m-1}_{\mathsf{B}} X \right),$$

where $\mathbf{D}^m$ is a tensor of rank m that contains all the derivatives of total rank m. For example,

$$\mathbf{D}^1 = \vec{D} = \begin{pmatrix} \frac{\partial}{\partial y^1} \\ \frac{\partial}{\partial y^2} \\ \frac{\partial}{\partial u^3} \end{pmatrix}$$

This is done for the derivatives in each of the coordinates B until degree 0 is reached.

Furthermore, each of these degrees of derivatives in coordinates B can be be derived optionally in the original coordinate system A, which yields the formula:

$$\mathbf{D}_{\mathsf{A}}^{p}\left(\mathbf{D}_{\mathsf{B}}^{m}X\right)=\sum_{q}\left(\begin{array}{c}p\\q\end{array}\right)\mathbf{D}_{\mathsf{A}}^{q}\left(\overline{\mathsf{T}}_{\mathsf{B}}^{\mathsf{A}}\right)\left(\mathbf{D}_{\mathsf{A}}^{p-q}\mathbf{D}_{\mathsf{A}}^{1}\right)\left(\mathbf{D}_{\mathsf{B}}^{m-1}X\right)$$

This way, ultimately the desired derivatives in the coordinates B can be obtained recursively from the lower orders in the coordinates B and higher orders in the coordinates A.

For example:

- For order M, the formula can be used with $p=0$.
- For this, it is necessary that $\mathbf{D}_{\mathsf{A}}^{1}\mathbf{D}_{\mathsf{B}}^{m-1}X$ be precalculated, which can be done using the formula with $p=1$ and $m=M-1$.
- For this, it is necessary that $\mathbf{D}_{\mathsf{A}}^{1}\mathbf{D}_{\mathsf{B}}^{M-1}X$ and $\mathbf{D}_{\mathsf{A}}^{2}\mathbf{D}_{\mathsf{B}}^{M-1}X$ are precalculated, which can be done with $p=1$ and $m=M-1$ as well as $p=2$ and $m=M-1$.
- etc.

See also

[15] for more detailed information.

Returns

ierr

Definition at line 118 of file eq_utilities.f90.

C.59.2 Member Function/Subroutine Documentation

C.59.2.1 transf_deriv_1_ind()

```
integer recursive function eq_utilities::transf_deriv::transf_deriv_1_ind (
               real(dp), dimension(1:,0:), intent(in) X_A,
               real(dp), dimension(1:,0:), intent(in) T_BA,
               real(dp), dimension(1:), intent(inout) X_B,
               integer, intent(in) max_deriv,
               integer, intent(in) deriv_B,
               integer, intent(in), optional deriv_A_input )
```

1-D scalar version with one derivative

Parameters

in	*x_a*	variable and derivs. in coord. system A
in,out	*x_b*	requested derivs. of variable in coord. system B
in	*t_ba*	transf. mat. and derivs. between coord. systems A and B
in	*deriv_a_input*	derivs. in coord. system A (optional)
in	*deriv_b*	derivs. in coord. system B
in	*max_deriv*	maximum degrees of derivs.

Definition at line 681 of file eq_utilities.f90.

C.59.2.2 transf_deriv_3_arr()

```
integer function eq_utilities::transf_deriv::transf_deriv_3_arr (
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) X_A,
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(in) T_BA,
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(inout) X_B,
               integer, intent(in) max_deriv,
               integer, dimension(:,:), intent(in) derivs )
```

3-D scalar version with multiple derivatives

Parameters

in	*x_a*	variable and derivs. in coord. system A
in	*t_ba*	transf. mat. and derivs. between coord. systems A and B
in,out	*x_b*	requested derivs. of variable in coord. system B
in	*max_deriv*	maximum degrees of derivs.
in	*derivs*	series of derivs. (in coordinate system B)

Definition at line 615 of file eq_utilities.f90.

C.59.2.3 transf_deriv_3_arr_2d()

```
integer function eq_utilities::transf_deriv::transf_deriv_3_arr_2d (
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(in) X_A,
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(in) T_BA,
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(inout) X_B,
               integer, intent(in) max_deriv,
               integer, dimension(:,:), intent(in) derivs )
```

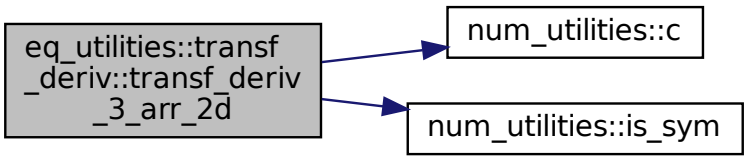
3-D matrix version with multiple derivatives

Parameters

in	*x_a*	variable and derivs. in coord. system A
in	*t_ba*	transf. mat. and derivs. between coord. systems A and B
in,out	*x_b*	requested derivs. of variable in coord. system B
in	*max_deriv*	maximum degrees of derivs.
in	*derivs*	series of derivs. (in coordinate system B)

Definition at line 640 of file eq_utilities.f90.

Here is the call graph for this function:



C.59.2.4 transf_deriv_3_ind()

```
integer recursive function eq_utilities::transf_deriv::transf_deriv_3_ind (
               real(dp), dimension(1:,1:,1:,0:,0:,0:), intent(in) X_A,
               real(dp), dimension(1:,1:,1:,1:,0:,0:,0:), intent(in) T_BA,
               real(dp), dimension(1:,1:,1:), intent(inout) X_B,
               integer, intent(in) max_deriv,
               integer, dimension(:), intent(in) deriv_B,
               integer, dimension(:), intent(in), optional deriv_A_input )
```

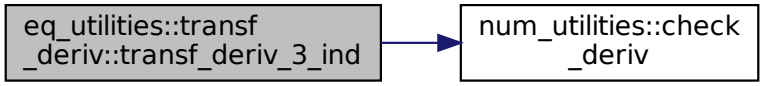
3-D scalar version with one derivative

Parameters

in	*x_a*	variable and derivs. in coord. system A
in	*t_ba*	transf. mat. and derivs. between coord. systems A and B
in,out	*x_b*	requested derivs. of variable in coord. system B
in	*max_deriv*	maximum degrees of derivs.
in	*deriv_b*	derivs. in coord. system B
in	*deriv_a_input*	derivs. in coord. system A (optional)

Definition at line 508 of file eq_utilities.f90.

Here is the call graph for this function:



C.60 vac_vars::vac_type Type Reference

vacuum type

Public Member Functions

- procedure init => init_vac
  initialize
- procedure dealloc => dealloc_vac
  deallocate

Public Attributes

- integer style
  style of vacuum (1: field-line 3-D, 2: axisymmetric)
- integer prim_x
  primary mode number
- integer ctxt_hg
  context for H and G
- integer n_bnd
  number of points in boundary
- integer bs
  block size in cyclical storage
- integer mpi_comm
  communicator for vacuum
- integer, dimension(blacsctxtsize) desc_h
  descriptor for H
- integer, dimension(blacsctxtsize) desc_g
  descriptor for G
- integer, dimension(2) n_p
  nr. of processes in grid
- integer, dimension(2) n_ang
  number of angles (1) and number of field lines (2)
- integer, dimension(2) ind_p
  index of local process in grid
- integer, dimension(2) n_loc

local number of rows and columns

- integer, dimension(:), allocatable sec_x
  secondary mode numbers
- integer, dimension(:,:), allocatable lims_c
  column limits for different subrows of G and H
- integer, dimension(:,:), allocatable lims_r
  row limits for different subcolumns of G and H
- real(dp) jq
  iota (tor. flux) or q (pol. flux) at edge
- real(dp), dimension(:,:), allocatable ang
  angle along field line, for each field line
- real(dp), dimension(:,:), allocatable norm
  J nabla psi normal vector.
- real(dp), dimension(:,:), allocatable dnorm
  poloidal derivative of norm (only for style 2)
- real(dp), dimension(:,:), allocatable h_fac
  metric factors (1,1), (1,3) and (3,3) (only for style 1)
- real(dp), dimension(:,:), allocatable x_vec
  Cartesian vector of position.
- real(dp), dimension(:,:), allocatable h
  H coefficient.
- real(dp), dimension(:,:), allocatable g
  G coefficient.
- complex(dp), dimension(:,:), allocatable res
  vacuum response
- real(dp) estim_mem_usage
  estimated memory usage

C.60.1 Detailed Description

vacuum type

The arrays here are of the form:

- `H, G: (n_loc,n_loc)`
- `res: (n_mod_X,n_mod_X)`

where `n_loc` is the number of points in the boundary, i.e. a subset of `n_bnd`.

For vacuum style 1, the vacuum is assumed to be equidsitant in the coordinate along the magnetic field lines and, if there are multiple field lines, also in the field line label α. The limits on the angles are assumed to be given by the global variables `grid_vars.min_par_x`, `grid_vars.max_par_x` and `grid_vars.min_alpha`, `grid_vars.max_alpha`.

For vacuum style 2, there is an additional angle `ang`. It is composed of the angles along the magnetic fields (which refers to `angle_1` in the discussion of `grid_vars.grid_type`), of which there can be multiple, but the total sum must be equal to `n_bnd`.

Definition at line 46 of file vac_vars.f90.

C.60.2 Member Function/Subroutine Documentation

C.60.2.1 dealloc()

```
procedure vac_vars::vac_type::dealloc
```

deallocate

Definition at line 78 of file vac_vars.f90.

C.60.2.2 init()

```
procedure vac_vars::vac_type::init
```

initialize

Definition at line 76 of file vac_vars.f90.

C.60.3 Member Data Documentation

C.60.3.1 ang

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::ang
```

angle along field line, for each field line

Definition at line 63 of file vac_vars.f90.

C.60.3.2 bs

```
integer vac_vars::vac_type::bs
```

block size in cyclical storage

Definition at line 51 of file vac_vars.f90.

C.60.3.3 ctxt_hg

```
integer vac_vars::vac_type::ctxt_hg
```

context for H and G

Definition at line 49 of file vac_vars.f90.

C.60.3.4 desc_g

```
integer, dimension(blacsctxtsize) vac_vars::vac_type::desc_g
```

descriptor for G

Definition at line 54 of file vac_vars.f90.

C.60.3.5 desc_h

```
integer, dimension(blacsctxtsize) vac_vars::vac_type::desc_h
```

descriptor for H

Definition at line 53 of file vac_vars.f90.

C.60.3.6 dnorm

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::dnorm
```

poloidal derivative of norm (only for style 2)

Definition at line 65 of file vac_vars.f90.

C.60.3.7 estim_mem_usage

```
real(dp) vac_vars::vac_type::estim_mem_usage
```

estimated memory usage

Note

Debug version only

Definition at line 72 of file vac_vars.f90.

C.60.3.8 g

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::g
```

G coefficient.

Definition at line 69 of file vac_vars.f90.

C.60.3.9 h

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::h
```

H coefficient.

Definition at line 68 of file vac_vars.f90.

C.60.3.10 h_fac

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::h_fac
```

metric factors (1,1), (1,3) and (3,3) (only for style 1)

Definition at line 66 of file vac_vars.f90.

C.60.3.11 ind_p

```
integer, dimension(2) vac_vars::vac_type::ind_p
```

index of local process in grid

Definition at line 57 of file vac_vars.f90.

C.60.3.12 jq

```
real(dp) vac_vars::vac_type::jq
```

iota (tor. flux) or q (pol. flux) at edge

Definition at line 62 of file vac_vars.f90.

C.60.3.13 lims_c

```
integer, dimension(:,:), allocatable vac_vars::vac_type::lims_c
```

column limits for different subrows of G and H

Definition at line 60 of file vac_vars.f90.

C.60.3.14 lims_r

```
integer, dimension(:,:), allocatable vac_vars::vac_type::lims_r
```

row limits for different subcolumns of G and H

Definition at line 61 of file vac_vars.f90.

C.60.3.15 mpi_comm

```
integer vac_vars::vac_type::mpi_comm
```

communicator for vacuum

Definition at line 52 of file vac_vars.f90.

C.60.3.16 n_ang

```
integer, dimension(2) vac_vars::vac_type::n_ang
```

number of angles (1) and number of field lines (2)

Definition at line 56 of file vac_vars.f90.

C.60.3.17 n_bnd

```
integer vac_vars::vac_type::n_bnd
```

number of points in boundary

Definition at line 50 of file vac_vars.f90.

C.60.3.18 n_loc

```
integer, dimension(2) vac_vars::vac_type::n_loc
```

local number of rows and columns

Definition at line 58 of file vac_vars.f90.

C.60.3.19 n_p

```
integer, dimension(2) vac_vars::vac_type::n_p
```

nr. of processes in grid

Definition at line 55 of file vac_vars.f90.

C.60.3.20 norm

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::norm
```

J nabla psi normal vector.

Definition at line 64 of file vac_vars.f90.

C.60.3.21 prim_x

```
integer vac_vars::vac_type::prim_x
```

primary mode number

Definition at line 48 of file vac_vars.f90.

C.60.3.22 res

```
complex(dp), dimension(:,:), allocatable vac_vars::vac_type::res
```

vacuum response

Definition at line 70 of file vac_vars.f90.

C.60.3.23 sec_x

```
integer, dimension(:), allocatable vac_vars::vac_type::sec_x
```

secondary mode numbers

Definition at line 59 of file vac_vars.f90.

C.60.3.24 style

```
integer vac_vars::vac_type::style
```

style of vacuum (1: field-line 3-D, 2: axisymmetric)

Definition at line 47 of file vac_vars.f90.

C.60.3.25 x_vec

```
real(dp), dimension(:,:), allocatable vac_vars::vac_type::x_vec
```

Cartesian vector of position.

Definition at line 67 of file vac_vars.f90.

C.61 hdf5_vars::var_1d_type Type Reference

1D equivalent of multidimensional variables, used for internal HDF5 storage.

Public Attributes

- real(dp), dimension(:), allocatable p
  1D equivalent of data of variable
- integer, dimension(:), allocatable tot_i_min
  total min.of indices of variable
- integer, dimension(:), allocatable tot_i_max
  total max.of indices of variable
- integer, dimension(:), allocatable loc_i_min
  group min.of indices of variable
- integer, dimension(:), allocatable loc_i_max
  group max.of indices of variable
- character(len=max_str_ln) var_name
  name of variable

C.61.1 Detailed Description

1D equivalent of multidimensional variables, used for internal HDF5 storage.

Definition at line 48 of file HDF5_vars.f90.

C.61.2 Member Data Documentation

C.61.2.1 loc_i_max

```
integer, dimension(:), allocatable hdf5_vars::var_1d_type::loc_i_max
```

group max.of indices of variable

Definition at line 53 of file HDF5_vars.f90.

C.61.2.2 loc_i_min

```
integer, dimension(:), allocatable hdf5_vars::var_1d_type::loc_i_min
```

group min.of indices of variable

Definition at line 52 of file HDF5_vars.f90.

C.61.2.3 p

```
real(dp), dimension(:), allocatable hdf5_vars::var_1d_type::p
```

1D equivalent of data of variable

Definition at line 49 of file HDF5_vars.f90.

C.61.2.4 tot_i_max

```
integer, dimension(:), allocatable hdf5_vars::var_1d_type::tot_i_max
```

total max.of indices of variable

Definition at line 51 of file HDF5_vars.f90.

C.61.2.5 tot_i_min

```
integer, dimension(:), allocatable hdf5_vars::var_1d_type::tot_i_min
```

total min.of indices of variable

Definition at line 50 of file HDF5_vars.f90.

C.61.2.6 var_name

```
character(len=max_str_ln) hdf5_vars::var_1d_type::var_name
```

name of variable

Definition at line 54 of file HDF5_vars.f90.

C.62 x_vars::x_1_type Type Reference

vectorial perturbation type

Public Member Functions

- procedure init => init_x_1
  initialize
- procedure copy => copy_x_1
  copy
- procedure dealloc => dealloc_x_1
  deallocate

Public Attributes

- integer n_mod
  size of n and m (nr. of modes)
- integer, dimension(2) lim_sec_x
  limits of `m_X` (pol. flux) or `n_X` (tor. flux)
- integer, dimension(:,:), allocatable n
  vector of poloidal mode numbers
- integer, dimension(:,:), allocatable m
  vector of poloidal mode numbers
- complex(dp), dimension(:,:,:,:), allocatable u_0
  U_m^0
- complex(dp), dimension(:,:,:,:), allocatable u_1
  U_m^1
- complex(dp), dimension(:,:,:,:), allocatable du_0
  $\mathcal{J}\vec{B}\cdot\nabla U_m^0$
- complex(dp), dimension(:,:,:,:), allocatable du_1
  $\mathcal{J}\vec{B}\cdot\nabla U_m^1$
- real(dp) estim_mem_usage
  estimated memory usage

C.62.1 Detailed Description

vectorial perturbation type

The arrays here are of the form:

- `U_x_i` and DU_X_i: `(1:angle_1,1:angle_2,1;n_mod)`

See also

See grid_vars.grid_type for a discussion on `ang_1` and `ang_2`.

Definition at line 51 of file X_vars.f90.

C.62.2 Member Function/Subroutine Documentation

C.62.2.1 copy()

```
procedure x_vars::x_1_type::copy
```

copy

Definition at line 67 of file X_vars.f90.

C.62.2.2 dealloc()

```
procedure x_vars::x_1_type::dealloc
```

deallocate

Definition at line 69 of file X_vars.f90.

C.62.2.3 init()

```
procedure x_vars::x_1_type::init
```

initialize

Definition at line 65 of file X_vars.f90.

C.62.3 Member Data Documentation

C.62.3.1 du_0

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_1_type::du_0
```

$\mathcal{J}\vec{B} \cdot \nabla U_m^0$

Definition at line 58 of file X_vars.f90.

C.62.3.2 du_1

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_1_type::du_1
```

$\mathcal{J}\vec{B} \cdot \nabla U_m^1$

Definition at line 59 of file X_vars.f90.

C.62.3.3 estim_mem_usage

```
real(dp) x_vars::x_1_type::estim_mem_usage
```

estimated memory usage

Note

Debug version only

Definition at line 61 of file X_vars.f90.

C.62.3.4 lim_sec_x

```
integer, dimension(2) x_vars::x_1_type::lim_sec_x
```

limits of `m_X` (pol. flux) or `n_X` (tor. flux)

Definition at line 53 of file X_vars.f90.

C.62.3.5 m

```
integer, dimension(:,:), allocatable x_vars::x_1_type::m
```

vector of poloidal mode numbers

Definition at line 55 of file X_vars.f90.

C.62.3.6 n

```
integer, dimension(:,:), allocatable x_vars::x_1_type::n
```

vector of poloidal mode numbers

Definition at line 54 of file X_vars.f90.

C.62.3.7 n_mod

```
integer x_vars::x_1_type::n_mod
```

size of n and m (nr. of modes)

Definition at line 52 of file X_vars.f90.

C.62.3.8 u_0

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_1_type::u_0
```

U_m^0

Definition at line 56 of file X_vars.f90.

C.62.3.9 u_1

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_1_type::u_1
```

U_m^1

Definition at line 57 of file X_vars.f90.

C.63 x_vars::x_2_type Type Reference

tensorial perturbation type

Public Member Functions

- procedure init => init_x_2
  initialize
- procedure copy => copy_x_2
  copy
- procedure dealloc => dealloc_x_2
  deallocate

Public Attributes

- integer, dimension(2) n_mod
  size of n and m (nr. of modes)
- integer, dimension(2, 2) lim_sec_x
  limits of m_X (pol. flux) or n_X (tor. flux)
- integer, dimension(:,:), allocatable n_1
  vector of toroidal mode numbers of dimension 1
- integer, dimension(:,:), allocatable n_2
  vector of toroidal mode numbers of dimension 2
- integer, dimension(:,:), allocatable m_1
  vector of poloidal mode numbers of dimension 1
- integer, dimension(:,:), allocatable m_2
  vector of poloidal mode numbers of dimension 2
- complex(dp), dimension(:,:,:,:), allocatable pv_0
  $\widetilde{PV}^0$ *coefficient*
- complex(dp), dimension(:,:,:,:), allocatable pv_1
  $\widetilde{PV}^1$ *coefficient*
- complex(dp), dimension(:,:,:,:), allocatable pv_2
  $\widetilde{PV}^2$ *coefficient*
- complex(dp), dimension(:,:,:,:), allocatable kv_0
  $\widetilde{KV}^0$ *coefficient*
- complex(dp), dimension(:,:,:,:), allocatable kv_1
  $\widetilde{KV}^1$ *coefficient*
- complex(dp), dimension(:,:,:,:), allocatable kv_2
  $\widetilde{KV}^2$ *coefficient*
- real(dp) estim_mem_usage
  estimated memory usage

C.63.1 Detailed Description

tensorial perturbation type

The arrays here are of the form:

- PV_i and KV_i: (1:angle_1,1:angle_2,1;n_mod^2)

See also

See grid_vars.grid_type for a discussion on ang_1 and ang_2.

Note

This type is also used for field-averaged tensorial perturbation variables, with angle_1 of size 1.

Definition at line 81 of file X_vars.f90.

C.63.2 Member Function/Subroutine Documentation

C.63.2.1 copy()

```
procedure x_vars::x_2_type::copy
```

copy

Definition at line 101 of file X_vars.f90.

C.63.2.2 dealloc()

```
procedure x_vars::x_2_type::dealloc
```

deallocate

Definition at line 103 of file X_vars.f90.

C.63.2.3 init()

```
procedure x_vars::x_2_type::init
```

initialize

Definition at line 99 of file X_vars.f90.

C.63.3 Member Data Documentation

C.63.3.1 estim_mem_usage

```
real(dp) x_vars::x_2_type::estim_mem_usage
```

estimated memory usage

Note

Debug version only

Definition at line 95 of file X_vars.f90.

C.63.3.2 kv_0

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_2_type::kv_0
```

$\widetilde{KV}^0$ coefficient

Definition at line 91 of file X_vars.f90.

C.63.3.3 kv_1

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_2_type::kv_1
```

$\widetilde{KV}^1$ coefficient

Definition at line 92 of file X_vars.f90.

C.63.3.4 kv_2

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_2_type::kv_2
```

$\widetilde{KV}^2$ coefficient

Definition at line 93 of file X_vars.f90.

C.63.3.5 lim_sec_x

```
integer, dimension(2,2) x_vars::x_2_type::lim_sec_x
```

limits of `m_X` (pol. flux) or `n_X` (tor. flux)

Definition at line 83 of file X_vars.f90.

C.63.3.6 m_1

```
integer, dimension(:,:), allocatable x_vars::x_2_type::m_1
```

vector of poloidal mode numbers of dimension 1

Definition at line 86 of file X_vars.f90.

C.63.3.7 m_2

```
integer, dimension(:,:), allocatable x_vars::x_2_type::m_2
```

vector of poloidal mode numbers of dimension 2

Definition at line 87 of file X_vars.f90.

C.63.3.8 n_1

```
integer, dimension(:,:), allocatable x_vars::x_2_type::n_1
```

vector of toroidal mode numbers of dimension 1

Definition at line 84 of file X_vars.f90.

C.63.3.9 n_2

```
integer, dimension(:,:), allocatable x_vars::x_2_type::n_2
```

vector of toroidal mode numbers of dimension 2

Definition at line 85 of file X_vars.f90.

C.63.3.10 n_mod

```
integer, dimension(2) x_vars::x_2_type::n_mod
```

size of n and m (nr. of modes)

Definition at line 82 of file X_vars.f90.

C.63.3.11 pv_0

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_2_type::pv_0
```

$\widetilde{PV}^0$ coefficient

Definition at line 88 of file X_vars.f90.

C.63.3.12 pv_1

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_2_type::pv_1
```

$\widetilde{PV}^1$ coefficient

Definition at line 89 of file X_vars.f90.

C.63.3.13 pv_2

```
complex(dp), dimension(:,:,:,:), allocatable x_vars::x_2_type::pv_2
```

$\widetilde{PV}^2$ coefficient

Definition at line 90 of file X_vars.f90.

C.64 hdf5_vars::xml_str_type Type Reference

XML strings used in XDMF.

Public Attributes

- character(len=max_str_ln) name
  name of this item
- integer max_xml_ln = 300
  max. length of xml string
- character(len=max_xml_ln), dimension(:), allocatable xml_str
  XML string.

C.64.1 Detailed Description

XML strings used in XDMF.

Definition at line 33 of file HDF5_vars.f90.

C.64.2 Member Data Documentation

C.64.2.1 max_xml_ln

```
integer hdf5_vars::xml_str_type::max_xml_ln = 300
```

max. length of xml string

Definition at line 35 of file HDF5_vars.f90.

C.64.2.2 name

```
character(len=max_str_ln) hdf5_vars::xml_str_type::name
```

name of this item

Definition at line 34 of file HDF5_vars.f90.

C.64.2.3 xml_str

```
character(len=max_xml_ln), dimension(:), allocatable hdf5_vars::xml_str_type::xml_str
```

XML string.

Definition at line 36 of file HDF5_vars.f90.

Appendix D

Examples

D.1 example.mk

An example makefile:

```
################################################################################
#
#   Example makefile for the program PB3D (Peeling Ballooning in 3D)
#   \author Author: Toon Weyens
#
#   Don't forget to set the directories:
#       - LIBSTELL_DIR
#       - HDF5_DIR
#       - NETCDFF_DIR (note: Fortran library)
#       - PETSC_DIR
#       - SLEPC_DIR
#       - STRUMPACK_DIR
################################################################################
################################################################################
#   Include
################################################################################
## [PETSc and SLEPc trick]
include  $(PETSC_DIR)/lib/petsc/conf/variables
include  $(SLEPC_DIR)/lib/slepc/conf/slepc_variables
## [PETSc and SLEPc trick]
## [PETSc and SLEPc trick inc]
INCLUDE = $(PETSC_FC_INCLUDES) $(SLEPC_INCLUDE)
## [PETSc and SLEPc trick inc]
## [Libstell special]
INCLUDE += -I$(LIBSTELL_DIR)/libstell_dir
## [Libstell special]
INCLUDE += -I$(STRUMPACK_DIR)/include
## [PB3D include]
INCLUDE += -I$(PB3D_DIR)/include
## [PB3D include]
INCLUDE += -I/usr/include/hdf5/openmpi
################################################################################
#   Link
################################################################################
## [PB3D libraries]
LIB_INTERNAL = libdfftpack.a libfoul.a libbspline.a
## [PB3D libraries]
LINK := $(LIB_INTERNAL)
## [PETSc and SLEPc trick lib]
LINK += $(PETSC_LIB)
LINK += $(SLEPC_LIB)
## [PETSc and SLEPc trick lib]
LINK += $(LIBSTELL_DIR)/libstell.a
LINK += -L$(STRUMPACK_DIR)/lib -lstrumpack
LINK += -L$(HDF5_DIR) -lhdf5_fortran -lhdf5
LINK += -L$(NETCDFF_DIR)/lib -lnetcdff
LINK += -Wl,-R$(NETCDFF_DIR)/lib
LINK += -lscalapack -lblacs -lblas -lm
LINK += -lstdc++ -lmpi_cxx
################################################################################
#   Compiler
################################################################################
COMPILER=mpifort
################################################################################
```

```
#   Linker
################################################################################
LINKER=mpifort
################################################################################
#   Compiler flags
#   options (used with -D[name]):
#       ldebug: debug
#       lIB: infiniband
#       lwith_gnu: use GNU compiler [default]
#       lwith_intel: use INTEL compiler, (checked for version 12.0.2)
#   note: INTEL warning 6536 is suppressed, which informs about extra "USE".
#   note: INTEL warning 6843 is suppressed, which informs about empty
#       intent(out) variables
################################################################################
COMP_FLAGS = -finit-real=snan -g -Og -Wall -Wextra -pedantic \
    -fimplicit-none -fbacktrace -fno-omit-frame-pointer \
    -fcheck=all -cpp -Dldebug# debug, profiling with gprof2dot, GCC
#COMP_FLAGS = -O3 -fbacktrace -g -fimplicit-none -fno-omit-frame-pointer \
    #-cpp# optimized, GCC
#COMP_FLAGS = -O0 -DlIB -Dldebug -g -heap-arrays 100 -recursive \
    #-ftrapuv -check bounds -check uninit -traceback -implicitnone \
    #-fno-omit-frame-pointer -cpp -Dlwith_intel -diag-disable 6536 \
    #-diag-disable 6843# debug, profiling with gprof2dot, INTEL
#COMP_FLAGS = -O3 -DlIB -traceback -g -heap-arrays 100 -recursive \
    #-implicitnone -fno-omit-frame-pointer -cpp -Dlwith_intel \
    #-diag-disable 6536 -diag-disable 6843# optimized, INTEL
COMP_FLAGS_EX= -O2 -w
COMP_FLAGS_F= -O2 -funroll-loops -fexpensive-optimizations
################################################################################
#   Link flags
################################################################################
LINK_FLAGS = -fPIC -finit-real=snan# debug
#LINK_FLAGS = -fPIC# optimized
################################################################################
#   Prepare
################################################################################
# Add "Modules" and "Libraries" to the search path for the prerequisites
VPATH = Modules:Libraries
# Contains list of source files (.o) and dependencies
DEPLIST = PB3D.dep
OBJLIST = ObjectList# defines "ObjectFiles"
# Includes source files and dependency list
include $(DEPLIST)# Dependencies of all the objects
include $(OBJLIST)# Names of all the objects
################################################################################
#   Rules
################################################################################
all:    PB3D POST
PB3D:   $(ObjectFiles) $(LIB_INTERNAL) PB3D.o
    $(LINKER) -o $@ $(ObjectFiles) PB3D.o $(LINK) $(LINK_FLAGS)
POST:   $(ObjectFiles) $(LIB_INTERNAL) POST.o
    $(LINKER) -o $@ $(ObjectFiles) POST.o $(LINK) $(LINK_FLAGS)
libdfftpack.a:  dfft.o
    ar -rcs libdfftpack.a dfft.o
libfoul.a:  foul.o
    ar -rcs libfoul.a foul.o
libbspline.a:   bspline_sub_module.o
    ar -rcs libbspline.a bspline_sub_module.o
%.o: %.f90
    $(COMPILER) $(INCLUDE) $(COMP_FLAGS) -c $<
%.o: %.f
    $(COMPILER) $(COMP_FLAGS_F) -c $<
dfft.o: dfft.f
    $(COMPILER) $(COMP_FLAGS_EX) -c $<
foul.o: foul.f90
    $(COMPILER) $(COMP_FLAGS_EX) -c $<
bspline_sub_module.o: bspline_sub_module.f90
    $(COMPILER) $(COMP_FLAGS_EX) -c $<
clean:
    @rm -f *.o *.a *.mod *~ fort.*
clean_all:
    @rm -f *.o *.mod *~ fort.* PB3D POST
```

D.2 PB3D.input

Example PB3D input.

```
&inputdata_PB3D
    min_n_par_X             = 500
    min_par_X               = 0.0
    max_par_X               = 2.0
```

```
    alpha                   = 0.0
    min_r_sol               = 0.1
    max_r_sol               = 1.0
    n_r_sol                 = 500
    prim_X                  = 10
    n_mod_X                 = 20
    BC_style                = 1 1
    U_style                 = 3
    max_tot_mem             = 8000
    max_it_rich             = 2
    use_normalization       = .true.
/
//! [some extra options for PB3D]
    rich_restart_lvl        = 1
    tol_rich                = 1.0E-4
    tol_SLEPC               = 1.0E-8 1.0E-8
//! [some extra options for PB3D]
```

D.3 POST.input

Example POST input.

```
&INPUTDATA_POST
    plot_resonance          = .true.
    plot_magn_grid          = .true.
    plot_flux_q             = .true.
    plot_sol_xi             = .true.
    plot_sol_Q              = .true.
    plot_E_rec              = .true.
    plot_B                  = .true.
    plot_J                  = .true.
    plot_kappa              = .true.
    min_r_plot              = 0.0
    max_r_plot              = 1.0
    n_theta_plot            = 101
    min_theta_plot          = 0.00
    max_theta_plot          = 2.00
    n_zeta_plot             = 1
    min_zeta_plot           = 0.00
    max_zeta_plot           = 0.00
    POST_style              = 1
    plot_grid_style         = 0
    pert_mult_factor_POST   = 0.0000
    max_tot_mem             = 8000
/
//! [some extra options for POST]
    ex_plot_style           = 2
    PB3D_rich_lvl           = 1
//! [some extra options for POST]
```

Bibliography

[1] Sivaram Ambikasaran. Fast Algorithms for Dense Numerical Linear Algebra and Applications. (August), 2013. 42

[2] Ake Bjorck and Victor Pereyra. Solutions of Vandermonde Systems of Equations. *Mathematics of Computation*, 24(12):893–903, 1970. 273

[3] Howard S. Cohl and Joel E. Tohline. A Compact Cylindrical Green's Function Expansion for the Solution of Potential Problems. *The Astrophysical Journal*, 527(1):86–101, dec 1999. 384

[4] G. Dahlquist and Å. Björk. *Numerical Methods*. Dover Publications, dover book edition, 2013. 325

[5] N. M. Ferraro, S. C. Jardin, and P. B. Snyder. Ideal and resistive edge stability calculations with M3D-C1. *Physics of Plasmas*, 17(10):102508, oct 2010. 36

[6] Per Helander. Theory of plasma confinement in non-axisymmetric magnetic fields. *Reports on Progress in Physics*, 77(8):087001, aug 2014. 384, 423

[7] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc. *ACM Transactions on Mathematical Software*, 31(3):351–362, sep 2005. 336, 348

[8] S. P. Hirshman. Steepest-descent moment method for three-dimensional magnetohydrodynamic equilibria. *Physics of Fluids*, 26(12):3553, 1983. 33

[9] P. Holoborodko. Smooth Noise Robust Differentiators, 2008. 256, 262

[10] Dominic Meiser. Replicated Computational Results (RCR) Report for "A Distributed-Memory Package for Dense Hierarchically Semi-Separable Matrix Computations Using Randomization". *ACM Transactions on Mathematical Software*, 42(4):1–5, jul 2016. 384

[11] A. B. Mikhailovskii, G.T.A. Huysmans, W. O. K. Kerner, and S. E. Sharapov. Optimisation of Computational MHD Normal-mode Analysis in Tokamaks. *Plasma Physics Reports*, 23(10):844–857, 1997. 33

[12] Cooper Redwine. Upgrading to FORTRAN 90. *Book*, page 514, 1995. 375

[13] Robert Ross, Robert Latham, William Gropp, Rajeev Thakur, Brian Toonen, and Computer Science Division. Implementing MPI-IO Atomic Mode Without File System Support. 239, 242, 243, 245, 246, 249

[14] J. Segura and A. Gil. Evaluation of toroidal harmonics. *Computer Physics Communications*, 124(1):104–122, jan 2000. 81

[15] T. Weyens. Three-dimensional linear peeling-ballooning theory in magnetic fusion devices. page 200. 84, 87, 88, 89, 108, 110, 195, 198, 202, 353, 384, 469, 480, 492, 493, 495, 501, 532, 588

[16] T. Weyens, R. Sánchez, L. García, A. Loarte, and G. Huijsmans. Three-dimensional linear peeling-ballooning theory in magnetic fusion devices. *Physics of Plasmas*, 21(4):042507, apr 2014. 28, 52, 53, 421, 424, 428

[17] T. Weyens, R. Sánchez, G. Huijsmans, A. Loarte, and L. García. PB3D: A new code for edge 3-D ideal linear peeling-ballooning stability. *Journal of Computational Physics*, 330:997–1009, feb 2017. 1, 27, 36, 52, 53, 342

[18] XDMF. XDMF Model and Format, 2015. 558

[19] Z. Zhang. An improvement to the Brent's method. *International Journal of Experimental Algorithms*, 2(1):21–26, 2011. 253