# SPEC

3.00

Generated by Doxygen 1.8.20

Fri Oct 2 2020 10:12:53

# 1 The Stepped Pressure Equilibrium Code

A PDF version of this manual is available:     `SPEC_manual.pdf`

- `Github pages`

- `Subroutine documentations`

- `SPEC on PPPL Theory Dept.`

- `MRxMHD website`

# 2 Compilation hints for SPEC

In order to run SPEC, you need a copy of the HDF5 libraries installed which has both the Fortran interface and the parallel (MPI I/O) enabled.

## 2.1 Mac

See e.g. this document for more detailed instructions:    `https://support.hdfgroup.org/ftp/HD←` `F5/current/src/unpacked/release_docs/INSTALL_CMake.txt`

In short:

1. download `hdf5-1.10.5.tar.gz` from `https://www.hdfgroup.org/downloads/hdf5/source-code/`

2. extract

```
tar xzf hdf5-1.10.5.tar.gz
```

1. cd into source folder

```
cd hdf5-1.10.5
```

1. make a build folder

```
mkdir build
```

1. cd into build folder

```
cd build
```

1. run cmake with options for parallel support and Fortran interface (parallel support and C++ interface are not compatible; so we have to disable the C++ interface)

```
cmake -DHDF5_BUILD_FORTRAN:BOOL=ON -DHDF5_ENABLE_PARALLEL:BOOL=ON -DHDF5_B↩
UILD_CPP_LIB:BOLL=OFF ..
```

1. actually build the HDF5 library

```
make
```

This should leave you with a file "hdf5-1.10.5.dmg" or similar, which you can install just as any other Mac application. During the build process of SPEC, you then only need to specify the HDF5 folder in the Makefile, which will likely be `/Applications/HDF_Group/HDF5/1.10.5`.

# 3   Manual / Documentation

## 3.1   Poloidal flux and rotational transform

Given the canonical integrable form, $\mathbf{A} = \psi\nabla\theta - \chi(\psi)\nabla\zeta$, we can derive $\mathbf{B} = \nabla\psi \times \nabla\theta + \nabla\zeta \times \nabla\psi \; \chi'$. The poloidal flux is given by

$$\Psi_p = \iint \mathbf{B} \cdot \mathbf{e}_\zeta \times \mathbf{e}_\psi \; d\zeta d\psi = 2\pi \int \chi' d\psi. \tag{1}$$

The rotational-transform is

$$\iota = \frac{\mathbf{B} \cdot \nabla\theta}{\mathbf{B} \cdot \nabla\zeta} = \chi'. \tag{2}$$

The rotational-transform has the same sign as the poloidal flux.

The SPEC representation for the magnetic vector potential is

$$\mathbf{A} = A_\theta\nabla\theta + A_\zeta\nabla\zeta, \tag{3}$$

where we can see that $A_\zeta = -\chi$. The poloidal flux is

$$\int \mathbf{B} \cdot d\mathbf{s} = \oint A_\zeta d\zeta. \tag{4}$$

It would seem that the rotational-transform has opposite sign to $A_\zeta$. To be honest, I am a little confused regarding the sign.

## 3.2   Outline

This document is intended to organise the different potentially valuable improvements to the SPEC code, which could make it more robust, faster, and increase its capabilities.

The document is divided in two categories:

Numerical Improvements : independent improvements that are of numerical importance but have no added physics value *per se*, although they may allow new or better physics investigations.

Physics Applications : research topics that could be addressed with the code, either in its present form or after the completion of one or more topics listed in Numerical Improvements .

## 3.3 Numerical Improvements

### 3.3.1 Compile code with GCC for error checking

Has been implemented in Makefile for most platforms. Checks against Intel version show small differences on the order of $10^{-15}$ relative deviation, which are likely due so slighly different optimization strategies.

### 3.3.2 Profile code with gprof to find inefficient lines of code

### 3.3.3 Run code with Valgrind to identify memory leaks

### 3.3.4 De-NAG-ification

Compilation of SPEC does not rely on NAG anymore; some functionality (e.g. SQP in ma02aa.f90) might need replacements for the NAG routines to be re-enabled.

### 3.3.5 Revision of spectral-constraints

This is bit of a mess. All the mathematics is standard, and all that is required is for someone to calmly go through lots of algebra. This task should be high priority, as SRH suspects that the spectral constraints as presently enforced result in an ill-conditioned force vector, which means that the code is overly sensitive to the initial guess and does not converge robustly. Potential speed improvements are tremendous.

### 3.3.6 Extension to arbitrary toroidal angle

This can further reduce the required Fourier resolution, and so this can reduce the computation. SRH is particularly interested in this as it will allow for exotic configurations (knots, figure-8, etc.) that cannot presently be computed.

### 3.3.7 Exploit symmetry of the metric

This is easy, but somewhat tedious. Take a look at ma00aa() to see what is required. Potential speed improvement is considerable.

### 3.3.8 symmetry of "local" Beltrami matrices

This is easy. Take a look at matrix(), which constructs the Beltrami matrices, and mp00ac(), which performs the inversion. Potential speed improvement is considerable.

### 3.3.9 Exploit block tri-diagonal structure of "global" linearized force balance matrix

This requires an efficient subroutine. SRH believes that Hirshman constructed such a routine (Hirshman et al. (2010) [4]). The potential speed improvement is tremendous. See newton() for where the tri-diagonal, linearized force-balance matrix is inverted.

### 3.3.10 Enforce Helicity constraint

This will allow investigation of different, arguably more-physical classes of equilibria. See ma02aa() .

### 3.3.11 Establish test-cases

A suite of test cases should be constructed, with different geometries etc., that run fast, and that can be benchmarked to machine precision. In the InputFiles/TestCases directory, some input files for SPEC are available for this purpose. One should write routines which execute these input files and compare the output data against a publicy-available set of output files to check SPEC before a new release is made.

### 3.3.12 Verify free-boundary

This is almost complete. The corresponding publication is being written. The virtual casing routines need to be investigated and made more efficient. The virtual casing routine in slab geometry needs revision (because of an integral over an infinite domain).

### 3.3.13 Enforcement of toroidal current profile

Adjust $\mu$'s, fluxes and/or rotational transform to obtain desired current profile (without singular currents). This is implemented and needs to be merged into the master branch. An additional routine is required to iterate on the helicity multipliers etc. as required *after* the local Beltrami fields have been calculated and *before* the global force balance iterations proceed.

### 3.3.14 Interpret eigenvectors and eigenvalues of Hessian

This is already completed: see hesian(). However, this actually computes the force gradient matrix. For toroidal geometry there is a complication; namely that the hessian matrix includes the derivatives of the spectral constraints. For Cartesian geometry, it is ready to go. SRH will begin writing a paper on the stability of slab MRxMHD equilibria.

## 3.4 Physics Applications

### 3.4.1 Calculate high-resolution equilibria, e.g. W7-X

requires: Exploit symmetry of the metric , symmetry of "local" Beltrami matrices , and other improvements that can make the code faster at high Fourier resolution

### 3.4.2 Calculate equilibria by conserving helicity and fluxes

Applications to saturated island studies, sawteeth, etc. requires: Calculate equilibria by conserving helicity and fluxes

### 3.4.3 Calculate free-boundary stellarator equilibria

to predict scrape-off-layer (SOL) topologies and $\beta$-limits. requires: Verify free-boundary Mostly complete.

### 3.4.4 Evaluate stability of MRxMHD equilibria

perhaps starting from simplest system (slab tearing). requires: Interpret eigenvectors and eigenvalues of Hessian

## 3.5 Revision of coordinate singularity: axisymmetric; polar coordinates

- Consider a general, magnetic vector potential given in Cartesian coordinates,

$$\mathbf{A} = A_x \nabla x + A_y \nabla y + A_z \nabla z + \nabla g \tag{5}$$

where $A_x$, $A_y$, $A_z$, and the as-yet-arbitrary gauge function, $g$, are regular at $(x, y) = (0, 0)$, i.e. they can be expanded as a Taylor series, e.g.

$$A_x = \sum_{i,j} \alpha_{i,j} x^i y^j, \qquad A_y = \sum_{i,j} \beta_{i,j} x^i y^j, \qquad A_z = \sum_{i,j} \gamma_{i,j} x^i y^j, \qquad g = \sum_{i,j} \delta_{i,j} x^i y^j, \tag{6}$$

for small $x$ and small $y$.

- Note that we have restricted attention to the "axisymmetric" case, as there is no dependence on $z$.

- The "polar" coordinate transformation,

$$
\begin{aligned}
x &= r \cos\theta, \\
y &= r \sin\theta, \\
z &= \zeta,
\end{aligned}
\tag{7}
$$

induces the vector transformation

$$
\begin{aligned}
\nabla x &= \cos\theta \, \nabla r &-& \quad r \sin\theta \, \nabla\theta &,\\
\nabla y &= \sin\theta \, \nabla r &+& \quad r \cos\theta \, \nabla\theta &,\\
\nabla z &= && \quad \nabla\zeta &.
\end{aligned}
\tag{8}
$$

- By repeated applications of the double-angle formula, the expressions for $A_x$, $A_y$ and $g$ can be cast as functions of $(r, \theta)$,

$$A_x = \sum_m r^m [a_{m,0} + a_{m,1} \, r^2 + a_{m,2} \, r^4 + ...] \sin(m\theta), \tag{9}$$

$$A_y = \sum_m r^m [b_{m,0} + b_{m,1} \, r^2 + b_{m,2} \, r^4 + ...] \cos(m\theta), \tag{10}$$

$$A_z = \sum_m r^m [c_{m,0} + c_{m,1} \, r^2 + c_{m,2} \, r^4 + ...] \cos(m\theta), \tag{11}$$

$$g = \sum_m r^m [g_{m,0} + g_{m,1} \, r^2 + g_{m,2} \, r^4 + ...] \sin(m\theta), \tag{12}$$

where attention is restricted to stellarator symmetric geometry, but similar expressions hold for the non-stellarator symmetric terms.

- Collecting these expressions, the vector potential can be expressed

$$\mathbf{A} = A_r \nabla r + A_\theta \nabla\theta + A_\zeta \nabla\zeta + \partial_r g \, \nabla r + \partial_\theta g \, \nabla\theta, \tag{13}$$

where

$$
\begin{aligned}
A_r =\ & r^0 && [\ ( && b_{0,0} &&+\ g_{1,0} &&) &&+\ (...)r^2 + (...)r^4 + ... &&] && \sin\theta \\
+\ & r^1 && [\ ( a_{1,0}/2 &&+\ b_{1,0}/2 &&+\ 2g_{2,0} &&) &&+\ (...)r^2 + (...)r^4 + ... &&] && \sin 2\theta \\
+\ & r^2 && [\ ( a_{2,0}/2 &&+\ b_{2,0}/2 &&+\ 3g_{3,0} &&) &&+\ (...)r^2 + (...)r^4 + ... &&] && \sin 3\theta \\
+\ & ...
\end{aligned}
\tag{14}
$$

(Note: Mathematica was used to perform the algebraic manipulations, and the relevant notebook was included as part of the SPEC `CVS` repository.)

- There is precisely enough gauge freedom so that we may choose $A_r = 0$. For example, the choice

$$
\begin{aligned}
g_{1,0} &= - & & & b_{0,0} & & , \\
g_{2,0} &= - & (\quad a_{1,0}/2 &+& b_{1,0}/2 \quad) &/ \; 2 & , \\
g_{3,0} &= - & (\quad a_{2,0}/2 &+& b_{2,0}/2 \quad) &/ \; 3 & , \\
\ldots &= & \ldots & & & &
\end{aligned}
\tag{15}
$$

eliminates the lowest order $r$ dependence in each harmonic.

- By working through the algebra (again, using Mathematica) the expressions for $A_\theta$ and $A_\zeta$ become

$$
A_\theta = r^2 f_0(\rho) + r^3 f_1(\rho)\cos(\theta) + r^4 f_2(\rho)\cos(2\theta) + r^5 f_3(\rho)\cos(3\theta) + \ldots
\tag{16}
$$

$$
A_\zeta = g_0(\rho) + r^1 g_1(\rho)\cos(\theta) + r^2 g_2(\rho)\cos(2\theta) + r^3 g_3(\rho)\cos(3\theta) + \ldots
\tag{17}
$$

where $\rho \equiv r^2$ and the $f_m(\rho)$ and $g_m(\rho)$ are abitrary polynomials in $\rho$. [The expression for $A_\zeta$ is unchanged from Eqn. (11).]

### 3.5.1 somewhat generally, ...

- For stellarator-symmetric configurations,

$$
\mathbf{A} = \sum_{m,n} A_{\theta,m,n} \cos(m\theta - n\zeta)\nabla\theta + \sum_{m,n} A_{\zeta,m,n} \cos(m\theta - n\zeta)\nabla\zeta,
\tag{18}
$$

where now the dependence on $\zeta$ is included, and the angles are arbitrary.

- The near-origin behaviour of $A_\theta$ and $A_\zeta$ given in Eqn. (16) and Eqn. (17) are flippantly generalized to

$$
A_{\theta,m,n} = r^{m+2} f_{m,n}(\rho),
\tag{19}
$$

$$
A_{\zeta,m,n} = r^m \; g_{m,n}(\rho),
\tag{20}
$$

where the $f_{m,n}(\rho)$ and $g_{m,n}(\rho)$ are arbitrary polynomials in $\rho$.

- Additional gauge freedom can be exploited: including an additional gauge term $\nabla h$ where $h$ only depends on $\zeta$, e.g.

$$
h(\zeta) = h_{0,0}\,\zeta + \sum h_{0,n}\sin(-n\zeta),
\tag{21}
$$

does not change the magnetic field and does not change any of the above discussion.

- The representation for the $A_{\theta,m,n}$ does not change, but we must clarify that Eqn. (20) holds for only the $m \neq 0$ harmonics:

$$
A_{\zeta,m,n} = r^m \; g_{m,n}(\rho), \quad \text{for} \quad m \neq 0.
\tag{22}
$$

- For the $m = 0$, $n \neq 0$ harmonics of $A_\zeta$, including the additional gauge gives $A_{\zeta,0,n} = g_{0,n}(\rho) + n\,h_{0,n}$. Recall that $g_{0,n}(\rho) = g_{0,n,0} + g_{0,n,1}\rho + g_{0,n,2}\rho^2 + \ldots$, and we can choose $h_{0,n} = -g_{0,n,0}/n$ to obtain

$$
A_{\zeta,m,n} = r^m \; g_{m,n}(\rho), \quad \text{for} \quad m = 0, n \neq 0, \quad \text{with} \quad g_{m,n}(0) = 0.
\tag{23}
$$

- For the $m = 0$, $n = 0$ harmonic of $A_\zeta$, we have $A_{\zeta,0,0} = g_{0,0}(\rho) + h_{0,0}$. Similarly, choose $h_{0,0} = -g_{0,n,0}$ to obtain

$$
A_{\zeta,m,m} = r^m \; g_{m,n}(\rho), \quad \text{for} \quad m = 0, n = 0, \quad \text{with} \quad g_{m,n}(0) = 0.
\tag{24}
$$

- To simplify the algorithmic implementation of these conditions, we shall introduce a "regularization" factor, $\rho^{m/2} = r^m$.

- Note that the representation for $A_{\theta,m,n}$ given in Eqn. (19), with an arbitrary polynomial $f_{m,n}(\rho) = f_{m,n,0} + f_{m,n,1}\rho + f_{m,n,2}\rho^2 + \ldots$, is equivalent to $A_{\theta,m,n} = \rho^{m/2}\alpha_{m,n}(\rho)$ where $\alpha_{m,n}(\rho)$ is an arbitrary polynomial with the constraint $\alpha_{m,n}(0) = 0$.

- We can write the vector potential as

$$
A_{\theta,m,n} = \rho^{m/2}\alpha_{m,n}(\rho), \quad \text{with} \quad \alpha_{m,n}(0) = 0 \;\; \text{for all} \;\; (m,n),
\tag{25}
$$

$$
A_{\zeta,m,n} = \rho^{m/2}\beta_{m,n}(\rho), \quad \text{with} \quad \beta_{m,n}(0) = 0 \;\; \text{for} \;\; m = 0.
\tag{26}
$$

### 3.5.2 non-stellarator symmetric terms

- Just guessing, for the non-stellarator-symmetric configurations,

$$
\begin{aligned}
A_{\theta,m,n} &= \rho^{m/2}\alpha_{m,n}(\rho), \quad \text{with } \alpha_{m,n}(0) = 0 \text{ for all } (m,n), & (27)\\
A_{\zeta,m,n} &= \rho^{m/2}\beta_{m,n}(\rho), \quad \text{with } \beta_{m,n}(0) = 0 \text{ for } m = 0. & (28)
\end{aligned}
$$

# 4  Todo List

**Subprogram bnorml (mn, Ntz, efmn, ofmn)**

There is a very clumsy attempt to parallelize this which could be greatly improved.

**Subprogram casing (teta, zeta, gBn, icasing)**

It would be MUCH faster to only require the tangential field on a regular grid!!!

Please check why $B_s$ is not computed. Is it because $B_s \nabla s \times \mathbf{n} = 0$ ?

This needs to be revised.

**Subprogram curent (lvol, mn, Nt, Nz, iflag, ldItGp)**

Perhaps this can be proved analytically; in any case it should be confirmed numerically.

**Subprogram inputlist::lconstraint**

if `Lconstraint==2`, under reconstruction.

**Subprogram inputlist::wbuild_vector_potential**

: what is this?

**Subprogram ma02aa (lvol, NN)**

If `Lconstraint = 2`, then $\mu = \boldsymbol{\mu}_1$ is varied in order to satisfy the helicity constraint, and $\Delta\psi_p = \boldsymbol{\mu}_2$ is *not* varied, and `Nxdof=1`. (under re-construction)

**Subprogram pc00aa (NGdof, position, Nvol, mn, ie04dgf)**

Unfortunately, $E04DGF$ seems to require approximately $3N$ function evaluations before proceeding to minimize the energy functional, where there are $N$ degrees of freedom. I don't know how to turn this off!

**Subprogram pc00ab (mode, NGdof, Position, Energy, Gradient, nstate, iuser, ruser)**

IT IS VERY LIKELY THAT FFTs WOULD BE FASTER!!!

**Subprogram stzxyz (lvol, stz, RpZ)**

Please see co01aa() for documentation.

**Subprogram ending**

The following belongs to the docs of the program xspech, not to the ending() subroutine. If you know how to attach the docs to the program xspech, please fix this.

If `Lminimize.eq.1`, call pc00aa() to find minimum of energy functional using quasi-Newton, preconditioned conjugate gradient method, `E04DGF`

# 5  Module Index

## 5.1  Modules

Here is a list of all modules:

# 6 Data Type Index

## 6.1 Data Types List

Here are the data types with brief descriptions:

# 7 File Index

## 7.1 File List

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

# 8 Module Documentation

## 8.1 Diagnostics to check the code

**Functions/Subroutines**

- subroutine bfield (zeta, st, Bst)

    *Compute the magnetic field.*

- subroutine hesian (NGdof, position, Mvol, mn, LGdof)

    *Computes eigenvalues and eigenvectors of derivative matrix, $\nabla_\xi \mathbf{F}$.*

- subroutine jo00aa (lvol, Ntz, lquad, mn)

    *Measures error in Beltrami equation, $\nabla \times \mathbf{B} - \mu \mathbf{B}$.*

- subroutine pp00aa

    *Constructs Poincaré plot and "approximate" rotational-transform (driver).*

- subroutine pp00ab (lvol, sti, Nz, nPpts, poincaredata, fittedtransform, utflag)

    *Constructs Poincaré plot and "approximate" rotational-transform (for single field line).*

- subroutine stzxyz (lvol, stz, RpZ)

    *Calculates coordinates, $\mathbf{x}(s, \theta, \zeta) \equiv R \, \mathbf{e}_R + Z \, \mathbf{e}_Z$, and metrics, at given $(s, \theta, \zeta)$.*

### 8.1.1 Detailed Description

### 8.1.2 Function/Subroutine Documentation

#### 8.1.2.1 bfield() `subroutine bfield (`
```
        real, intent(in) zeta,
        real, dimension(1:node), intent(in) st,
        real, dimension(1:node), intent(out) Bst )
```

Compute the magnetic field.

Returns the magnetic field field line equations, $d\mathbf{x}/d\phi = \mathbf{B}/B^\phi$ .

**Equations of field line flow**

- The equations for the fieldlines are normalized to the toroidal field, i.e.

$$\dot{s} \equiv \frac{B^s}{B^\zeta}, \qquad \dot{\theta} \equiv \frac{B^\theta}{B^\zeta}. \tag{29}$$

**Representation of magnetic field**

- The components of the vector potential, $\mathbf{A} = A_\theta \nabla + A_\zeta \nabla \zeta$, are

$$A_\theta(s,\theta,\zeta) = \sum_{i,l} A_{\theta,e,i,l}\, \overline{T}_{l,i}(s) \cos\alpha_i + \sum_{i,l} A_{\theta,o,i,l}\, \overline{T}_{l,i}(s) \sin\alpha_i, \tag{30}$$

$$A_\zeta(s,\theta,\zeta) = \sum_{i,l} A_{\zeta,e,i,l}\, \overline{T}_{l,i}(s) \cos\alpha_i + \sum_{i,l} A_{\zeta,o,i,l}\, \overline{T}_{l,i}(s) \sin\alpha_i, \tag{31}$$

where $\overline{T}_{l,i}(s) \equiv \bar{s}^{m_i/2}\, T_l(s)$, $T_l(s)$ is the Chebyshev polynomial, and $\alpha_j \equiv m_j\theta - n_j\zeta$. The regularity factor, $\bar{s}^{m_i/2}$, where $\bar{s} \equiv (1+s)/2$, is only included if there is a coordinate singularity in the domain (i.e. only in the innermost volume, and only in cylindrical and toroidal geometry.)

- The magnetic field, $\sqrt{g}\,\mathbf{B} = \sqrt{g}B^s \mathbf{e}_s + \sqrt{g}B^\theta \mathbf{e}_\theta + \sqrt{g}B^\zeta \mathbf{e}_\zeta$, is

$$
\begin{aligned}
\sqrt{g}\,\mathbf{B} = \; & \mathbf{e}_s && \sum_{i,l}[( & -m_i A_{\zeta,e,i,l} & - & n_i A_{\theta,e,i,l} & )\overline{T}_{l,i}\sin\alpha_i + ( & +m_i A_{\zeta,o,i,l} & + & n_i A_{\theta,o,i,l} & )\overline{T}_{l,i}\cos\alpha_i] \\
+ \; & \mathbf{e}_\theta && \sum_{i,l}[( & & - & A_{\zeta,e,i,l} & )\overline{T}'_{l,i}\cos\alpha_i + ( & & - & A_{\zeta,o,i,l} & )\overline{T}'_{l,i}\sin\alpha_i] \\
+ \; & \mathbf{e}_\zeta && \sum_{i,l}[( & A_{\theta,e,i,l} & & & )\overline{T}'_{l,i}\cos\alpha_i + ( & A_{\theta,o,i,l} & & & )\overline{T}'_{l,i}\sin\alpha_i]
\end{aligned}
\tag{32}
$$

- In Eqn. $(29)$, the coordinate Jacobian, $\sqrt{g}$, cancels. No coordinate metric information is required to construct the fieldline equations from the magnetic vector potential.

IT IS REQUIRED TO SET IVOL THROUGH GLOBAL MEMORY BEFORE CALLING BFIELD.

The format of this subroutine is constrained by the NAG ode integration routines.

**Parameters**

| in | *zeta* | toroidal angle $\zeta$ |
|------|--------|------------------------|
| in | *st* | radial coordinate $s$ and poloidal angle $\theta$ |
| out | *Bst* | tangential magnetic field directions $B_s, B_\theta$ |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::cpus, allglobal::gbzeta, get_↩
cheby(), get_zernike(), constants::half, allglobal::halfmm, allglobal::im, allglobal::in, allglobal::ivol, allglobal↩
::lcoordinatesingularity, inputlist::lrad, allglobal::mn, inputlist::mpol, allglobal::mvol, allglobal::myid, allglobal↩
::ncpu, allglobal::node, allglobal::notstellsym, constants::one, fileunits::ounit, allglobal::regumm, numerical::small,
constants::two, numerical::vsmall, inputlist::wmacros, and constants::zero.

Referenced by bfield_tangent(), jo00aa(), pp00ab(), and sphdf5::write_grid().

Here is the call graph for this function:
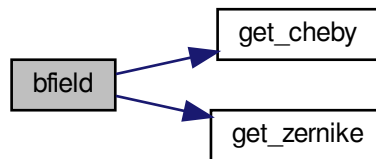
Here is the caller graph for this function:
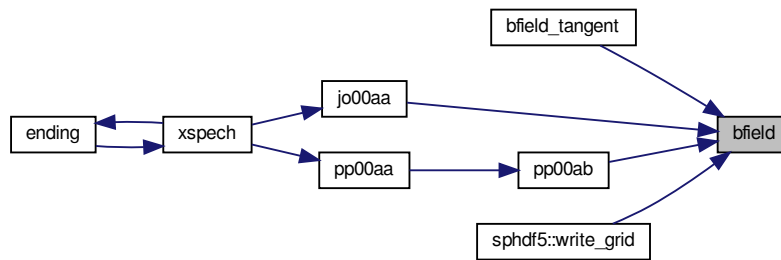


**8.1.2.2 hesian()** `subroutine hesian (`
　　　`integer, intent(in)` *NGdof,*
　　　`real, dimension(0:ngdof)` *position,*
　　　`integer, intent(in)` *Mvol,*
　　　`integer, intent(in)` *mn,*
　　　`integer, intent(in)` *LGdof )*

Computes eigenvalues and eigenvectors of derivative matrix, $\nabla_\xi \mathbf{F}$.

**Parameters**

| in | *NGdof* | number of global degrees of freedom |
|---|---|---|
| in,out | *position* | internal geometrical degrees of freedom |
| in | *Mvol* | total number of volumes in computation |
| in | *mn* | number of Fourier harmonics |
| in | *LGdof* | what is this? |

**construction of Hessian matrix**

- The routine dforce() is used to compute the derivatives, with respect to interface geometry, of the force imbalance harmonics, $[[p + B^2/2]]_j$, which may be considered to be the "physical" constraints, and if `Igeometry==3` then also the derivatives of the "artificial" spectral constraints, $I_j \equiv (R_\theta X + Z_\theta Y)_j$.

- The input variable `Lconstraint` determines how the enclosed fluxes, $\Delta\psi_t$ and $\Delta\psi_p$, and the helicity multiplier, $\mu$, vary as the geometry is varied; see global.f90 and mp00ac() for more details.

**construction of eigenvalues and eigenvectors**

- If `LHevalues==T` then the eigenvalues of the Hessian are computed using the NAG routine `F02EBF`.

- If `LHevectors==T` then the eigenvalues *and* the eigenvectors of the Hessian are computed.

- Note that if `Igeometry==3`, then the derivative-matrix also contains information regarding how the "artificial" spectral constraints vary with geometry; so, the eigenvalues and eigenvectors are not purely "physical".

- The eigenvalues and eigenvectors (if required) are written to the file .ext.GF.ev as follows:

```
open(hunit,file="."//trim(ext)//".GF.ev",status="unknown",form="unformatted")
write(hunit)ngdof,ldvr,ldvi        ! integers; if only the eigenvalues were computed then Ldvr=Ldvi=1;
write(hunit)evalr(1:ngdof)         ! reals  ; real      part of eigenvalues;
write(hunit)evali(1:ngdof)         ! reals  ; imaginary part of eigenvalues;
write(hunit)evecr(1:ngdof,1:ngdof) ! reals  ; real      part of eigenvalues; only if Ldvr=NGdof;
write(hunit)eveci(1:ngdof,1:ngdof) ! reals  ; imaginary part of eigenvalues; only if Ldvi=NGdof;
close(hunit)
```

- The eigenvectors are saved in columns of `evecr`, `eveci`, as described by the NAG documentation for `F02EBF`.

References allglobal::cpus, allglobal::dbbdmp, allglobal::dbbdrz, allglobal::dessian, allglobal::dffdrz, dforce(), allglobal::dmupfdx, inputlist::dpp, inputlist::dqq, allglobal::drbc, allglobal::drbs, allglobal::dzbc, allglobal::dzbs, allglobal::energy, inputlist::ext, constants::half, inputlist::helicity, allglobal::hessian, fileunits::hunit, inputlist↩ ::igeometry, allglobal::im, allglobal::in, allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, allglobal↩ ::lbbintegral, inputlist::lcheck, inputlist::lfindzero, inputlist::lfreebound, allglobal::lhessianallocated, inputlist↩ ::lhevalues, inputlist::lhevectors, inputlist::lhmatrix, allglobal::localconstraint, inputlist::lperturbed, inputlist::mu, fileunits::munit, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, inputlist::nvol, constants::one, fileunits::ounit, packxi(), inputlist::pflux, preset(), allglobal::psifactor, numerical::small, numerical::sqrtmachprec, constants::ten, constants::two, numerical::vsmall, inputlist::wmacros, allglobal:yesstellsym, and constants::zero.

Referenced by xspech().

Here is the call graph for this function:
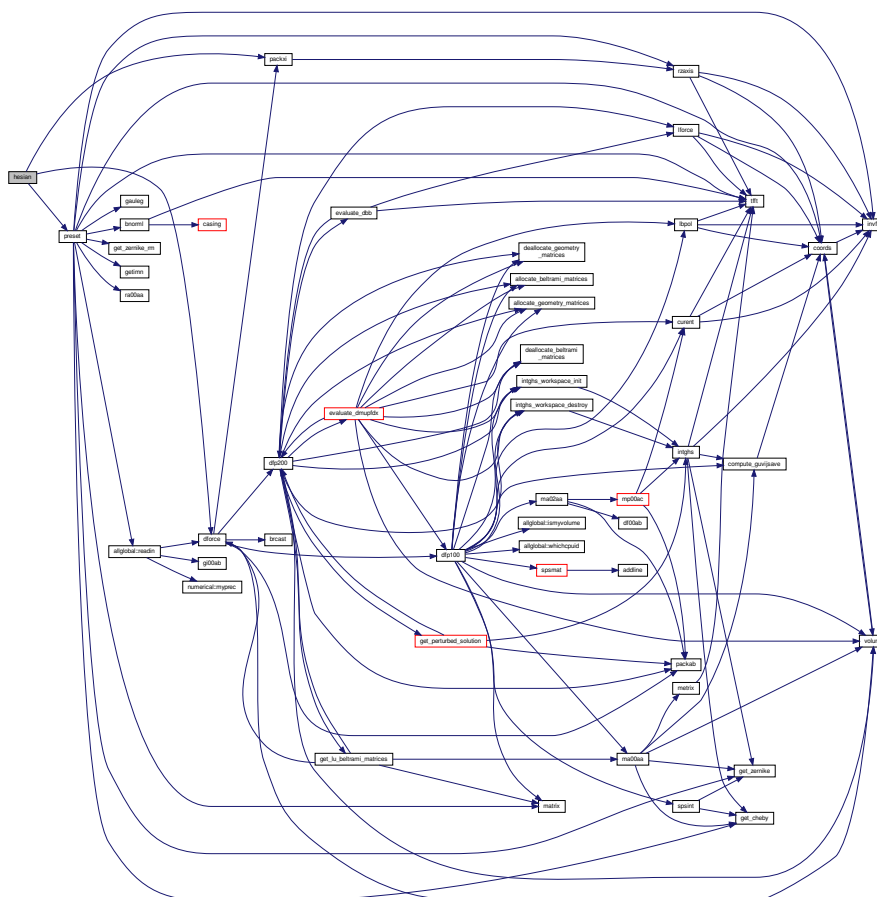
Here is the caller graph for this function:
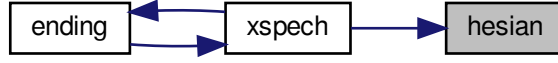


**8.1.2.3 jo00aa()** `subroutine jo00aa (`
```
integer, intent(in) lvol,
integer, intent(in) Ntz,
integer, intent(in) lquad,
integer, intent(in) mn )
```

Measures error in Beltrami equation, $\nabla \times \mathbf{B} - \mu \mathbf{B}$.

This routine is called by xspech() as a post diagnostic and only if `Lcheck==1`.

**construction of current, $\mathbf{j} \equiv \nabla \times \nabla \times \mathbf{A}$**

- The components of the vector potential, $\mathbf{A} = A_\theta \nabla + A_\zeta \nabla \zeta$, are

$$A_\theta(s, \theta, \zeta) \quad = \quad \sum_{i,l} A_{\theta,e,i,l}\, \overline{T}_{l,i}(s) \cos \alpha_i + \sum_{i,l} A_{\theta,o,i,l}\, \overline{T}_{l,i}(s) \sin \alpha_i, \tag{33}$$

$$A_\zeta(s, \theta, \zeta) \quad = \quad \sum_{i,l} A_{\zeta,e,i,l}\, \overline{T}_{l,i}(s) \cos \alpha_i + \sum_{i,l} A_{\zeta,o,i,l}\, \overline{T}_{l,i}(s) \sin \alpha_i, \tag{34}$$

where $\overline{T}_{l,i}(s) \equiv \bar{s}^{m_i/2}\, T_l(s)$, $T_l(s)$ is the Chebyshev polynomial, and $\alpha_j \equiv m_j \theta - n_j \zeta$. The regularity factor, $\bar{s}^{m_i/2}$, where $\bar{s} \equiv (1+s)/2$, is only included if there is a coordinate singularity in the domain (i.e. only in the innermost volume, and only in cylindrical and toroidal geometry.)

- The magnetic field, $\sqrt{g}\,\mathbf{B} = \sqrt{g}B^s \mathbf{e}_s + \sqrt{g}B^\theta \mathbf{e}_\theta + \sqrt{g}B^\zeta \mathbf{e}_\zeta$, is

$$
\begin{aligned}
\sqrt{g}\,\mathbf{B} \quad = \quad &\mathbf{e}_s &&\sum_{i,l}[(\quad -m_i A_{\zeta,e,i,l} \quad - \quad n_i A_{\theta,e,i,l} \quad)\overline{T}_{l,i}\sin\alpha_i + (\quad +m_i A_{\zeta,o,i,l} \quad + \quad n_i A_{\theta,o,i,l} \quad)\overline{T}_{l,i}\cos\alpha_i] \\
+ \quad &\mathbf{e}_\theta &&\sum_{i,l}[(\quad\quad - \quad A_{\zeta,e,i,l} \quad)\overline{T}'_{l,i}\cos\alpha_i + (\quad\quad - \quad A_{\zeta,o,i,l} \quad)\overline{T}'_{l,i}\sin\alpha_i] \\
+ \quad &\mathbf{e}_\zeta &&\sum_{i,l}[(\quad A_{\theta,e,i,l} \quad)\overline{T}'_{l,i}\cos\alpha_i + (\quad A_{\theta,o,i,l} \quad)\overline{T}'_{l,i}\sin\alpha_i]
\end{aligned}
\tag{35}
$$

- The current is

$$\sqrt{g}\,\mathbf{j} = (\partial_\theta B_\zeta - \partial_\zeta B_\theta)\,\mathbf{e}_s + (\partial_\zeta B_s - \partial_s B_\zeta)\,\mathbf{e}_\theta + (\partial_s B_\theta - \partial_\theta B_s)\,\mathbf{e}_\zeta, \tag{36}$$

where (for computational convenience) the covariant components of $\mathbf{B}$ are computed as

$$B_s \quad = \quad (\sqrt{g}B^s)\,g_{ss}/\sqrt{g} + (\sqrt{g}B^\theta)\,g_{s\theta}/\sqrt{g} + (\sqrt{g}B^\zeta)\,g_{s\zeta}/\sqrt{g}, \tag{37}$$

$$B_\theta \quad = \quad (\sqrt{g}B^s)\,g_{s\theta}/\sqrt{g} + (\sqrt{g}B^\theta)\,g_{\theta\theta}/\sqrt{g} + (\sqrt{g}B^\zeta)\,g_{\theta\zeta}/\sqrt{g}, \tag{38}$$

$$B_\zeta \quad = \quad (\sqrt{g}B^s)\,g_{s\zeta}/\sqrt{g} + (\sqrt{g}B^\theta)\,g_{\theta\zeta}/\sqrt{g} + (\sqrt{g}B^\zeta)\,g_{\zeta\zeta}/\sqrt{g}. \tag{39}$$

**quantification of the error**

- The measures of the error are

$$|| \, (\mathbf{j} - \mu \mathbf{B}) \cdot \nabla s || \quad \equiv \quad \int ds \oint\oint d\theta d\zeta \; |\sqrt{g}\,\mathbf{j} \cdot \nabla s - \mu \, \sqrt{g}\, \mathbf{B} \cdot \nabla s| \,, \qquad (40)$$

$$|| \, (\mathbf{j} - \mu \mathbf{B}) \cdot \nabla \theta || \quad \equiv \quad \int ds \oint\oint d\theta d\zeta \; |\sqrt{g}\,\mathbf{j} \cdot \nabla \theta - \mu \, \sqrt{g}\, \mathbf{B} \cdot \nabla \theta| \,, \qquad (41)$$

$$|| \, (\mathbf{j} - \mu \mathbf{B}) \cdot \nabla \zeta || \quad \equiv \quad \int ds \oint\oint d\theta d\zeta \; |\sqrt{g}\,\mathbf{j} \cdot \nabla \zeta - \mu \, \sqrt{g}\, \mathbf{B} \cdot \nabla \zeta| \,. \qquad (42)$$

**comments**

- Is there a better definition and quantification of the error? For example, should we employ an error measure that is dimensionless?

- If the coordinate singularity is in the domain, then $|\nabla \theta| \to \infty$ at the coordinate origin. What then happens to $|| \, (\mathbf{j} - \mu \mathbf{B}) \cdot \nabla \theta ||$ as defined in Eqn. (41)?

- What is the predicted scaling of the error in the Chebyshev-Fourier representation scale with numerical resolution? Note that the predicted error scaling for $E^s$, $E^\theta$ and $E^\zeta$ may not be standard, as various radial derivatives are taken to compute the components of $\mathbf{j}$. (See for example the discussion in Sec.IV.C in Hudson et al. (2011) [5] , where the expected scaling of the error for a finite-element implementation is confirmed numerically.)

- Instead of using Gaussian integration to compute the integral over $s$, an adaptive quadrature algorithm may be preferable.

**Parameters**

| in | *lvol* | in which volume should the Beltrami error be computed |
|----|--------|--------------------------------------------------------|
| in | *Ntz*  | number of grid points in $\theta$ and $\zeta$ |
| in | *lquad* | degree of Gaussian quadrature |
| in | *mn*   | number of Fourier harmonics |

**details of the numerics**

- The integration over $s$ is performed using Gaussian integration, e.g., $\int f(s)ds \approx \sum_k \omega_k f(s_k)$; with the abscissae, $s_k$, and the weights, $\omega_k$, for $k = 1$, `Iquad` $_v$, determined by `CDGQF`. The resolution, $\mathtt{N} \equiv \mathtt{Iquad}$ $_v$, is determined by `Nquad` (see [global.f90](global.f90) and [preset()](preset()) ). A fatal error is enforced by [jo00aa()](jo00aa()) if `CDGQF` returns an $\mathtt{ifail} \neq 0$.

- Inside the Gaussian quadrature loop, i.e. for each $s_k$,
    - The metric elements, $g_{\mu,\nu} \equiv \mathtt{gij(1:6,0,1:Ntz)}$, and the Jacobian, $\sqrt{g} \equiv \mathtt{sg(0,1:Ntz)}$, are calculated on a regular angular grid, $(\theta_i, \zeta_j)$, in [coords()](coords()). The derivatives $\partial_i g_{\mu,\nu} \equiv \mathtt{gij(1:6,i,1{\leftarrow}:Ntz)}$ and $\partial_i \sqrt{g} \equiv \mathtt{sg(i,1:Ntz)}$, with respect to $i \in \{s,\theta,\zeta\}$ are also returned.
    - The Fourier components of the vector potential given in Eqn. (33) and Eqn. (34), and their first and second radial derivatives, are summed.
    - The quantities $\sqrt{g}B^s$, $\sqrt{g}B^\theta$ and $\sqrt{g}B^\zeta$, and their first and second derivatives with respect to $(s,\theta,\zeta)$, are computed on the regular angular grid (using FFTs).
    - The following quantities are then computed on the regular angular grid

$$\sqrt{g}j^s = \sum_u \left[ \partial_\theta(\sqrt{g}B^u)\, g_{u,\zeta} + (\sqrt{g}B^u)\, \partial_\theta g_{u,\zeta} - (\sqrt{g}B^u)g_{u,\zeta}\, \partial_\theta \sqrt{g}/\sqrt{g} \right]/\sqrt{g}$$

$$- \sum_u \left[ \partial_\zeta(\sqrt{g}B^u)\, g_{u,\theta} + (\sqrt{g}B^u)\, \partial_\zeta g_{u,\theta} - (\sqrt{g}B^u)g_{u,\theta}\, \partial_\zeta \sqrt{g}/\sqrt{g} \right]/\sqrt{g}, \qquad (43)$$

$$\sqrt{g}j^{\theta} \quad = \quad \sum_{u} \left[ \partial_{\zeta}(\sqrt{g}B^{u}) \, g_{u,s} + (\sqrt{g}B^{u}) \, \partial_{\zeta}g_{u,s} - (\sqrt{g}B^{u})g_{u,s} \, \partial_{\zeta}\sqrt{g}/\sqrt{g} \right] /\sqrt{g}$$

$$- \quad \sum_{u} \left[ \partial_{s}(\sqrt{g}B^{u}) \, g_{u,\zeta} + (\sqrt{g}B^{u}) \, \partial_{s}g_{u,\zeta} - (\sqrt{g}B^{u})g_{u,\zeta} \, \partial_{s}\sqrt{g}/\sqrt{g} \right] /\sqrt{g}, \qquad (44)$$

$$\sqrt{g}j^{\zeta} \quad = \quad \sum_{u} \left[ \partial_{s}(\sqrt{g}B^{u}) \, g_{u,\theta} + (\sqrt{g}B^{u}) \, \partial_{s}g_{u,\theta} - (\sqrt{g}B^{u})g_{u,\theta} \, \partial_{s}\sqrt{g}/\sqrt{g} \right] /\sqrt{g}$$

$$- \quad \sum_{u} \left[ \partial_{\theta}(\sqrt{g}B^{u}) \, g_{u,s} + (\sqrt{g}B^{u}) \, \partial_{\theta}g_{u,s} - (\sqrt{g}B^{u})g_{u,s} \, \partial_{\theta}\sqrt{g}/\sqrt{g} \right] /\sqrt{g}. \qquad (45)$$

- The error is stored into an array called `beltramierror` which is then written to the HDF5 file in hdfint().

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::beltramierror, bfield(), allglobal↩
::cfmn, allglobal::cheby, coords(), allglobal::cpus, allglobal::dpflux, allglobal::dtflux, allglobal::efmn, inputlist::ext,
allglobal::gbzeta, get_cheby_d2(), get_zernike_d2(), allglobal::guvij, constants::half, inputlist::igeometry, allglobal↩
::im, allglobal::in, invfft(), allglobal::ivol, allglobal::lcoordinatesingularity, inputlist::lerrortype, inputlist::lrad, inputlist↩
::mpol, inputlist::mu, allglobal::mvol, allglobal::myid, inputlist::nfp, allglobal::node, allglobal::notstellsym, allglobal↩
::nt, inputlist::nvol, allglobal::nz, allglobal::ofmn, constants::one, fileunits::ounit, constants::pi2, allglobal::pi2nfp,
allglobal::regumm, allglobal::rij, allglobal::rtt, allglobal::sfmn, allglobal::sg, allglobal::tt, constants::two, inputlist↩
::wmacros, allglobal::zernike, constants::zero, and allglobal::zij.

Referenced by xspech().

Here is the call graph for this function:
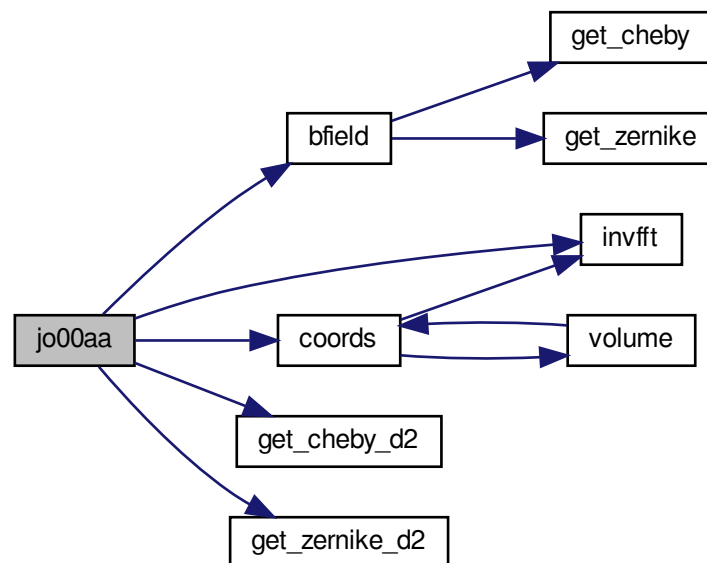
Here is the caller graph for this function:



**8.1.2.4   pp00aa()**   `subroutine pp00aa`

Constructs Poincaré plot and "approximate" rotational-transform (driver).

**relevant input variables**

- The resolution of Poincaré plot is controlled by

  - `nPtraj` trajectories will be located in each volume;

  - `nPpts` iterations per trajectory;

  - `odetol` o.d.e. integration tolerance;

- The magnetic field is given by bfield() .

- The approximate rotational transform is determined, in pp00ab() , by fieldline integration.

**format of output: Poincaré**

- The Poincaré data is written to .ext.poincare:xxxx , where `xxxx` is an integer indicating the volume. The format of this file is as follows:
  ```
  write(svol,'(i4.4)')lvol ! lvol labels volume;
  open(lunit+myid,file="."//trim(ext)//".poincare."//svol,status="unknown",form="unformatted")
  do until end of file
    write(lunit+myid) nz, nppts                 ! integers
    write(lunit+myid) data(1:4,0:nz-1,1:nppts) ! doubles
  enddo
  close(lunit+myid)
  ```

  where

  - $\theta \equiv$ `data(1,k,j)` is the poloidal angle,

  - $s \equiv$ `data(2,k,j)` is the radial coordinate,

  - $R \equiv$ `data(3,k,j)` is the cylindrical $R$,

  - $Z \equiv$ `data(4,k,j)` is the cylindrical $Z$,

- The integer `k=0,Nz-1` labels toroidal planes, so that $\phi = (2\pi/\text{Nfp})(k/\text{Nz})$,

- The integer `j=1,nPpts` labels toroidal iterations.

- Usually (if no fieldline integration errors are encountered) the number of fieldlines followed in volume `lvol` is given by $N + 1$, where the radial resolution, $N \equiv$ `Ni(lvol)` , is given on input. This will be over-ruled by if `nPtrj(lvol)` , given on input, is non-negative.

---

- The starting location for the fieldline integrations are equally spaced in the radial coordinate $s_i = s_{l-1} + i(s_l - s_{l-1})/N$ for $i = 0, N$, along the line $\theta = 0, \zeta = 0$.

**format of output: rotational-transform**

- The rotational-transform data is written to .ext.transform:xxxx , where $xxxx$ is an integer indicating the volume. The format of this file is as follows:

```
open(lunit+myid,file="."//trim(ext)//".sp.t."//svol,status="unknown",form="unformatted")
write(lunit+myid) lnptrj-ioff+1                              ! integer
write(lunit+myid) diotadxup(0:1,0,lvol)                     ! doubles
write(lunit+myid) ( fiota(itrj,1:2), itrj = ioff, lnptrj ) ! doubles
close(lunit+myid)
```

References allglobal::cpus, allglobal::diotadxup, inputlist::ext, constants::half, inputlist::igeometry, inputlist↩
::iota, allglobal::ivol, inputlist::lconstraint, allglobal::lcoordinatesingularity, allglobal::lplasmaregion, inputlist::lrad,
allglobal::lvacuumregion, allglobal::mvol, allglobal::myid, allglobal::ncpu, inputlist::nppts, inputlist::nptrj, inputlist↩
::nvol, allglobal::nz, inputlist::odetol, inputlist::oita, constants::one, fileunits::ounit, constants::pi, allglobal::pi2nfp,
pp00ab(), inputlist::ppts, constants::two, inputlist::wmacros, and constants::zero.

Referenced by xspech().

Here is the call graph for this function:
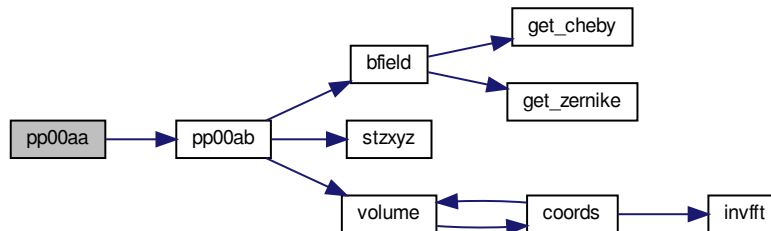


Here is the caller graph for this function:
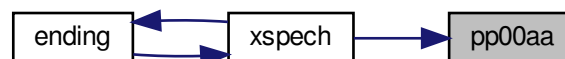
**8.1.2.5 pp00ab()** `subroutine pp00ab (`
        `integer, intent(in)` *lvol,*
        `real, dimension(1:2)` *sti,*
        `integer, intent(in)` *Nz,*
        `integer, intent(in)` *nPpts,*
        `real, dimension(1:4,0:nz-1,1:nppts)` *poincaredata,*
        `real, dimension(1:2)` *fittedtransform,*
        `integer, intent(out)` *utflag )*

Constructs Poincaré plot and "approximate" rotational-transform (for single field line).

**relevant input variables**

- The resolution of Poincaré plot is controlled by

  - `nPpts` iterations per trajectory;
  - `odetol` o.d.e. integration tolerance;

  The magnetic field is given by bfield() .

**rotational-transform**

- The approximate rotational transform is determined by field line integration. This is constructed by fitting a least squares fit to the field line trajectory.

**Parameters**

| in | lvol | |
|---|---|---|
| | sti | |
| in | Nz | |
| in | nPpts | |
| | poincaredata | |
| | fittedtransform | |
| out | utflag | |

References bfield(), allglobal::cpus, allglobal::ivol, allglobal::mvol, allglobal::myid, allglobal::ncpu, allglobal::node, inputlist::nvol, inputlist::odetol, constants::one, fileunits::ounit, constants::pi2, allglobal::pi2nfp, numerical::small, stzxyz(), constants::two, volume(), and constants::zero.

Referenced by pp00aa().

Here is the call graph for this function:
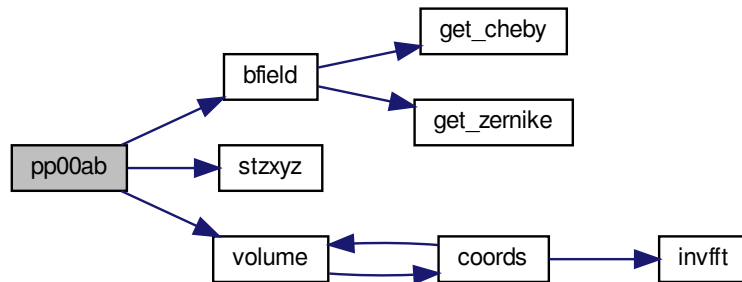


Here is the caller graph for this function:



**8.1.2.6  stzxyz()**  `subroutine stzxyz (`
`            integer, intent(in) lvol,`
`            real, dimension(1:3), intent(in) stz,`
`            real, dimension(1:3), intent(out) RpZ )`

Calculates coordinates, $\mathbf{x}(s,\theta,\zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, at given $(s,\theta,\zeta)$.

- This routine is a "copy" of co01aa(), which calculates the coordinate information on a regular, discrete grid in $\theta$ and $\zeta$ at given $s$ whereas stzxyz() calculates the coordinate information at a single point $(s,\theta,\zeta)$.

- **Todo**  Please see co01aa() for documentation.

**Parameters**

| | | |
|------|-----|---|
| in | *lvol* | |
| in | *stz* | |
| out | *RpZ* | |

References allglobal::cpus, constants::half, allglobal::halfmm, inputlist::igeometry, allglobal::im, allglobal←
::in, allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, allglobal::lcoordinatesingularity, allglobal::mvol,

allglobal::myid, allglobal::notstellsym, inputlist::ntor, inputlist::nvol, constants::one, fileunits::ounit, numerical↩
::vsmall, and constants::zero.

Referenced by pp00ab().

Here is the caller graph for this function:
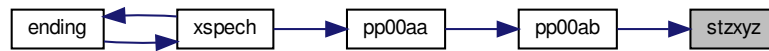
## 8.2 Free-Boundary Computation

**Functions/Subroutines**

- subroutine bnorml (mn, Ntz, efmn, ofmn)

  *Computes* $\mathbf{B}_{Plasma} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta$ *on the computational boundary,* $\partial \mathcal{D}$*.*
- subroutine casing (teta, zeta, gBn, icasing)

  *Constructs the field created by the plasma currents, at an arbitrary, external location using virtual casing.*
- subroutine dvcfield (Ndim, tz, Nfun, vcintegrand)

  *Differential virtual casing integrand.*

### 8.2.1 Detailed Description

### 8.2.2 Function/Subroutine Documentation

#### 8.2.2.1 bnorml()  subroutine bnorml (
```
       integer, intent(in) mn,
       integer, intent(in) Ntz,
       real, dimension(1:mn), intent(out) efmn,
       real, dimension(1:mn), intent(out) ofmn )
```

Computes $\mathbf{B}_{Plasma} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta$ on the computational boundary, $\partial \mathcal{D}$.

**free-boundary constraint**

- The normal field at the computational boundary, $\partial \mathcal{D}$, should be equal to $(\mathbf{B}_P + \mathbf{B}_C) \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta$, where $\mathbf{B}_P$ is the "plasma" field (produced by internal plasma currents) and is computed using virtual casing, and $\mathbf{B}_C$ is the "vacuum" field (produced by the external coils) and is given on input.

- The plasma field, $\mathbf{B}_P$, can only be computed after the equilibrium is determined, but this information is required to compute the equilibrium to begin with; and so there is an iteration involved.

- Suggested values of the vacuum field can be self generated; see xspech() for more documentation on this.

**compute the normal field on a regular grid on the computational boundary**

- For each point on the compuational boundary, casing() is called to compute the normal field produced by the plasma currents.

- **Todo** There is a very clumsy attempt to parallelize this which could be greatly improved.

- An FFT gives the required Fourier harmonics.

See also

  casing.f90

**Parameters**

| in | *mn* | total number of Fourier harmonics |
|---|---|---|
| in | *Ntz* | total number of grid points in $\theta$ and $zeta$ |
| out | *efmn* | even Fourier coefficients |
| out | *ofmn* | odd Fouier coefficients |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, casing(), allglobal::cfmn, allglobal::cpus, allglobal::dxyz, allglobal::globaljk, allglobal::gteta, allglobal::guvij, allglobal::gzeta, constants::half, inputlist←
::igeometry, allglobal::ijimag, allglobal::ijreal, allglobal::im, allglobal::in, allglobal::jiimag, allglobal::jireal, inputlist←
::lcheck, allglobal::lcoordinatesingularity, inputlist::lrad, fileunits::lunit, allglobal::mvol, allglobal::myid, allglobal←
::ncpu, allglobal::notstellsym, allglobal::nt, allglobal::nxyz, allglobal::nz, constants::one, fileunits::ounit, constants←
::pi, constants::pi2, allglobal::pi2nfp, allglobal::rij, allglobal::sfmn, allglobal::sg, numerical::small, constants←
::ten, allglobal::tetazeta, tfft(), allglobal::tt, constants::two, inputlist::vcasingper, inputlist::vcasingtol, allglobal←
::virtualcasingfactor, inputlist::wmacros, constants::zero, and allglobal::zij.

Referenced by preset(), and xspech().

Here is the call graph for this function:
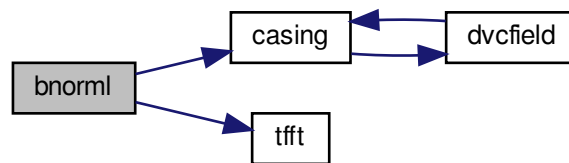


Here is the caller graph for this function:
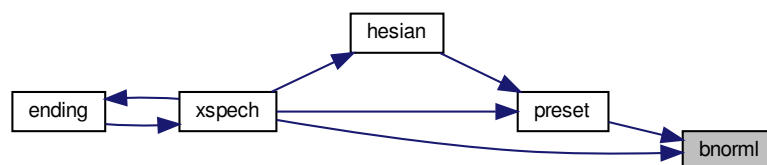
**8.2.2.2 casing()** `subroutine casing (`
```
        real, intent(in) teta,
        real, intent(in) zeta,
        real, intent(out) gBn,
        integer, intent(out) icasing )
```

Constructs the field created by the plasma currents, at an arbitrary, external location using virtual casing.

Compute the external magnetic field using virtual casing.

**Theory and numerics**

- Required inputs to this subroutine are the geometry of the plasma boundary,

$$\mathbf{x}(\theta, \zeta) \equiv x(\theta, \zeta)\mathbf{i} + y(\theta, \zeta)\mathbf{j} + z(\theta, \zeta)\mathbf{k}, \tag{46}$$

and the tangential field on this boundary,

$$\mathbf{B}_s = B^\theta \mathbf{e}_\theta + B^\zeta \mathbf{e}_\zeta, \tag{47}$$

where $\theta$ and $\zeta$ are arbitrary poloidal and toroidal angles, and $\mathbf{e}_\theta \equiv \partial \mathbf{x}/\partial \theta$, $\mathbf{e}_\zeta \equiv \partial \mathbf{x}/\partial \zeta$. This routine assumes that the plasma boundary is a flux surface, i.e. $\mathbf{B} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta = 0$.

- The virtual casing principle (Shafranov & Zakharov (1972) [8], Lazerson (2012) [6] and Hanson (2015) [1]) shows that the field outside/inside the plasma arising from plasma currents inside/outside the boundary is equivalent to the field generated by a surface current,

$$\mathbf{j} = \mathbf{B}_s \times \mathbf{n}, \tag{48}$$

where $\mathbf{n}$ is normal to the surface.

- The field at some arbitrary point, $\bar{\mathbf{x}}$, created by this surface current is given by

$$\mathbf{B}(\bar{\mathbf{x}}) = -\frac{1}{4\pi} \int_{\mathcal{S}} \frac{(\mathbf{B}_s \times d\mathbf{s}) \times \hat{\mathbf{r}}}{r^2}, \tag{49}$$

where $d\mathbf{s} \equiv \mathbf{e}_\theta \times \mathbf{e}_\zeta \, d\theta d\zeta$.

- For ease of notation introduce

$$\mathbf{J} \quad \equiv \quad \mathbf{B}_s \times d\mathbf{s} = \alpha \, \mathbf{e}_\theta - \beta \, \mathbf{e}_\zeta, \tag{50}$$

where $\alpha \equiv B_\zeta = B^\theta g_{\theta\zeta} + B^\zeta g_{\zeta\zeta}$ and $\beta \equiv B_\theta = B^\theta g_{\theta\theta} + B^\zeta g_{\theta\zeta}$.

- We may write in Cartesian coordinates $\mathbf{J} = j_x \, \mathbf{i} + j_y \, \mathbf{j} + j_z \, \mathbf{k}$, where

$$j_x \quad = \quad \alpha \, x_\theta - \beta \, x_\zeta \tag{51}$$
$$j_y \quad = \quad \alpha \, y_\theta - \beta \, y_\zeta \tag{52}$$
$$j_z \quad = \quad \alpha \, z_\theta - \beta \, z_\zeta. \tag{53}$$

- Requiring that the current,

$$\mathbf{j} \quad \equiv \quad \nabla \times \mathbf{B} = \sqrt{g}^{-1}(\partial_\theta B_\zeta - \partial_\zeta B_\theta) \, \mathbf{e}_s + \sqrt{g}^{-1}(\partial_\zeta B_s - \partial_s B_\zeta) \, \mathbf{e}_\theta + \sqrt{g}^{-1}(\partial_s B_\theta - \partial_\theta B_s) \, \mathbf{e}_\zeta, \tag{54}$$

has no normal component to the surface, i.e. $\mathbf{j} \cdot \nabla s = 0$, we obtain the condition $\partial_\theta B_\zeta = \partial_\zeta B_\theta$, or $\partial_\theta \alpha = \partial_\zeta \beta$. In axisymmetric configurations, where $\partial_\zeta \beta = 0$, we must have $\partial_\theta \alpha = 0$.

- The displacement from an arbitrary point, $(X, Y, Z)$, to a point, $(x, y, z)$, that lies on the surface is given

$$\mathbf{r} \equiv r_x \, \mathbf{i} + r_y \, \mathbf{j} + r_z \, \mathbf{k} = (X - x) \, \mathbf{i} + (Y - y) \, \mathbf{j} + (Z - z) \, \mathbf{k}. \tag{55}$$

- The components of the magnetic field produced by the surface current are then

$$B^x = \oiint d\theta d\zeta \ (j_y r_z - j_z r_y)/r^3, \tag{56}$$

$$B^y = \oiint d\theta d\zeta \ (j_z r_x - j_x r_z)/r^3, \tag{57}$$

$$B^z = \oiint d\theta d\zeta \ (j_x r_y - j_y r_x)/r^3 \tag{58}$$

up to a scaling factor `virtualcasingfactor` $= -1/4\pi$ that is taken into account at the end.

- When all is said and done, this routine calculates

$$\int_0^{2\pi} \int_0^{2\pi} \texttt{vcintegrand} \ d\theta d\zeta \tag{59}$$

for a given $(X, Y, Z)$, where `vcintegrand` is given in Eqn. (61).

- The surface integral is performed using `DCUHRE`, which uses an adaptive subdivision strategy and also computes absolute error estimates. The absolute and relative accuracy required are provided by the `inputvar` `vcasingtol`. The minimum number of function evaluations is provided by the `inputvar vcasingits`.

**Calculation of integrand**

- An adaptive integration is used to compute the integrals. Consequently, the magnetic field tangential to the plasma boundary is required at an arbitrary point. This is computed, as always, from $\mathbf{B} = \nabla \times \mathbf{A}$, and this provides $\mathbf{B} = B^\theta \mathbf{e}_\theta + B^\zeta \mathbf{e}_\zeta$. Recall that $B^s = 0$ by construction on the plasma boundary.

  **Todo** It would be MUCH faster to only require the tangential field on a regular grid!!!

- Then, the metric elements $g_{\theta\theta}$, $g_{\theta\zeta}$ and $g_{\zeta\zeta}$ are computed. These are used to "lower" the components of the magnetic field, $\mathbf{B} = B_\theta \nabla \theta + B_\zeta \nabla \zeta$.

  **Todo** Please check why $B_s$ is not computed. Is it because $B_s \nabla s \times \mathbf{n} = 0$ ?

- The distance between the "evaluate" point, $(X, Y, Z)$, and the given point on the surface, $(x, y, z)$ is computed.

- If the computational boundary becomes too close to the plasma boundary, the distance is small and this causes problems for the numerics. I have tried to regularize this problem by introducing $\epsilon \equiv$ `inputvar` `vcasingeps`. Let the "distance" be

$$D \equiv \sqrt{(X-x)^2 + (Y-y)^2 + (Z-Z)^2} + \epsilon^2. \tag{60}$$

- On taking the limit that $\epsilon \to 0$, the virtual casing integrand is

$$\texttt{vcintegrand} \equiv (B_x n_x + B_y n_y + B_z n_z)(1 + 3\epsilon^2/D^2)/D^3, \tag{61}$$

where the normal vector is $\mathbf{n} \equiv n_x \mathbf{i} + n_y \mathbf{j} + n_z \mathbf{k}$. The normal vector, `Nxyz`, to the computational boundary (which does not change) is computed in preset().

  **Todo** This needs to be revised.

**Parameters**

| in | *teta* | $\theta$ |
|------|--------|----------|
| in | *zeta* | $\zeta$ |
| out | *gBn* | $\sqrt{g}\mathbf{B} \cdot \mathbf{n}$ |
| out | *icasing* | return flag from dcuhre() |

References allglobal::cpus, dvcfield(), allglobal::dxyz, inputlist::ext, allglobal::globaljk, allglobal::myid, allglobal↩
::ncpu, allglobal::nxyz, fileunits::ounit, constants::pi, constants::pi2, inputlist::vcasingits, inputlist::vcasingper,
inputlist::vcasingtol, fileunits::vunit, inputlist::wmacros, and constants::zero.

Referenced by bnorml(), and dvcfield().

Here is the call graph for this function:



Here is the caller graph for this function:
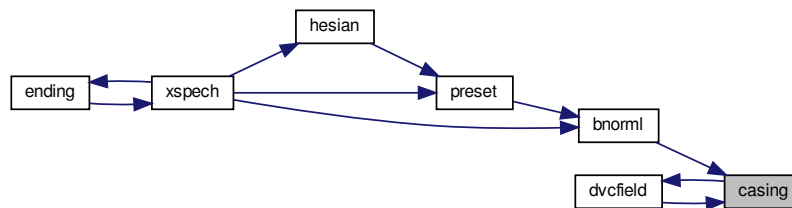


### 8.2.2.3 dvcfield() `subroutine dvcfield (`

```
        integer, intent(in) Ndim,
        real, dimension(1:ndim), intent(in) tz,
        integer, intent(in) Nfun,
        real, dimension(1:nfun), intent(out) vcintegrand )
```

Differential virtual casing integrand.

Differential virtual casing integrand

**Parameters**

| in  | *Ndim*        | number of parameters (==2)             |
|-----|---------------|----------------------------------------|
| in  | *tz*          | $\theta$ and $\zeta$                   |
| in  | *Nfun*        | number of function values (==3)        |
| out | *vcintegrand* | cartesian components of magnetic field |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, casing(), allglobal::cpus, allglobal::dxyz,

allglobal::first_free_bound, constants::four, allglobal::globaljk, constants::half, inputlist::igeometry, allglobal::im, allglobal::in, allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, inputlist::lrad, allglobal::mn, allglobal::mvol, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, inputlist::nvol, allglobal::nxyz, constants::one, fileunits::ounit, allglobal::pi2nfp, numerical::small, constants::three, allglobal::tt, inputlist::vcasingeps, fileunits::vunit, allglobal↩
::yesstellsym, and constants::zero.

Referenced by casing().

Here is the call graph for this function:



Here is the caller graph for this function:
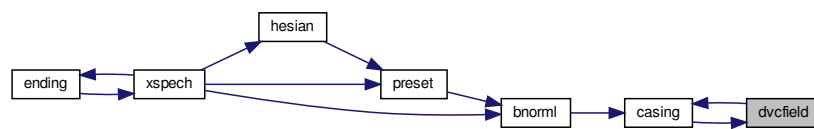
## 8.3 Parallelization

**Functions/Subroutines**

- subroutine brcast (lvol)

    *Broadcasts Beltrami fields, profiles, . . .*

### 8.3.1 Detailed Description

### 8.3.2 Function/Subroutine Documentation

**8.3.2.1 brcast()** `subroutine brcast (`
            `integer, intent(in) lvol )`

Broadcasts Beltrami fields, profiles, . . .

**broadcasting**

- The construction of the Beltrami fields is distributed on separate cpus.

- All "local" information needs to be broadcast so that the "global" force vector,

$$\mathbf{F}_i \equiv [[p + B^2/2]]_i = (p + B^2/2)_{v,i} - (p + B^2/2)_{v-1,i} \tag{62}$$

    can be constructed, and so that restart and output files can be saved to file.

**Parameters**

| in | *lvol* | index of nested volume |
|----|--------|------------------------|

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::bemn, allglobal::bomn, allglobal↩
::cpus, inputlist::curpol, inputlist::curtor, allglobal::diotadxup, allglobal::ditgpdxtp, allglobal::dpflux, allglobal::dtflux,
inputlist::helicity, allglobal::iemn, allglobal::imagneticok, allglobal::iomn, inputlist::lconstraint, inputlist::lfindzero,
inputlist::lrad, allglobal::mn, inputlist::mnvol, inputlist::mu, allglobal::mvol, allglobal::myid, allglobal::ncpu, allglobal↩
::ntz, inputlist::nvol, fileunits::ounit, allglobal::pemn, allglobal::pomn, allglobal::semn, allglobal::somn, inputlist↩
::wmacros, and constants::zero.

Referenced by dforce().

Here is the caller graph for this function:
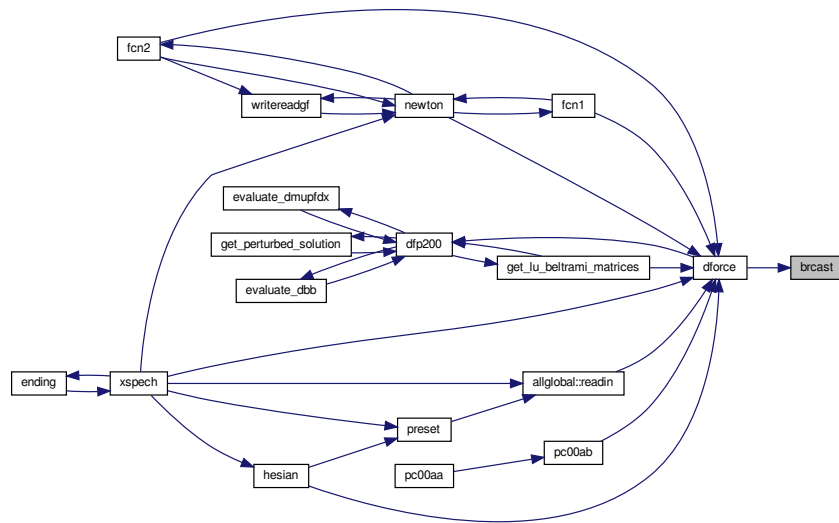
## 8.4 Geometry

### Functions/Subroutines

- subroutine coords (lvol, lss, Lcurvature, Ntz, mn)

    *Calculates coordinates,* $\mathbf{x}(s, \theta, \zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, *and metrics, using FFTs.*

### 8.4.1 Detailed Description

### 8.4.2 Function/Subroutine Documentation

#### 8.4.2.1 coords() 

```
subroutine coords (
    integer, intent(in) lvol,
    real, intent(in) lss,
    integer, intent(in), value Lcurvature,
    integer, intent(in) Ntz,
    integer, intent(in) mn )
```

Calculates coordinates, $\mathbf{x}(s, \theta, \zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, using FFTs.

**Coordinates**

- We work in coordinates, $(s, \theta, \zeta)$, which are be defined *inversely* via a transformation *to* Cartesian coordinates, $(x, y, z)$.

- The toroidal angle, $\zeta$, is identical to the cylindrical angle, $\zeta \equiv \phi$.

- The radial coordinate, $s$, is *not* a global variable: it only needs to be defined in each volume, and in each volume $s \in [-1, 1]$.

- The choice of poloidal angle, $\theta$, does not affect the following.

**Geometry**

- The geometry of the "ideal"-interfaces, $\mathbf{x}_v(\theta, \zeta)$, is given by $R(\theta, \zeta)$ and $Z(\theta, \zeta)$ as follows:

    - `Igeometry=1` : Cartesian

$$\mathbf{x} \;\; \equiv \;\; r_{pol}\theta\,\hat{\mathbf{i}} + r_{tor}\zeta\,\hat{\mathbf{j}} + R\,\hat{\mathbf{k}} \tag{63}$$

    where $r_{pol}$ and $r_{tor}$ are inputs and $r_{pol} = r_{tor} = 1$ by default.

    - `Igeometry=2` : Cylindrical

$$\mathbf{x} \;\; = \;\; R\,\cos\theta\,\hat{\mathbf{i}} + R\,\sin\theta\,\hat{\mathbf{j}} + \zeta\,\hat{\mathbf{k}} \tag{64}$$

    - `Igeometry=3` : Toroidal

$$\mathbf{x} \;\; \equiv \;\; R\,\hat{\mathbf{r}} + Z\,\hat{\mathbf{k}} \tag{65}$$

    where $\hat{\mathbf{r}} \equiv \cos\phi\,\hat{\mathbf{i}} + \sin\phi\,\hat{\mathbf{j}}$ and $\hat{\phi} \equiv -\sin\phi\,\hat{\mathbf{i}} + \cos\phi\,\hat{\mathbf{j}}$.

- The geometry of the ideal interfaces is given as Fourier summation: e.g., for stellarator-symmetry

$$R_v(\theta, \zeta) \equiv \sum_j R_{j,v} \cos \alpha_j, \tag{66}$$

$$Z_v(\theta, \zeta) \equiv \sum_j Z_{j,v} \sin \alpha_j, \tag{67}$$

where $\alpha_j \equiv m_j \theta - n_j \zeta$.

**interpolation between interfaces**

- The "coordinate" functions, $R(s, \theta, \zeta)$ and $Z(s, \theta, \zeta)$, are constructed by radially interpolating the Fourier representations of the ideal-interfaces.

- The $v$-th volume is bounded by $\mathbf{x}_{v-1}$ and $\mathbf{x}_v$.

- In each *annular* volume, the coordinates are constructed by linear interpolation:

$$
\begin{aligned}
R(s, \theta, \zeta) &\equiv \sum_j \left[ \frac{(1-s)}{2} R_{j,v-1} + \frac{(1+s)}{2} R_{j,v} \right] \cos \alpha_j, \\
Z(s, \theta, \zeta) &\equiv \sum_j \left[ \frac{(1-s)}{2} Z_{j,v-1} + \frac{(1+s)}{2} Z_{j,v} \right] \sin \alpha_j,
\end{aligned}
\tag{68}
$$

**coordinate singularity: regularized extrapolation**

- For cylindrical or toroidal geometry, in the innermost, "simple-torus" volume, the coordinates are constructed by an interpolation that "encourages" the interpolated coordinate surfaces to not intersect.

- Introduce $\bar{s} \equiv (s+1)/2$, so that in each volume $\bar{s} \in [0, 1]$, then

$$R_j(s) = R_{j,0} + (R_{j,1} - R_{j,0}) f_j, \tag{69}$$

$$Z_j(s) = Z_{j,0} + (Z_{j,1} - Z_{j,0}) f_j, \tag{70}$$

where, in toroidal geometry,

$$
f_j \equiv \left\{
\begin{array}{ll}
\bar{s} & , \quad \text{for } m_j = 0, \\
\bar{s}^{m_j} & , \quad \text{otherwise.}
\end{array}
\right\}.
\tag{71}
$$

- Note: The location of the coordinate axis, i.e. the $R_{j,0}$ and $Z_{j,0}$, is set in the coordinate "packing" and "unpacking" routine, packxi().

**Jacobian**

- The coordinate Jacobian (and some other metric information) is given by

    - `Igeometry=1` : Cartesian

$$\mathbf{e}_\theta \times \mathbf{e}_\zeta = -r_{tor} R_\theta \, \hat{\mathbf{i}} - r_{pol} R_\zeta \, \hat{\mathbf{j}} + r_{pol} r_{tor} \hat{\mathbf{k}} \tag{72}$$

$$\boldsymbol{\xi} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta = \delta R \tag{73}$$

$$\sqrt{g} = R_s \, r_{pol} \, r_{tor} \tag{74}$$

    - `Igeometry=2` : Cylindrical

$$\mathbf{e}_\theta \times \mathbf{e}_\zeta = (R_\theta \sin \theta + R \cos \theta) \, \hat{\mathbf{i}} + (R \sin \theta - R_\theta \cos \theta) \, \hat{\mathbf{j}} - R R_\zeta \, \hat{\mathbf{k}} \tag{75}$$

$$\boldsymbol{\xi} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta = \delta R \, R \tag{76}$$

$$\sqrt{g} = R_s \, R \tag{77}$$

    – `Igeometry=3` : Toroidal

$$\mathbf{e}_\theta \times \mathbf{e}_\zeta \;=\; -R\,Z_\theta\,\hat{r} + (Z_\theta\,R_\zeta - R_\theta\,Z_\zeta)\hat{\phi} + R\,R_\theta\,\hat{z} \tag{78}$$

$$\boldsymbol{\xi}\cdot\mathbf{e}_\theta \times \mathbf{e}_\zeta \;=\; R(\delta Z\,R_\theta - \delta R\,Z_\theta) \tag{79}$$

$$\sqrt{g} \;=\; R(Z_s\,R_\theta - R_s\,Z_\theta) \tag{80}$$

**cartesian metrics**

- The cartesian metrics are

$$g_{ss} = R_s R_s,\quad g_{s\theta} = R_s R_\theta,\quad g_{s\zeta} = R_s R_\zeta,\quad g_{\theta\theta} = R_\theta R_\theta + r_{pol}^2,\quad g_{\theta\zeta} = R_\theta R_\zeta,\quad g_{\zeta\zeta} = R_\zeta R_\zeta + r_{tor}^2 \tag{81}$$

**cylindrical metrics**

- The cylindrical metrics are

$$g_{ss} = R_s R_s,\quad g_{s\theta} = R_s R_\theta,\quad g_{s\zeta} = R_s R_\zeta,\quad g_{\theta\theta} = R_\theta R_\theta + R^2,\quad g_{\theta\zeta} = R_\theta R_\zeta,\quad g_{\zeta\zeta} = R_\zeta R_\zeta + 1 \tag{82}$$

**logical control**

- The logical control is provided by `Lcurvature` as follows:

    – `Lcurvature=0` : only the coordinate transformation is computed, i.e. only $R$ and $Z$ are calculated, e.g. global()

    – `Lcurvature=1` : the Jacobian, $\sqrt{g}$, and "lower" metrics, $g_{\mu,\nu}$, are calculated , e.g. bnorml(), lforce(), curent(), metrix(), sc00aa()

    – `Lcurvature=2` : the "curvature" terms are calculated, by which I mean the second derivatives of the position vector; this information is required for computing the current, $\mathbf{j} = \nabla \times \nabla \times \mathbf{A}$, e.g. jo00aa()

    – `Lcurvature=3` : the derivative of the $g_{\mu,\nu}/\sqrt{g}$ w.r.t. the interface boundary geometry is calculated, e.g. metrix(), curent()

    – `Lcurvature=4` : the derivative of the $g_{\mu,\nu}$ w.r.t. the interface boundary geometry is calculated, e.g. dforce()

    – `Lcurvature=5` : the derivative of $\sqrt{g}$ w.r.t. the interface boundary geometry is calculated, e.g. rzaxis()

**Parameters**

| in | *lvol* | specified in which volume to compute coordinates |
|----|--------|--------------------------------------------------|
| in | *lss* | radial coordinate $s$ |
| in | *Lcurvature* | logical control flag |
| in | *Ntz* | number of points in $\theta$ and $\zeta$ |
| in | *mn* | number of Fourier harmonics |

References allglobal::cosi, allglobal::cpus, allglobal::dbdx, allglobal::guvij, constants::half, allglobal::halfmm, inputlist::igeometry, allglobal::im, allglobal::in, invfft(), allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, allglobal::lcoordinatesingularity, allglobal::mvol, allglobal::myid, allglobal::notstellsym, allglobal::nt, inputlist::ntor, allglobal::nz, constants::one, fileunits::ounit, constants::pi2, allglobal::pi2nfp, allglobal::rij, inputlist::rpol, inputlist←::rtor, allglobal::sg, allglobal::sini, numerical::small, constants::two, volume(), numerical::vsmall, constants::zero, and allglobal::zij.

Referenced by compute_guvijsave(), curent(), jo00aa(), lbpol(), lforce(), preset(), rzaxis(), volume(), and sphdf5←↩
::write_grid().

Here is the call graph for this function:



Here is the caller graph for this function:

## 8.5 Plasma Currents

**Functions/Subroutines**

- subroutine curent (lvol, mn, Nt, Nz, iflag, ldItGp)

    *Computes the plasma current, $I \equiv \int B_\theta \, d\theta$, and the "linking" current, $G \equiv \int B_\zeta \, d\zeta$.*

### 8.5.1 Detailed Description

### 8.5.2 Function/Subroutine Documentation

#### 8.5.2.1 curent() `subroutine curent (`
```
        integer, intent(in) lvol,
        integer, intent(in) mn,
        integer, intent(in) Nt,
        integer, intent(in) Nz,
        integer, intent(in) iflag,
        real, dimension(0:1,-1:2), intent(out) ldItGp )
```

Computes the plasma current, $I \equiv \int B_\theta \, d\theta$, and the "linking" current, $G \equiv \int B_\zeta \, d\zeta$.

**enclosed currents**

- In the vacuum region, the enclosed currents are given by either surface integrals of the current density or line integrals of the magnetic field,

$$\int_{\mathcal{S}} \mathbf{j} \cdot d\mathbf{s} = \int_{\partial \mathcal{S}} \mathbf{B} \cdot d\mathbf{l}, \tag{83}$$

and line integrals are usually easier to compute than surface integrals.

- The magnetic field is given by the curl of the magnetic vector potential, as described in e.g. bfield() .

- The toroidal, plasma current is obtained by taking a "poloidal" loop, $d\mathbf{l} = \mathbf{e}_\theta \, d\theta$, on the plasma boundary, where $B^s = 0$, to obtain

$$I \equiv \int_0^{2\pi} \mathbf{B} \cdot \mathbf{e}_\theta \, d\theta = \int_0^{2\pi} (-\partial_s A_\zeta \, \bar{g}_{\theta\theta} + \partial_s A_\theta \, \bar{g}_{\theta\zeta}) \, d\theta, \tag{84}$$

where $\bar{g}_{\mu\nu} \equiv g_{\mu\nu}/\sqrt{g}$.

- The poloidal, "linking" current through the torus is obtained by taking a "toroidal" loop, $d\mathbf{l} = \mathbf{e}_\zeta \, d\zeta$, on the plasma boundary to obtain

$$G \equiv \int_0^{2\pi} \mathbf{B} \cdot \mathbf{e}_\zeta \, d\zeta = \int_0^{2\pi} (-\partial_s A_\zeta \, \bar{g}_{\theta\zeta} + \partial_s A_\theta \, \bar{g}_{\zeta\zeta}) \, d\zeta. \tag{85}$$

**Fourier integration**

- Using $f \equiv -\partial_s A_\zeta \, \bar{g}_{\theta\theta} + \partial_s A_\theta \, \bar{g}_{\theta\zeta}$, the integral for the plasma current is

$$I = \sum_i{}' f_i \cos(n_i \zeta) 2\pi, \tag{86}$$

where $\sum'$ includes only the $m_i = 0$ harmonics.

- Using $g \equiv -\partial_s A_\zeta \, \bar{g}_{\theta\zeta} + \partial_s A_\theta \, \bar{g}_{\zeta\zeta}$, the integral for the linking current is

$$G = \sum_i{}' g_i \cos(m_i \zeta) 2\pi, \tag{87}$$

where $\sum'$ includes only the $n_i = 0$ harmonics.

- The plasma current, Eqn. (86), should be independent of $\zeta$, and the linking current, Eqn. (87), should be independent of $\theta$.

  **Todo** Perhaps this can be proved analytically; in any case it should be confirmed numerically.

**Parameters**

| in | *lvol* | index of volume |
|------|--------|------------------------------|
| in | *mn* | number of Fourier harmonics |
| in | *Nt* | number of grid points along $\theta$ |
| in | *Nz* | number of grid points along $\zeta$ |
| in | *iflag* | some integer flag |
| out | *ldItGp* | plasma and linking current |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::cfmn, allglobal::comn, coords(), allglobal::cpus, allglobal::efmn, allglobal::evmn, allglobal::guvij, allglobal::ijimag, allglobal::ijreal, allglobal::im, allglobal::ime, allglobal::in, allglobal::ine, invfft(), allglobal::jiimag, allglobal::jireal, inputlist::lrad, allglobal::mne, allglobal::mvol, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, allglobal::ntz, allglobal::odmn, allglobal::ofmn, constants::one, fileunits::ounit, constants::pi2, allglobal::sfmn, allglobal::sg, allglobal::simn, tfft(), allglobal::tt, constants::two, inputlist::wmacros, allglobal::yesstellsym, and constants::zero.

Referenced by dfp100(), evaluate_dmupfdx(), and mp00ac().

Here is the call graph for this function:

Here is the caller graph for this function:
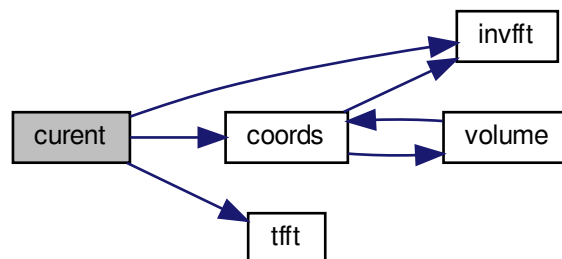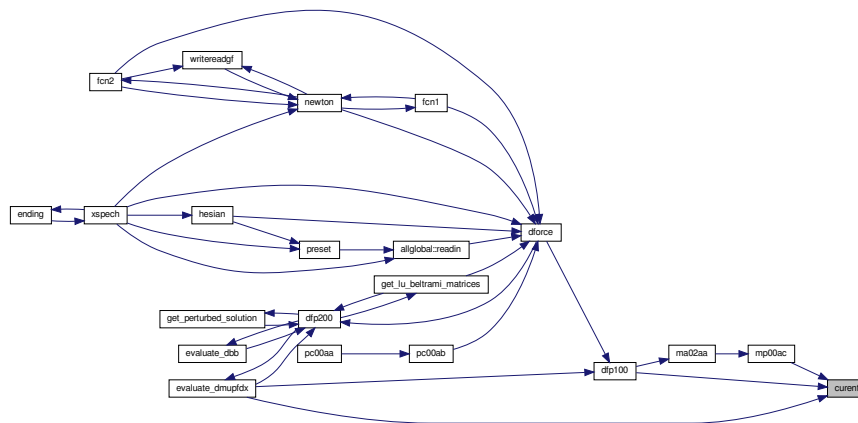
## 8.6 "global" force

**Functions/Subroutines**

- subroutine dforce (NGdof, position, force, LComputeDerivatives, LComputeAxis)

  *Calculates* $\mathbf{F}(\mathbf{x})$, *where* $\mathbf{x} \equiv \{geometry\} \equiv \{R_{i,v}, Z_{i,v}\}$ *and* $\mathbf{F} \equiv [[p + B^2/2]] + \{spectral\ constraints\}$, *and* $\nabla\mathbf{F}$.

### 8.6.1 Detailed Description

### 8.6.2 Function/Subroutine Documentation

**8.6.2.1 dforce()**    subroutine dforce (

```
        integer, intent(in) NGdof,
        real, dimension(0:ngdof), intent(in) position,
        real, dimension(0:ngdof), intent(out) force,
        logical, intent(in) LComputeDerivatives,
        logical LComputeAxis )
```

Calculates $\mathbf{F}(\mathbf{x})$, where $\mathbf{x} \equiv \{geometry\} \equiv \{R_{i,v}, Z_{i,v}\}$ and $\mathbf{F} \equiv [[p + B^2/2]] + \{spectral\ constraints\}$, and $\nabla\mathbf{F}$.

**unpacking**

- The geometrical degrees of freedom are represented as a vector, $\mathbf{x} \equiv \{R_{i,v}, Z_{i,v}\}$, where $i = 1$, mn labels the Fourier harmonic and $v = 1$, Mvol $-1$ is the interface label. This vector is "unpacked" using packxi(). (Note that packxi() also sets the coordinate axis, i.e. the $R_{i,0}$ and $Z_{i,0}$.)

**Matrices computation**

- the volume-integrated metric arrays, DToocc, etc. are evaluated in each volume by calling ma00aa()

- the energy and helicity matrices, dMA(0:NN,0:NN), dMB(0:NN,0:2), etc. are evaluated in each volume by calling matrix()

**parallelization over volumes**

Two different cases emerge: either a local constraint or a global constraint is considered. This condition is determined by the flag LocalConstraint.

- Local constraint

  - In each volume, vvol=1,Mvol ,

    * the logical array ImagneticOK(vvol) is set to .false.
    * The MPI node associated to the volume calls dfp100(). This routine calls ma02aa() (and might iterate on mp00ac()) and computes the field solution in each volume consistent with the constraint.
    * The MPI node associated to the volume calls dfp200(). This computes $p + B^2/2$ (and the spectral constraints if required) at the interfaces in each volumes, as well as the derivatives of the force-balance if LComputeDerivatives=1.

  - After the parallelization loop over the volumes, brcast() is called to broadcast the required information.

- Global constraint
  The MPI node $0$ minimizes the constraint with HYBRID1() by iterating on dfp100() until the field matches the constraint. Other MPI nodes enter the subroutine loop_dfp100(). In loop_dfp100(), each MPI node

  - calls dfp100(),
  - solves the field in its associated volumes,
  - communicates the field to the node $0$ and
  - repeats this loop until the node $0$ sends a flag `iflag=5`.

**broadcasting**

- The required quantities are broadcast by brcast().

**construction of force**

- The force vector, $\mathbf{F}(\mathbf{x})$, is a combination of the pressure-imbalance Fourier harmonics, $[[p+B^2/2]]_{i,v}$, where $i$ labels Fourier harmonic and $v$ is the interface label:

$$F_{i,v} \equiv \left[(p_{v+1} + B_{i,v+1}^2/2) - (p_v + B_{i,v}^2/2)\right] \times \text{BBweight}_i, \tag{88}$$

where BBweight\_i is defined in preset() ; and the spectral condensation constraints,

$$F_{i,v} \equiv I_{i,v} \times \text{epsilon} + S_{i,v,1} \times \text{sweight}_v - S_{i,v+1,0} \times \text{sweight}_{v+1}, \tag{89}$$

where the spectral condensation constraints, $I_{i,v}$, and the "star-like" poloidal angle constraints, $S_{i,v,\pm 1}$, are calculated and defined in lforce() ; and the $\text{sweight}_v$ are defined in preset(). All quantities local to a volume are computed in dfp200(), information is then broadcasted to the MPI node $0$ in dforce() and the global force is evaluated.

**construct derivatives of matrix equation**

- Matrix perturbation theory is used to compute the derivatives of the solution, i.e. the Beltrami fields, as the geometry of the interfaces changes:

**Parameters**

| in | *NGdof* | number of global degrees of freedom |
|---|---|---|
| in | *position* | |
| out | *force* | |
| in | *LComputeDerivatives* | |
| in,out | *LComputeAxis* | |

References brcast(), allglobal::cpus, dfp100(), dfp200(), inputlist::drz, inputlist::epsilon, inputlist::ext, constants::half, inputlist::igeometry, allglobal::iquad, inputlist::lcheck, inputlist::lconstraint, inputlist::lextrap, inputlist::lfreebound, numerical::logtolerance, inputlist::lrad, inputlist::mupftol, allglobal::mvol, allglobal::myid, allglobal::nadof, allglobal←↩
::ncpu, inputlist::ntor, inputlist::nvol, constants::one, fileunits::ounit, packab(), packxi(), constants::pi, constants::pi2, volume(), inputlist::wmacros, and constants::zero.

Referenced by fcn1(), fcn2(), get_lu_beltrami_matrices(), hesian(), newton(), pc00ab(), allglobal::readin(), and xspech().

Here is the call graph for this function:
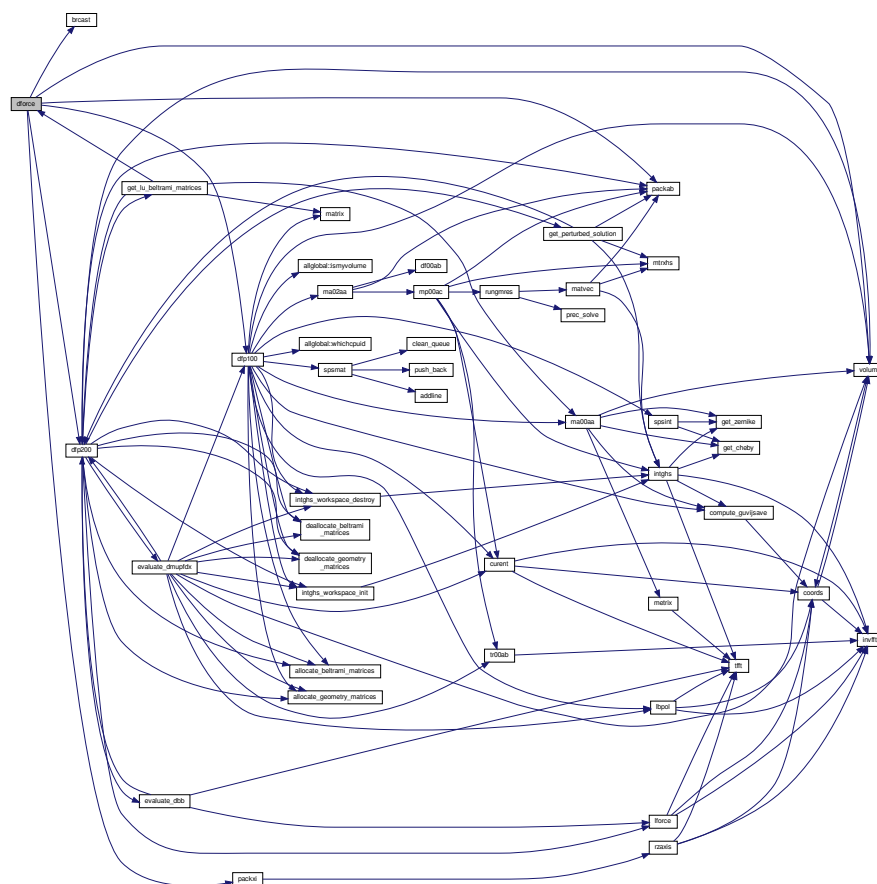


Here is the caller graph for this function:
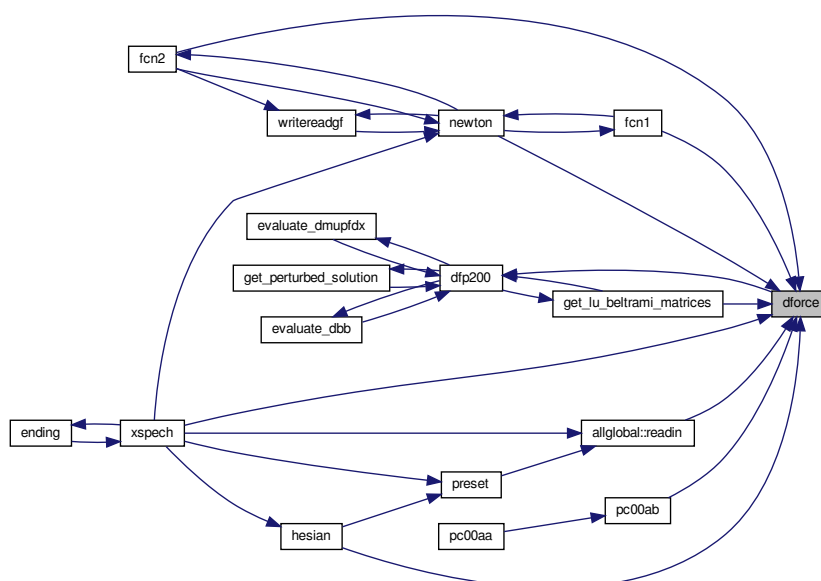
## 8.7 Input namelists and global variables

Collaboration diagram for Input namelists and global variables:
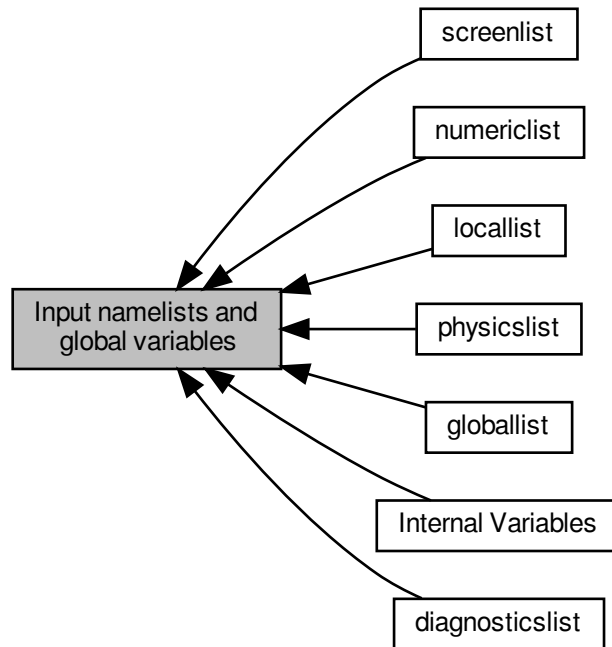


**Modules**

- physicslist

    *The namelist $physicslist$ controls the geometry, profiles, and numerical resolution.*

- numericlist

    *The namelist $numericlist$ controls internal resolution parameters that the user rarely needs to consider.*

- locallist

    *The namelist $locallist$ controls the construction of the Beltrami fields in each volume.*

- globallist

    *The namelist $globallist$ controls the search for global force-balance.*

- diagnosticslist

    *The namelist $diagnosticslist$ controls post-processor diagnostics, such as Poincaré plot resolution, etc.*

- screenlist

    *The namelist $screenlist$ controls screen output. Every subroutine, e.g. $xy00aa.h$, has its own write flag, $Wxy00aa$.*

- Internal Variables

**Modules**

- module constants

    *some constants used throughout the code*
- module numerical

    *platform-dependant numerical resolution*
- module fileunits

    *central definition of file units to avoid conflicts*
- module cputiming

    *timing variables*
- module typedefns

    *type definitions for custom datatypes*
- module allglobal

    *global variable storage used as "workspace" throughout the code*

**Functions/Subroutines**

- subroutine **allglobal::build_vector_potential** (lvol, iocons, aderiv, tderiv)
- subroutine allglobal::whichcpuid (vvol, cpu_id)

    *Returns which MPI node is associated to a given volume.*

**Variables**

- character inputlist::ext

    *The input file is , `ext.sp` , where `ext*100` or `ext.sp*100` is given as command line input.*
- integer, parameter inputlist::mnvol = 256

    *The maximum value of `Nvol` is `MNvol=256`.*
- integer, parameter inputlist::mmpol = 64

    *The maximum value of `Mpol` is `MNpol=64`.*
- integer, parameter inputlist::mntor = 64

    *The maximum value of `Ntor` is `MNtor=64`.*
- integer allglobal::ncpu

    *number of MPI tasks*
- integer allglobal::ismyvolumevalue

    *flag to indicate if a CPU is operating on its assigned volume*
- real allglobal::cpus

    *initial time*
- real allglobal::pi2nfp

    *pi2/nfp ; assigned in readin()*
- real allglobal::pi2pi2nfp

    $4\pi^2 Nfp$
- real allglobal::pi2pi2nfphalf

    $2\pi^2 Nfp$
- real allglobal::pi2pi2nfpquart

    $\pi^2 Nfp$
- real allglobal::forceerr

    *total force-imbalance*
- real allglobal::energy

    *MHD energy.*
- real, dimension(:), allocatable allglobal::ipdt

*Toroidal pressure-driven current.*

- real, dimension(:,:), allocatable allglobal::ipdtdpf

  *Toroidal pressure-driven current.*

- integer allglobal::mvol

  *number of total volumes; equal to Nvol for fixed-boundary; equal to Nvol+1 for free-boundary*

- logical allglobal::yesstellsym

  *internal shorthand copies of Istellsym, which is an integer input;*

- logical allglobal::notstellsym

  *internal shorthand copies of Istellsym, which is an integer input;*

- logical **allglobal::yesmatrixfree**

- logical allglobal::notmatrixfree

  *to use matrix-free method or not*

- real, dimension(:,:), allocatable allglobal::cheby

  *local workspace for evaluation of Chebychev polynomials*

- real, dimension(:,:,:), allocatable allglobal::zernike

  *local workspace for evaluation of Zernike polynomials*

- real, dimension(:,:,:), allocatable allglobal::tt

  *derivatives of Chebyshev polynomials at the inner and outer interfaces;*

- real, dimension(:,:,:,:), allocatable allglobal::rtt

  *derivatives of Zernike polynomials at the inner and outer interfaces;*

- real, dimension(:,:), allocatable allglobal::rtm

  $r^m$ *term of Zernike polynomials at the origin*

- real, dimension(:), allocatable allglobal::zernikedof

  *Zernike degree of freedom for each* $m$.

- logical, dimension(:), allocatable allglobal::imagneticok

  *used to indicate if Beltrami fields have been correctly constructed;*

- logical allglobal::iconstraintok

  *Used to break iteration loops of slaves in the global constraint minimization.*

- real, dimension(:,:), allocatable allglobal::beltramierror

  *to store the integral of* |*curlB-mu*∗*B*| *computed by jo00aa;*

### 8.7.1 Detailed Description

Input namelists.

## 8.8 "local" force

**Functions/Subroutines**

- subroutine lforce (lvol, iocons, ideriv, Ntz, dBB, XX, YY, length, DDl, MMl, iflag)

    *Computes $B^2$, and the spectral condensation constraints if required, on the interfaces, $\mathcal{I}_i$.*

### 8.8.1 Detailed Description

### 8.8.2 Function/Subroutine Documentation

**8.8.2.1 lforce()** `subroutine lforce (`

```
        integer, intent(in) lvol,
        integer, intent(in) iocons,
        integer, intent(in) ideriv,
        integer, intent(in) Ntz,
        real, dimension(1:ntz, -1:2) dBB,
        real, dimension(1:ntz) XX,
        real, dimension(1:ntz) YY,
        real, dimension(1:ntz) length,
        real DDl,
        real MMl,
        integer, intent(in) iflag )
```

Computes $B^2$, and the spectral condensation constraints if required, on the interfaces, $\mathcal{I}_i$.

**field strength**

- The field strength is given by $B^2 = B^s B_s + B^\theta B_\theta + B^\zeta B_\zeta$, and on the interfaces $B^s = 0$ by construction.

- The magnetic field is $\sqrt{g}\,\mathbf{B} = (\partial_\theta A_\zeta - \partial_\zeta A_\theta)\mathbf{e}_s - \partial_s A_\zeta \mathbf{e}_\theta + \partial_s A_\theta \mathbf{e}_\zeta$.

- The covariant components of the field are computed via $B_\theta = B^\theta g_{\theta\theta} + B^\zeta g_{\theta\zeta}$ and $B_\zeta = B^\theta g_{\theta\zeta} + B^\zeta g_{\zeta\zeta}$.

- The expression for $B^2$ is

$$(\sqrt{g})^2 B^2 = A'_\zeta\,A'_\zeta\,g_{\theta\theta} - 2\,A'_\zeta\,A'_\theta\,g_{\theta\zeta} + A'_\theta\,A'_\theta\,g_{\zeta\zeta}, \tag{90}$$

    where the "$\prime$" denotes derivative with respect to $s$.

- The quantity returned is

$$F \equiv \mathtt{pscale} \times \frac{P}{V^\gamma} + \frac{B^2}{2}, \tag{91}$$

    where $P \equiv \mathtt{adiabatic}$ and $V \equiv$ volume.

**spectral constraints**

- In addition to the physical-force-balance constraints, namely that $[[p + B^2/2]] = 0$ across the interfaces, additional angle constraints are required to obtain a unique Fourier representation of the interface geometry.

- Introducing the angle functional: a weighted combination of the "polar" constraint; the normalized, poloidal, spectral width (Hirshman & Meier (1985) [3], Hirshman & Breslau (1998) [2]) the poloidal-angle origin constraint; and the "length" of the angle curves

$$
F \equiv \sum_{i=1}^{N-1} \alpha_i \underbrace{\oint\!\!\!\oint d\theta d\zeta \, \frac{1}{\Theta_{i,\theta}}}_{polar-angle} + \sum_{i=1}^{N-1} \beta_i \underbrace{M_i}_{spectral-width} + \sum_{i=1}^{N-1} \gamma_i \int_0^{2\pi} \frac{1}{2} \left[ Z_i(0,\zeta) - Z_{i,0} \right]^2 d\zeta + \oint\!\!\!\oint d\theta d\zeta \sum_{i=1}^{N} \delta_i \underbrace{L_i}_{poloidal-length}
$$

where $i$ labels the interfaces, and

$$
\Theta_{i,\theta} \equiv \frac{x \, y_\theta - x_\theta \, y}{x^2 + y^2}, \tag{93}
$$

$$
M_i \equiv \frac{\sum_j m_j^p (R_{j,i}^2 + Z_{j,i}^2)}{\sum_j (R_{j,i}^2 + Z_{j,i}^2)}, \tag{94}
$$

$$
L_i \equiv \sqrt{[R_i(\theta,\zeta) - R_{i-1}(\theta,\zeta)]^2 + [Z_i(\theta,\zeta) - Z_{i-1}(\theta,\zeta)]^2}, \tag{95}
$$

and where $j$ labels the Fourier harmonics. The $\alpha_i$, $\beta_i$, $\gamma_i$ and $\delta_i \equiv$ `sweight` are user-supplied weight factors.

- The polar constraint is derived from defining $\tan \Theta \equiv y/x$, where

$$
x(\theta,\zeta) \equiv R_i(\theta,\zeta) - R_{i,0}(\zeta), \tag{96}
$$

$$
y(\theta,\zeta) \equiv Z_i(\theta,\zeta) - Z_{i,0}(\zeta), \tag{97}
$$

and where the geometric center of each interface is given by the arc-length weighted integrals, see rzaxis(),

$$
R_{i,0} \equiv \int_0^{2\pi} d\theta \, R_i(\theta,\zeta) \sqrt{R_{i,\theta}(\theta,\zeta)^2 + Z_{i,\theta}(\theta,\zeta)^2}, \tag{98}
$$

$$
Z_{i,0} \equiv \int_0^{2\pi} d\theta \, Z_i(\theta,\zeta) \sqrt{R_{i,\theta}(\theta,\zeta)^2 + Z_{i,\theta}(\theta,\zeta)^2}, \tag{99}
$$

and $\cos \Theta = x/\sqrt{x^2 + y^2}$ has been used to simplify the expressions and to avoid divide-by-zero.

- Only "poloidal tangential" variations will be allowed to find the extremum of $F$, which are described by

$$
\delta R_i(\theta,\zeta) \equiv R_{i,\theta}(\theta,\zeta) \, \delta u_i(\theta,\zeta), \tag{100}
$$

$$
\delta Z_i(\theta,\zeta) \equiv Z_{i,\theta}(\theta,\zeta) \, \delta u_i(\theta,\zeta), \tag{101}
$$

from which it follows that the variation in each Fourier harmonic is

$$
\delta R_{j,i} = \oint\!\!\!\oint d\theta d\zeta \, R_{i,\theta}(\theta,\zeta) \, \delta u_i(\theta,\zeta) \, \cos(m_j\theta - n_j\zeta), \tag{102}
$$

$$
\delta Z_{j,i} = \oint\!\!\!\oint d\theta d\zeta \, Z_{i,\theta}(\theta,\zeta) \, \delta u_i(\theta,\zeta) \, \sin(m_j\theta - n_j\zeta), \tag{103}
$$

and

$$
\delta R_{i,\theta}(\theta,\zeta) \equiv R_{i,\theta\theta}(\theta,\zeta) \, \delta u_i(\theta,\zeta) + R_{i,\theta}(\theta,\zeta) \, \delta u_{i,\theta}(\theta,\zeta) \tag{104}
$$

$$
\delta Z_{i,\theta}(\theta,\zeta) \equiv Z_{i,\theta\theta}(\theta,\zeta) \, \delta u_i(\theta,\zeta) + Z_{i,\theta}(\theta,\zeta) \, \delta u_{i,\theta}(\theta,\zeta) \tag{105}
$$

- The variation in $F$ is

$$
\begin{aligned}
\delta F = & \sum_{i=1}^{N-1} \alpha_i \oint\!\!\!\oint d\theta d\zeta \left( \frac{-2\Theta_{i,\theta\theta}}{\Theta_{i,\theta}^2} \right) \delta u_i \\
& + \sum_{i=1}^{N-1} \beta_i \oint\!\!\!\oint d\theta d\zeta \, (R_{i,\theta} X_i + Z_{i,\theta} Y_i) \, \delta u_i \\
& + \sum_{i=1}^{N-1} \gamma_i \int d\zeta \, (Z_i(0,\zeta) - Z_{i,0}) \, Z_{i,\theta} \, \delta u_i
\end{aligned}
$$

$$+ \sum_{i=1}^{N-1} \delta_i \oint\oint d\theta d\zeta \left( \frac{\Delta R_i R_{i,\theta} + \Delta Z_i Z_{i,\theta}}{L_i} \right) \delta u_i$$

$$- \sum_{i=1}^{N-1} \delta_{i+1} \oint\oint d\theta d\zeta \left( \frac{\Delta R_{i+1} R_{i,\theta} + \Delta Z_{i+1} Z_{i,\theta}}{L_{i+1}} \right) \delta u_i \tag{106}$$

where, for the stellarator symmetric case,

$$X_i \equiv \sum_j (m_j^p - M_i) R_{j,i} \cos(m_j\theta - n_j\zeta), \tag{107}$$

$$Y_i \equiv \sum_j (m_j^p - M_i) Z_{j,i} \sin(m_j\theta - n_j\zeta), \tag{108}$$

and

$$\Delta R_i \equiv R_i(\theta,\zeta) - R_{i-1}(\theta,\zeta), \tag{109}$$

$$\Delta Z_i \equiv Z_i(\theta,\zeta) - Z_{i-1}(\theta,\zeta), \tag{110}$$

- The spectral constraints derived from Eqn. (106) are

$$I_i(\theta,\zeta) \equiv -2\alpha_i \frac{\Theta_{i,\theta\theta}}{\Theta_{i,\theta}^2} + \beta_i (R_{i,\theta} X_i + Z_{i,\theta} Y_i) + \gamma_i (Z_i(0,\zeta) - Z_{i,0}) Z_{i,\theta}(0,\zeta)$$

$$+ \delta_i \frac{\Delta R_i R_{i,\theta} + \Delta Z_i Z_{i,\theta}}{L_i} - \delta_{i+1} \frac{\Delta R_{i+1} R_{i,\theta} + \Delta Z_{i+1} Z_{i,\theta}}{L_{i+1}} \tag{111}$$

- Note that choosing $p = 2$ gives $X = -R_{\theta\theta}$ and $Y = -Z_{\theta\theta}$, and the spectrally condensed angle constraint, $R_\theta X + Z_\theta Y = 0$, becomes $\partial_\theta(R_\theta^2 + Z_\theta^2) = 0$, which defines the equal arc length angle.

- The poloidal-angle origin term, namely $\gamma_i (Z_i(0,\zeta) - Z_{i,0}) Z_{i,\theta}(0,\zeta)$ is only used to constrain the $m_j = 0$ harmonics.

- The construction of the angle functional was influenced by the following considerations:

  - The minimal spectral width constraint is very desirable as it reduces the required Fourier resolution, but it does not constrain the $m = 0$ harmonics and the minimizing spectral-width poloidal-angle may not be consistent with the poloidal angle used on adjacent interfaces.

  - The regularization of the vector potential and the coordinate interpolation near the coordinate origin (see elsewhere) assumes that the poloidal angle is the polar angle.

  - The user will provide the Fourier harmonics of the boundary, and thus the user will implicitly define the poloidal angle used on the boundary.

  - Minimizing the length term will ensure that the poloidal angle used on each interface is smoothly connected to the poloidal angle used on adjacent interfaces.

- A suitable choice of the weight factors, $\alpha_i$, $\beta_i$, $\gamma_i$ and $\delta_i$, will ensure that the polar constraint dominates for the innermost surfaces and that this constraint rapidly becomes insignificant away from the origin; that the minimal spectral constraint dominates in the "middle"; and that the minimizing length constraint will be significant near the origin and dominant near the edge, so that the minimizing spectral width angle will be continuously connected to the polar angle on the innermost surfaces and the user-implied angle at the plasma boundary. The length constraint should not be insignificant where the spectral constraint is dominant (so that the $m = 0$ harmonics are constrained).

- The polar constraint does not need normalization. The spectral width constraint has already been normalized. The length constraint is not yet normalized, but perhaps it should be.

- The spectral constraints given in Eqn. (111) need to be differentiated with respect to the interface Fourier harmonics, $R_{j,i}$ and $Z_{j,i}$. The first and second terms lead to a block diagonal hessian, and the length term leads to a block tri-diagonal hessian.

- Including the poloidal-angle origin constraint means that the polar angle constraint can probably be ignored, i.e. $\alpha_i = 0$.

**Parameters**

| | | | |
|---|---|---|---|
| in | *lvol* | |
| in | *iocons* | |
| in | *ideriv* | |
| in | *Ntz* | |
| | *dBB* | |
| | *XX* | |
| | *YY* | |
| | *length* | |
| | *DDl* | |
| | *MMl* | |
| in | *iflag* | |

References inputlist::adiabatic, allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::bemn, allglobal↩
::bomn, allglobal::cfmn, allglobal::comn, coords(), allglobal::cpus, allglobal::drij, allglobal::dzij, allglobal↩
::efmn, allglobal::evmn, inputlist::gamma, allglobal::guvij, constants::half, allglobal::iemn, inputlist::igeometry,
allglobal::ijimag, allglobal::ijreal, allglobal::im, allglobal::in, invfft(), allglobal::iomn, allglobal::irbc, allglobal↩
::irbs, allglobal::irij, allglobal::izbc, allglobal::izbs, allglobal::izij, allglobal::jiimag, allglobal::jireal, inputlist::lcheck,
allglobal::lcoordinatesingularity, inputlist::lrad, allglobal::mmpp, allglobal::mvol, allglobal::myid, allglobal::ncpu,
allglobal::notstellsym, allglobal::nt, inputlist::nvol, allglobal::nz, allglobal::odmn, allglobal::ofmn, constants::one,
fileunits::ounit, allglobal::pemn, allglobal::pomn, inputlist::pscale, allglobal::regumm, allglobal::rtt, allglobal↩
::semn, allglobal::sfmn, allglobal::sg, allglobal::simn, allglobal::somn, tfft(), allglobal::trij, allglobal::tt, constants::two,
allglobal::tzij, allglobal::vvolume, allglobal::yesstellsym, and constants::zero.

Referenced by dfp200(), and evaluate_dbb().

Here is the call graph for this function:
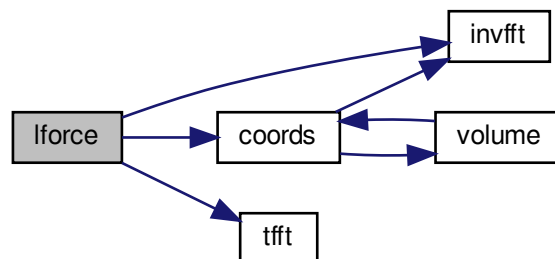
Here is the caller graph for this function:
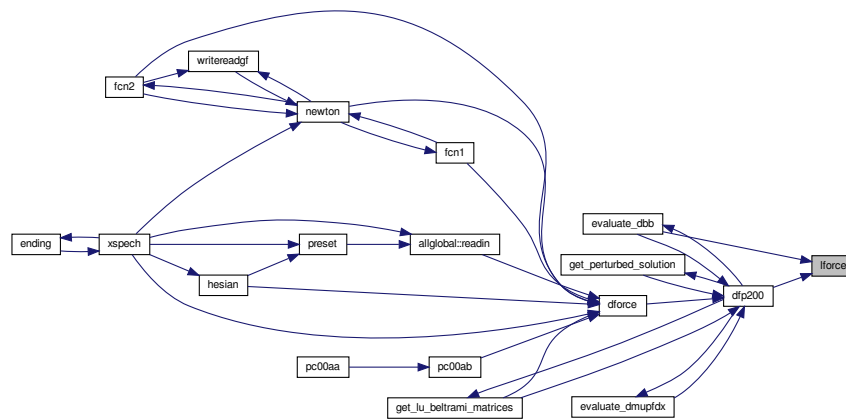
## 8.9 Integrals

**Functions/Subroutines**

- subroutine df00ab (pNN, xi, Fxi, DFxi, Ldfjac, iflag)

  *Evaluates volume integrals, and their derivatives w.r.t. interface geometry, using "packed" format.*
- subroutine ma00aa (lquad, mn, lvol, lrad)

  *Calculates volume integrals of Chebyshev polynomials and metric element products.*

### 8.9.1 Detailed Description

### 8.9.2 Function/Subroutine Documentation

#### 8.9.2.1 df00ab()  subroutine df00ab (
```
        integer, intent(in) pNN,
        real, dimension(0:pnn-1), intent(in) xi,
        real, dimension(0:pnn-1), intent(out) Fxi,
        real, dimension(0:ldfjac-1,0:pnn-1), intent(out) DFxi,
        integer, intent(in) Ldfjac,
        integer, intent(in), value iflag )
```

Evaluates volume integrals, and their derivatives w.r.t. interface geometry, using "packed" format.

**Parameters**

| in | *pNN* | |
|------|--------|--|
| in | *xi* | |
| out | *Fxi* | |
| out | *DFxi* | |
| in | *Ldfjac* | |
| in | *iflag* | |

References allglobal::cpus, allglobal::dma, allglobal::dmd, constants::half, inputlist::helicity, allglobal::myid, inputlist::nvol, constants::one, fileunits::ounit, numerical::small, constants::two, and constants::zero.

Referenced by ma02aa().

Here is the caller graph for this function:
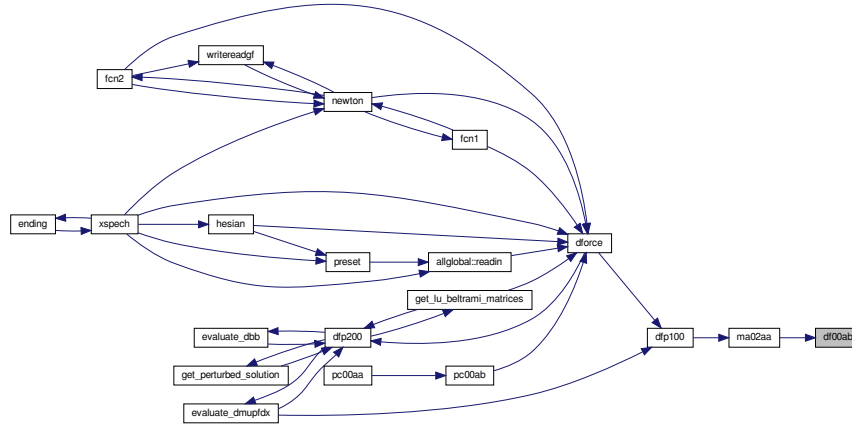


**8.9.2.2   ma00aa()**  `subroutine ma00aa (`
           `integer, intent(in) lquad,`
           `integer, intent(in) mn,`
           `integer, intent(in) lvol,`
           `integer, intent(in) lrad )`

Calculates volume integrals of Chebyshev polynomials and metric element products.

**Chebyshev-metric information**

- The following quantities are calculated:

$$\texttt{DToocc(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}'_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \cos\alpha_i \cos\alpha_j \tag{112}$$

$$\texttt{DToocs(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}'_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \cos\alpha_i \sin\alpha_j \tag{113}$$

$$\texttt{DToosc(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}'_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \sin\alpha_i \cos\alpha_j \tag{114}$$

$$\texttt{DTooss(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}'_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \sin\alpha_i \sin\alpha_j \tag{115}$$

$$\texttt{TTsscc(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \cos\alpha_i \cos\alpha_j\, \bar{g}_{ss} \tag{116}$$

$$\texttt{TTsscs(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \cos\alpha_i \sin\alpha_j\, \bar{g}_{ss} \tag{117}$$

$$\texttt{TTsssc(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \sin\alpha_i \cos\alpha_j\, \bar{g}_{ss} \tag{118}$$

$$\texttt{TTssss(l,p,i,j)} \;\equiv\; \int ds\, \overline{T}_{l,i}\, \overline{T}_{p,j}\; \oint\!\!\oint d\theta d\zeta\;\; \sin\alpha_i \sin\alpha_j\, \bar{g}_{ss} \tag{119}$$

$$\texttt{TDstcc(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \cos\alpha_j\ \bar{g}_{s\theta} \tag{120}$$

$$\texttt{TDstcs(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \sin\alpha_j\ \bar{g}_{s\theta} \tag{121}$$

$$\texttt{TDstsc(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \cos\alpha_j\ \bar{g}_{s\theta} \tag{122}$$

$$\texttt{TDstss(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \sin\alpha_j\ \bar{g}_{s\theta} \tag{123}$$

$$\texttt{TDstcc(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \cos\alpha_j\ \bar{g}_{s\zeta} \tag{124}$$

$$\texttt{TDstcs(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \sin\alpha_j\ \bar{g}_{s\zeta} \tag{125}$$

$$\texttt{TDstsc(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \cos\alpha_j\ \bar{g}_{s\zeta} \tag{126}$$

$$\texttt{TDstss(l,p,i,j)} \equiv \int ds\, \overline{T}_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \sin\alpha_j\ \bar{g}_{s\zeta} \tag{127}$$

$$\texttt{DDstcc(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \cos\alpha_j\ \bar{g}_{\theta\theta} \tag{128}$$

$$\texttt{DDstcs(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \sin\alpha_j\ \bar{g}_{\theta\theta} \tag{129}$$

$$\texttt{DDstsc(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \cos\alpha_j\ \bar{g}_{\theta\theta} \tag{130}$$

$$\texttt{DDstss(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \sin\alpha_j\ \bar{g}_{\theta\theta} \tag{131}$$

$$\texttt{DDstcc(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \cos\alpha_j\ \bar{g}_{\theta\zeta} \tag{132}$$

$$\texttt{DDstcs(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \sin\alpha_j\ \bar{g}_{\theta\zeta} \tag{133}$$

$$\texttt{DDstsc(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \cos\alpha_j\ \bar{g}_{\theta\zeta} \tag{134}$$

$$\texttt{DDstss(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \sin\alpha_j\ \bar{g}_{\theta\zeta} \tag{135}$$

$$\texttt{DDstcc(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \cos\alpha_j\ \bar{g}_{\zeta\zeta} \tag{136}$$

$$\texttt{DDstcs(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \cos\alpha_i \sin\alpha_j\ \bar{g}_{\zeta\zeta} \tag{137}$$

$$\texttt{DDstsc(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \cos\alpha_j\ \bar{g}_{\zeta\zeta} \tag{138}$$

$$\texttt{DDstss(l,p,i,j)} \equiv \int ds\, \overline{T}'_{l,i}\, \overline{T}'_{p,j}\, \oint\!\!\!\oint d\theta d\zeta\ \sin\alpha_i \sin\alpha_j\ \bar{g}_{\zeta\zeta} \tag{139}$$

where $\overline{T}_{l,i} \equiv T_l\, \bar{s}^{m_i/2}$ if the domain includes the coordinate singularity, and $\overline{T}_{l,i} \equiv T_l$ if not; and $\bar{g}_{\mu\nu} \equiv g_{\mu\nu}/\sqrt{g}$.

- The double-angle formulae are used to reduce the above expressions to the Fourier harmonics of $\bar{g}_{\mu\nu}$: see `kija` and `kijs`, which are defined in preset.f90 .

**Parameters**

| in | *lquad* | degree of quadrature |
|----|---------|----------------------|
| in | *mn* | number of Fourier harmonics |
| in | *lvol* | index of nested volume |
| in | *lrad* | order of Chebychev polynomials |

References compute_guvijsave(), allglobal::cpus, allglobal::dbdx, allglobal::ddttcc, allglobal::ddttcs, allglobal↩
::ddttsc, allglobal::ddttss, allglobal::ddtzcc, allglobal::ddtzcs, allglobal::ddtzsc, allglobal::ddtzss, allglobal↩
::ddzzcc, allglobal::ddzzcs, allglobal::ddzzsc, allglobal::ddzzss, allglobal::dtoocc, allglobal::dtoocs, allglobal::dtoosc,
allglobal::dtooss, allglobal::gaussianabscissae, allglobal::gaussianweight, get_cheby(), get_zernike(), allglobal↩
::goomne, allglobal::goomno, allglobal::gssmne, allglobal::gssmno, allglobal::gstmne, allglobal::gstmno, allglobal↩
::gszmne, allglobal::gszmno, allglobal::gttmne, allglobal::gttmno, allglobal::gtzmne, allglobal::gtzmno, allglobal↩
::gzzmne, allglobal::gzzmno, constants::half, allglobal::im, allglobal::in, allglobal::ki, allglobal::kija, allglobal↩
::kijs, allglobal::lcoordinatesingularity, allglobal::lsavedguvij, metrix(), allglobal::mne, inputlist::mpol, allglobal::mvol,
allglobal::myid, allglobal::ncpu, allglobal::notstellsym, constants::one, fileunits::ounit, constants::pi, constants::pi2,
allglobal::pi2pi2nfp, allglobal::pi2pi2nfphalf, allglobal::regumm, allglobal::tdstcc, allglobal::tdstcs, allglobal::tdstsc,
allglobal::tdstss, allglobal::tdszcc, allglobal::tdszcs, allglobal::tdszsc, allglobal::tdszss, allglobal::ttsscc, allglobal↩
::ttsscs, allglobal::ttsssc, allglobal::ttssss, constants::two, volume(), inputlist::wmacros, allglobal::yesstellsym, and
constants::zero.

Referenced by dfp100(), and get_lu_beltrami_matrices().

Here is the call graph for this function:
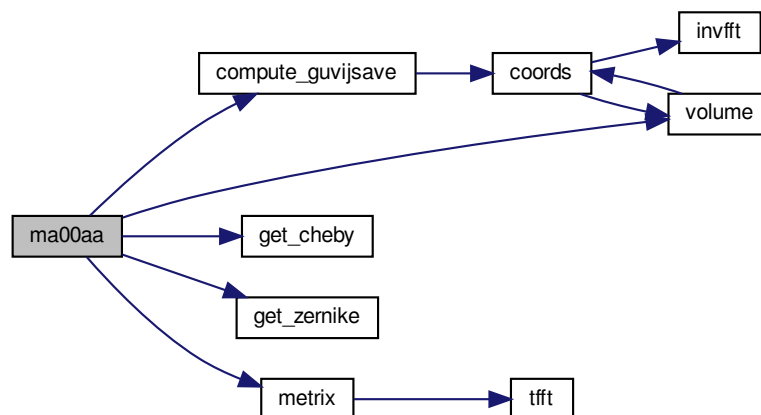
Here is the caller graph for this function:
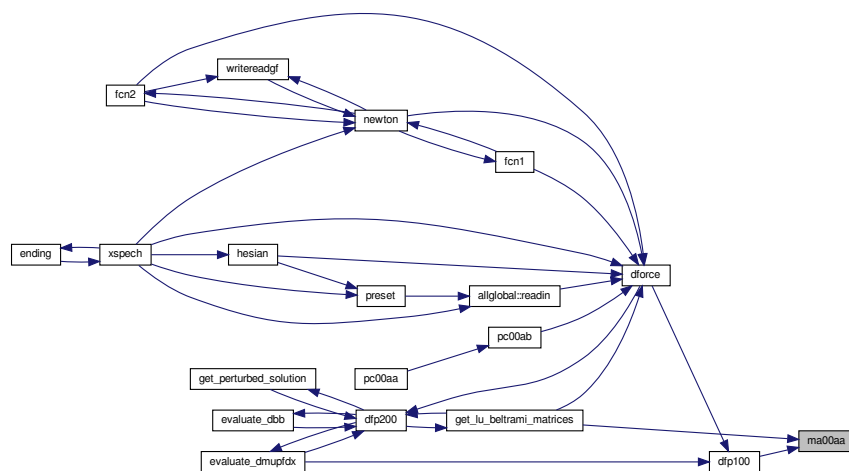
## 8.10 Solver/Driver

**Functions/Subroutines**

- subroutine ma02aa (lvol, NN)

  *Constructs Beltrami field in given volume consistent with flux, helicity, rotational-transform and/or parallel-current constraints.*

### 8.10.1 Detailed Description

### 8.10.2 Function/Subroutine Documentation

#### 8.10.2.1 ma02aa()
```
subroutine ma02aa (
          integer, intent(in) lvol,
          integer, intent(in) NN )
```

Constructs Beltrami field in given volume consistent with flux, helicity, rotational-transform and/or parallel-current constraints.

**Parameters**

| in | *lvol* | index of nested volume for which to run this |
|---|---|---|
| in | *NN* | number of degrees of freedom in the (packed format) vector potential; |

**sequential quadratic programming**

- Only relevant if `LBsequad=T` . See `LBeltrami` for details.

- Documentation on the implementation of `E04UFF` is under construction.

**Newton method**

- Only relevant if `LBnewton=T` . See `LBeltrami` for details.

**linear method**

- Only relevant if `LBlinear=T` . See `LBeltrami` for details.

- The quantity $\mu$ is *not* not treated as a "magnetic" degree-of-freedom equivalent to in the degrees-of-freedom in the magnetic vector potential (as it strictly should be, because it is a Lagrange multiplier introduced to enforce the helicity constraint).

- In this case, the Beltrami equation, $\nabla \times \mathbf{B} = \mu \mathbf{B}$, is *linear* in the magnetic degrees-of-freedom.

- The algorithm proceeds as follows:

  **plasma volumes**

- In addition to the enclosed toroidal flux, $\Delta\psi_t$, which is held constant in the plasma volumes, the Beltrami field in a given volume is assumed to be parameterized by $\mu$ and $\Delta\psi_p$. (Note that $\Delta\psi_p$ is not defined in a torus.)

- These are "packed" into an array, e.g. $\boldsymbol{\mu} \equiv (\mu, \Delta\psi_p)^T$, so that standard library routines , e.g. C05PCF, can be used to (iteratively) find the appropriately-constrained Beltrami solution, i.e. $\mathbf{f}(\boldsymbol{\mu}) = 0$.

- The function $\mathbf{f}(\boldsymbol{\mu})$, which is computed by mp00ac(), is defined by the input parameter Lconstraint:

  * If Lconstraint = -1, 0, then $\boldsymbol{\mu}$ is *not* varied and Nxdof=0.

  * If Lconstraint = 1, then $\boldsymbol{\mu}$ is varied to satisfy the transform constraints; and Nxdof=1 in the simple torus and Nxdof=2 in the annular regions. (Note that in the "simple-torus" region, the enclosed poloidal flux $\Delta\psi_p$ is not well-defined, and only $\mu = \boldsymbol{\mu}_1$ is varied in order to satisfy the transform constraint on the "outer" interface of that volume.)

  * **Todo** If Lconstraint = 2, then $\mu = \boldsymbol{\mu}_1$ is varied in order to satisfy the helicity constraint, and $\Delta\psi_p = \boldsymbol{\mu}_2$ is *not* varied, and Nxdof=1. (under re-construction)

**vacuum volume**

- In the vacuum, $\mu = 0$, and the enclosed fluxes, $\Delta\psi_t$ and $\Delta\psi_p$, are considered to parameterize the family of solutions. (These quantities may not be well-defined if $\mathbf{B} \cdot \mathbf{n} \neq 0$ on the computational boundary.)

- These are "packed" into an array, $\boldsymbol{\mu} \equiv (\Delta\psi_t, \Delta\psi_p)^T$, so that, as above, standard routines can be used to iteratively find the appropriately constrained solution, i.e. $\mathbf{f}(\boldsymbol{\mu}) = 0$.

- The function $\mathbf{f}(\boldsymbol{\mu})$, which is computed by mp00ac(), is defined by the input parameter Lconstraint:

  * If Lconstraint = -1, then $\boldsymbol{\mu}$ is *not* varied and Nxdof=0.

  * If Lconstraint = 0,2, then $\boldsymbol{\mu}$ is varied to satisfy the enclosed current constraints, and Nxdof=2.

  * If Lconstraint = 1, then $\boldsymbol{\mu}$ is varied to satisfy the constraint on the transform on the inner boundary $\equiv$ plasma boundary and the "linking" current, and Nxdof=2.

- The Beltrami fields, and the rotational-transform and helicity etc. as required to determine the function $\mathbf{f}(\boldsymbol{\mu})$ are calculated in mp00ac().

- This routine, mp00ac(), is called iteratively if Nxdof>1 via C05PCF to determine the appropriately constrained Beltrami field, $\mathbf{B}_{\boldsymbol{\mu}}$, so that $\mathbf{f}(\boldsymbol{\mu}) = 0$.

- The input variables mupftol and mupfits control the required accuracy and maximum number of iterations.

- If Nxdof=1, then mp00ac() is called only once to provide the Beltrami fields with the given value of $\boldsymbol{\mu}$.

**debugging: finite-difference confirmation of the derivatives of the rotational-transform**

- Note that the rotational-transform (if required) is calculated by tr00ab(), which is called by mp00ac().

- If Lconstraint=1, then mp00ac() will ask tr00ab() to compute the derivatives of the transform with respect to variations in the helicity-multiplier, $\mu$, and the enclosed poloidal-flux, $\Delta\psi_p$, so that C05PCF may more efficiently find the solution.

- The required derivatives are

$$\frac{\partial \iota}{\partial \mu} \tag{140}$$

$$\frac{\partial \iota}{\partial \Delta\psi_p} \tag{141}$$

to improve the efficiency of the iterative search. A finite difference estimate of these derivatives is available; need DEBUG, Lcheck=2 and Lconstraint=1.

References allglobal::cpus, df00ab(), constants::half, inputlist::helicity, allglobal::im, allglobal::in, allglobal::lblinear, allglobal::lbnewton, allglobal::lbsequad, inputlist::lcheck, inputlist::lconstraint, inputlist::lrad, mp00ac(), inputlist::mu, inputlist::mupfits, inputlist::mupftol, allglobal::mvol, allglobal::myid, allglobal::ncpu, constants::one, fileunits::ounit, packab(), numerical::small, constants::ten, numerical::vsmall, inputlist::wmacros, and constants::zero.

Referenced by dfp100().

Here is the call graph for this function:
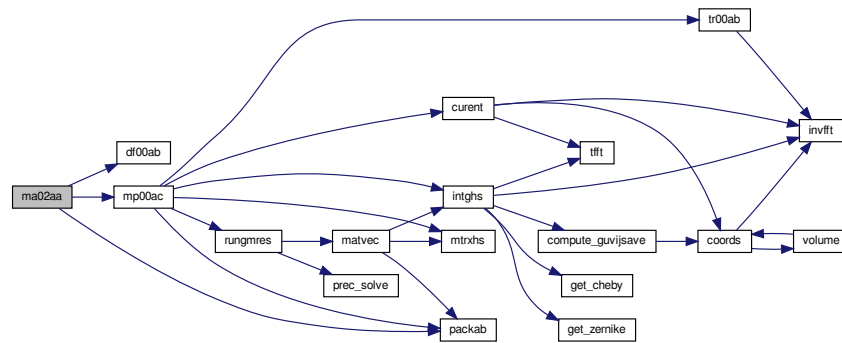


Here is the caller graph for this function:
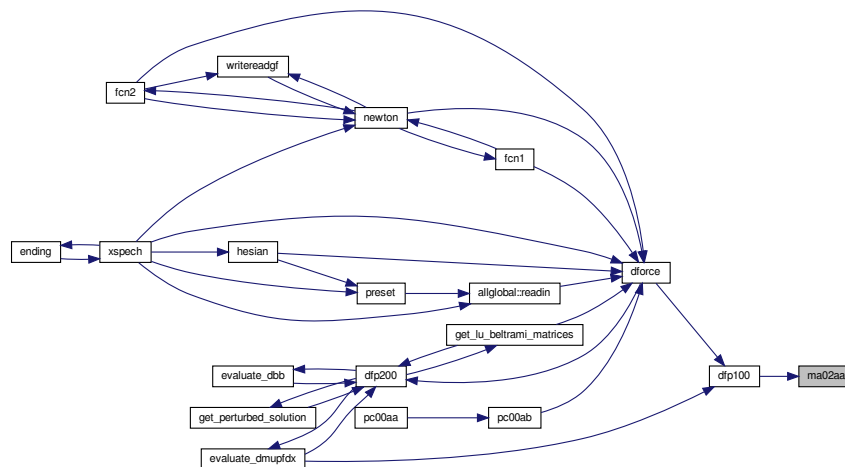
## 8.11 Build matrices

**Functions/Subroutines**

- subroutine matrix (lvol, mn, lrad)

  *Constructs energy and helicity matrices that represent the Beltrami linear system.*
  ***gauge conditions***

### 8.11.1 Detailed Description

### 8.11.2 Function/Subroutine Documentation

#### 8.11.2.1 matrix() subroutine matrix (
            integer, intent(in) *lvol,*
            integer, intent(in) *mn,*
            integer, intent(in) *lrad* )

Constructs energy and helicity matrices that represent the Beltrami linear system.

**gauge conditions**

- In the $v$-th annulus, bounded by the $(v-1)$-th and $v$-th interfaces, a general covariant representation of the magnetic vector-potential is written

$$\bar{\mathbf{A}} = \bar{A}_s \nabla s + \bar{A}_\theta \nabla \theta + \bar{A}_\zeta \nabla \zeta eta. \tag{142}$$

- To this add $\nabla g(s, \theta, \zeta)$, where $g$ satisfies

$$\begin{array}{rcll}
\partial_s g(s, \theta, \zeta) &=& - & \bar{A}_s(s, \theta, \zeta) \\
\partial_\theta g(-1, \theta, \zeta) &=& - & \bar{A}_\theta(-1, \theta, \zeta) \\
\partial_\zeta g(-1, 0, \zeta) &=& - & \bar{A}_\zeta(-1, 0, \zeta).
\end{array} \tag{143}$$

- Then $\mathbf{A} = \bar{\mathbf{A}} + \nabla g$ is given by $\mathbf{A} = A_\theta \nabla \theta + A_\zeta \nabla \zeta$ with

$$A_\theta(-1, \theta, \zeta) = 0 \tag{144}$$
$$A_\zeta(-1, 0, \zeta) = 0 \tag{145}$$

- This specifies the gauge: to see this, notice that no gauge term can be added without violating the conditions in Eqn. (144) or Eqn. (145).

- Note that the gauge employed in each volume is distinct.

**boundary conditions**

- The magnetic field is $\sqrt{g}\,\mathbf{B} = (\partial_\theta A_\zeta - \partial_\zeta A_\theta)\,\mathbf{e}_s - \partial_s A_\zeta\,\mathbf{e}_\theta + \partial_s A_\theta\,\mathbf{e}_\zeta$.

- In the annular volumes, the condition that the field is tangential to the inner interface, $\sqrt{g}\mathbf{B} \cdot \nabla s = 0$ at $s = -1$, gives $\partial_\theta A_\zeta - \partial_\zeta A_\theta = 0$. With the above condition on $A_\theta$ given in Eqn. (144), this gives $\partial_\theta A_\zeta = 0$, which with Eqn. (145) gives

$$A_\zeta(-1, \theta, \zeta) = 0. \tag{146}$$

- The condition at the outer interface, $s = +1$, is that the field is $\sqrt{g}\,\mathbf{B}\cdot\nabla s = \partial_\theta A_\zeta - \partial_\zeta A_\theta = b$, where $b$ is supplied by the user. For each of the plasma regions, $b = 0$. For the vacuum region, generally $b \neq 0$.

**enclosed fluxes**

- In the plasma regions, the enclosed fluxes must be constrained.

- The toroidal and poloidal fluxes enclosed in each volume are determined using

$$\int_S \mathbf{B}\cdot\mathbf{ds} = \int_{\partial S} \mathbf{A}\cdot\mathbf{dl}. \tag{147}$$

**Fourier-Chebyshev representation**

- The components of the vector potential, $\mathbf{A} = A_\theta\nabla + A_\zeta\nabla\zeta eta$, are

$$A_\theta(s,\theta,\zeta) = \sum_{i,l} A_{\theta,e,i,l}\,\overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} A_{\theta,o,i,l}\,\overline{T}_{l,i}(s)\sin\alpha_i, \tag{148}$$

$$A_\zeta(s,\theta,\zeta) = \sum_{i,l} A_{\zeta,e,i,l}\,\overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} A_{\zeta,o,i,l}\,\overline{T}_{l,i}(s)\sin\alpha_i, \tag{149}$$

where $\overline{T}_{l,i}(s) \equiv \bar{s}^{m_i/2}\,T_l(s)$, $T_l(s)$ is the Chebyshev polynomial, and $\alpha_j \equiv m_j\theta - n_j\zeta$. The regularity factor, $\bar{s}^{m_i/2}$, where $\bar{s} \equiv (1+s)/2$, is only included if there is a coordinate singularity in the domain (i.e. only in the innermost volume, and only in cylindrical and toroidal geometry.)

- The magnetic field, $\sqrt{g}\,\mathbf{B} = \sqrt{g}B^s\mathbf{e}_s + \sqrt{g}B^\theta\mathbf{e}_\theta + \sqrt{g}B^\zeta\mathbf{e}_\zeta$, is

$$
\begin{aligned}
\sqrt{g}\,\mathbf{B} = \;&\mathbf{e}_s &&\sum_{i,l}[(\;-m_i A_{\zeta,e,i,l} &&- n_i A_{\theta,e,i,l}&&)\overline{T}_{l,i}\sin\alpha_i + (\;+m_i A_{\zeta,o,i,l} &&+ n_i A_{\theta,o,i,l}&&)\overline{T}_{l,i}\cos\alpha_i]\\
+\;&\mathbf{e}_\theta &&\sum_{i,l}[(\; &&- A_{\zeta,e,i,l}&&)\overline{T}'_{l,i}\cos\alpha_i + (\; &&- A_{\zeta,o,i,l}&&)\overline{T}'_{l,i}\sin\alpha_i]\\
+\;&\mathbf{e}_\zeta &&\sum_{i,l}[(\;A_{\theta,e,i,l} &&&&)\overline{T}'_{l,i}\cos\alpha_i + (\;A_{\theta,o,i,l} &&&&)\overline{T}'_{l,i}\sin\alpha_i]
\end{aligned}
\tag{150}
$$

- The components of the velocity, $\mathbf{v} \equiv v_s\nabla s + v_\theta\nabla\theta + v_\zeta\nabla\zeta eta$, are

$$v_s(s,\theta,\zeta) = \sum_{i,l} v_{s,e,i,l}\,\overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} v_{s,o,i,l}\,\overline{T}_{l,i}(s)\sin\alpha_i, \tag{151}$$

$$v_\theta(s,\theta,\zeta) = \sum_{i,l} v_{\theta,e,i,l}\,\overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} v_{\theta,o,i,l}\,\overline{T}_{l,i}(s)\sin\alpha_i, \tag{152}$$

$$v_\zeta(s,\theta,\zeta) = \sum_{i,l} v_{\zeta,e,i,l}\,\overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} v_{\zeta,o,i,l}\,\overline{T}_{l,i}(s)\sin\alpha_i. \tag{153}$$

**constrained energy functional**

- The constrained energy functional in each volume depends on the vector potential and the Lagrange multipliers,

$$\mathcal{F} \equiv \mathcal{F}[A_{\theta,e,i,l}, A_{\zeta,e,i,l}, A_{\theta,o,i,l}, A_{\zeta,o,i,l}, v_{s,e,i,l}, v_{s,o,i,l}, v_{\theta,e,i,l}, v_{\theta,o,i,l}, v_{\zeta,e,i,l}, v_{\zeta,o,i,l}, \mu, a_i, b_i, c_i, d_i, e_i, f_i, g_1, h_1], \tag{154}$$

and is given by:

$$
\begin{aligned}
\mathcal{F} \quad \equiv \quad & \int \mathbf{B} \cdot \mathbf{B} \, dv + \int \mathbf{v} \cdot \mathbf{v} \, dv - \mu \left[ \int \mathbf{A} \cdot \mathbf{B} \, dv - K \right] \\
+ & \qquad \sum_{i=1} a_i \left[ \sum_l A_{\theta,e,i,l} T_l(-1) - 0 \right] \\
+ & \qquad \sum_{i=1} b_i \left[ \sum_l A_{\zeta,e,i,l} T_l(-1) - 0 \right] \\
+ & \qquad \sum_{i=2} c_i \left[ \sum_l A_{\theta,o,i,l} T_l(-1) - 0 \right] \\
+ & \qquad \sum_{i=2} d_i \left[ \sum_l A_{\zeta,o,i,l} T_l(-1) - 0 \right] \\
+ & \qquad \sum_{i=2} e_i \left[ \sum_l \left( -m_i A_{\zeta,e,i,l} - n_i A_{\theta,e,i,l} \right) T_l(+1) - b_{s,i} \right] \\
+ & \qquad \sum_{i=2} f_i \left[ \sum_l \left( +m_i A_{\zeta,o,i,l} + n_i A_{\theta,o,i,l} \right) T_l(+1) - b_{c,i} \right] \\
+ & \qquad g_1 \left[ \sum_l A_{\theta,e,1,l} T_l(+1) - \Delta\psi_t \right] \\
+ & \qquad h_1 \left[ \sum_l A_{\zeta,e,1,l} T_l(+1) + \Delta\psi_p \right]
\end{aligned}
$$

where

- $a_i$, $b_i$, $c_i$ and $d_i$ are Lagrange multipliers used to enforce the combined gauge and interface boundary condition on the inner interface,
- $e_i$ and $f_i$ are Lagrange multipliers used to enforce the interface boundary condition on the outer interface, namely $\sqrt{g}\,\mathbf{B}\cdot\nabla s = b$; and
- $g_1$ and $h_1$ are Lagrange multipliers used to enforce the constraints on the enclosed fluxes.

- In each plasma volume the boundary condition on the outer interface is $b = 0$.

- In the vacuum volume (only for free-boundary), we may set $\mu = 0$.

## derivatives of magnetic energy integrals

- The first derivatives of $\int dv\,\mathbf{B}\cdot\mathbf{B}$ with respect to $A_{\theta,e,i,l}$, $A_{\theta,o,i,l}$, $A_{\zeta,e,i,l}$ and $A_{\zeta,o,i,l}$ are

$$
\frac{\partial}{\partial A_{\theta,e,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2 \int dv\,\mathbf{B}\cdot\frac{\partial \mathbf{B}}{\partial A_{\theta,e,i,l}} = 2\int dv\,\mathbf{B}\cdot\left[ -n_i \overline{T}_{l,i}\sin\alpha_i\,\mathbf{e}_s + \overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\zeta \right]/\sqrt{g} \tag{156}
$$

$$
\frac{\partial}{\partial A_{\theta,o,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2 \int dv\,\mathbf{B}\cdot\frac{\partial \mathbf{B}}{\partial A_{\theta,o,i,l}} = 2\int dv\,\mathbf{B}\cdot\left[ +n_i \overline{T}_{l,i}\cos\alpha_i\,\mathbf{e}_s + \overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\zeta \right]/\sqrt{g} \tag{157}
$$

$$
\frac{\partial}{\partial A_{\zeta,e,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2 \int dv\,\mathbf{B}\cdot\frac{\partial \mathbf{B}}{\partial A_{\zeta,e,i,l}} = 2\int dv\,\mathbf{B}\cdot\left[ -m_i \overline{T}_{l,i}\sin\alpha_i\,\mathbf{e}_s - \overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\theta \right]/\sqrt{g} \tag{158}
$$

$$
\frac{\partial}{\partial A_{\zeta,o,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2 \int dv\,\mathbf{B}\cdot\frac{\partial \mathbf{B}}{\partial A_{\zeta,o,i,l}} = 2\int dv\,\mathbf{B}\cdot\left[ +m_i \overline{T}_{l,i}\cos\alpha_i\,\mathbf{e}_s - \overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\theta \right]/\sqrt{g} \tag{159}
$$

- The second derivatives of $\int dv\,\mathbf{B}\cdot\mathbf{B}$ with respect to $A_{\theta,e,i,l}$, $A_{\theta,o,i,l}$, $A_{\zeta,e,i,l}$ and $A_{\zeta,o,i,l}$ are

$$
\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+n_j n_i \overline{T}_{p,j}\overline{T}_{l,i} s_j s_i g_{ss} - n_j \overline{T}_{p,j}\overline{T}'_{l,i} s_j c_i g_{s\zeta} - n_i \overline{T}_{l,i}\overline{T}'_{p,j} s_i c_j g_{s\zeta} + \overline{T}'_{p,j}\overline{T}'_{l,i} c_j c_i g_{\zeta\zeta})/\sqrt{}
$$

$$
\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-n_j n_i \overline{T}_{p,j}\overline{T}_{l,i} c_j s_i g_{ss} + n_j \overline{T}_{p,j}\overline{T}'_{l,i} c_j c_i g_{s\zeta} - n_i \overline{T}_{l,i}\overline{T}'_{p,j} s_i s_j g_{s\zeta} + \overline{T}'_{p,j}\overline{T}'_{l,i} s_j c_i g_{\zeta\zeta})/\sqrt{}
$$

$$
\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+m_j n_i \overline{T}_{p,j}\overline{T}_{l,i} s_j s_i g_{ss} - m_j \overline{T}_{p,j}\overline{T}'_{l,i} s_j c_i g_{s\zeta} + n_i \overline{T}_{l,i}\overline{T}'_{p,j} s_i c_j g_{s\theta} - \overline{T}'_{p,j}\overline{T}'_{l,i} c_j c_i g_{\theta\zeta})/\sqrt{}
$$

$$
\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}} \int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-m_j n_i \overline{T}_{p,j}\overline{T}_{l,i} c_j s_i g_{ss} + m_j \overline{T}_{p,j}\overline{T}'_{l,i} c_j c_i g_{s\zeta} + n_i \overline{T}_{l,i}\overline{T}'_{p,j} s_i s_j g_{s\theta} - \overline{T}'_{p,j}\overline{T}'_{l,i} s_j c_i g_{\theta\zeta})/\sqrt{}
$$

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-n_j n_i \overline{T}_{p,j}\overline{T}_{l,i}s_j c_i g_{ss} - n_j \overline{T}_{p,j}\overline{T}'_{l,i}s_j s_i g_{s\zeta} + n_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i c_j g_{s\zeta} + \overline{T}'_{p,j}\overline{T}'_{l,i}c_j s_i g_{\zeta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+n_j n_i \overline{T}_{p,j}\overline{T}_{l,i}c_j c_i g_{ss} + n_j \overline{T}_{p,j}\overline{T}'_{l,i}c_j s_i g_{s\zeta} + n_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i s_j g_{s\zeta} + \overline{T}'_{p,j}\overline{T}'_{l,i}s_j s_i g_{\zeta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-m_j n_i \overline{T}_{p,j}\overline{T}_{l,i}s_j c_i g_{ss} - m_j \overline{T}_{p,j}\overline{T}'_{l,i}s_j s_i g_{s\zeta} - n_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i c_j g_{s\theta} - \overline{T}'_{p,j}\overline{T}'_{l,i}c_j s_i g_{\theta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+m_j n_i \overline{T}_{p,j}\overline{T}_{l,i}c_j c_i g_{ss} + m_j \overline{T}_{p,j}\overline{T}'_{l,i}c_j s_i g_{s\zeta} - n_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i s_j g_{s\theta} - \overline{T}'_{p,j}\overline{T}'_{l,i}s_j s_i g_{\theta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+n_j m_i \overline{T}_{p,j}\overline{T}_{l,i}s_j s_i g_{ss} + n_j \overline{T}_{p,j}\overline{T}'_{l,i}s_j c_i g_{s\theta} - m_i \overline{T}_{l,i}\overline{T}'_{p,j}s_i c_j g_{s\zeta} - \overline{T}'_{p,j}\overline{T}'_{l,i}c_j c_i g_{\theta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-n_j m_i \overline{T}_{p,j}\overline{T}_{l,i}c_j s_i g_{ss} - n_j \overline{T}_{p,j}\overline{T}'_{l,i}c_j c_i g_{s\theta} - m_i \overline{T}_{l,i}\overline{T}'_{p,j}s_i s_j g_{s\zeta} - \overline{T}'_{p,j}\overline{T}'_{l,i}s_j c_i g_{\theta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+m_j m_i \overline{T}_{p,j}\overline{T}_{l,i}s_j s_i g_{ss} + m_j \overline{T}_{p,j}\overline{T}'_{l,i}s_j c_i g_{s\theta} + m_i \overline{T}_{l,i}\overline{T}'_{p,j}s_i c_j g_{s\theta} + \overline{T}'_{p,j}\overline{T}'_{l,i}c_j c_i g_{\theta\theta})/$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-m_j m_i \overline{T}_{p,j}\overline{T}_{l,i}c_j s_i g_{ss} - m_j \overline{T}_{p,j}\overline{T}'_{l,i}c_j c_i g_{s\theta} + m_i \overline{T}_{l,i}\overline{T}'_{p,j}s_i s_j g_{s\theta} + \overline{T}'_{p,j}\overline{T}'_{l,i}c_j c_i g_{\theta\theta})/$$

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-n_j m_i \overline{T}_{p,j}\overline{T}_{l,i}s_j c_i g_{ss} + n_j \overline{T}_{p,j}\overline{T}'_{l,i}s_j s_i g_{s\theta} + m_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i c_j g_{s\zeta} - \overline{T}'_{p,j}\overline{T}'_{l,i}c_j s_i g_{\theta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+n_j m_i \overline{T}_{p,j}\overline{T}_{l,i}c_j c_i g_{ss} - n_j \overline{T}_{p,j}\overline{T}'_{l,i}c_j s_i g_{s\theta} + m_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i s_j g_{s\zeta} - \overline{T}'_{p,j}\overline{T}'_{l,i}s_j s_i g_{\theta\zeta})/\sqrt{\,}$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(-m_j m_i \overline{T}_{p,j}\overline{T}_{l,i}s_j c_i g_{ss} + m_j \overline{T}_{p,j}\overline{T}'_{l,i}s_j s_i g_{s\theta} - m_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i c_j g_{s\theta} + \overline{T}'_{p,j}\overline{T}'_{l,i}c_j s_i g_{\theta\theta})/$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{B}\cdot\mathbf{B} = 2\int dv\,(+m_j m_i \overline{T}_{p,j}\overline{T}_{l,i}c_j c_i g_{ss} - m_j \overline{T}_{p,j}\overline{T}'_{l,i}c_j s_i g_{s\theta} - m_i \overline{T}_{l,i}\overline{T}'_{p,j}c_i s_j g_{s\theta} + \overline{T}'_{p,j}\overline{T}'_{l,i}s_j s_i g_{\theta\theta})/$$

**derivatives of helicity integrals**

- The first derivatives of $\int dv\,\mathbf{A}\cdot\mathbf{B}$ with respect to $A_{\theta,e,i,l}$, $A_{\theta,o,i,l}$, $A_{\zeta,e,i,l}$ and $A_{\zeta,o,i,l}$ are

$$\frac{\partial}{\partial A_{\theta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left(\frac{\partial\mathbf{A}}{\partial A_{\theta,e,i,l}}\cdot\mathbf{B}+\mathbf{A}\cdot\frac{\partial\mathbf{B}}{\partial A_{\theta,e,i,l}}\right) = \int dv\,(\overline{T}_{l,i}\cos\alpha_i\nabla\theta\cdot\mathbf{B}+\mathbf{A}\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\zeta)/\sqrt{\,} \quad (160)$$

$$\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left(\frac{\partial\mathbf{A}}{\partial A_{\theta,o,i,l}}\cdot\mathbf{B}+\mathbf{A}\cdot\frac{\partial\mathbf{B}}{\partial A_{\theta,o,i,l}}\right) = \int dv\,(\overline{T}_{l,i}\sin\alpha_i\nabla\theta\cdot\mathbf{B}+\mathbf{A}\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\zeta)/\sqrt{\,} \quad (161)$$

$$\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left(\frac{\partial\mathbf{A}}{\partial A_{\zeta,e,i,l}}\cdot\mathbf{B}+\mathbf{A}\cdot\frac{\partial\mathbf{B}}{\partial A_{\zeta,e,i,l}}\right) = \int dv\,(\overline{T}_{l,i}\cos\alpha_i\nabla\zeta\cdot\mathbf{B}-\mathbf{A}\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\theta)/\sqrt{\,} \quad (162)$$

$$\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left(\frac{\partial\mathbf{A}}{\partial A_{\zeta,o,i,l}}\cdot\mathbf{B}+\mathbf{A}\cdot\frac{\partial\mathbf{B}}{\partial A_{\zeta,o,i,l}}\right) = \int dv\,(\overline{T}_{l,i}\sin\alpha_i\nabla\zeta\cdot\mathbf{B}-\mathbf{A}\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\theta)/\sqrt{\,} \quad (163)$$

- Note that in the above expressions, $\mathbf{A}\cdot\mathbf{e}_s=0$ has been used.

- The second derivatives of $\int dv\,\mathbf{A}\cdot\mathbf{B}$ with respect to $A_{\theta,e,i,l}$, $A_{\theta,o,i,l}$, $A_{\zeta,e,i,l}$ and $A_{\zeta,o,i,l}$ are

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\cos\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\zeta+\overline{T}_{p,j}\cos\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\zeta\right]/\sqrt{\,} \quad (164)$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\cos\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\zeta+\overline{T}_{p,j}\sin\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\zeta\right]/\sqrt{\,} \quad (165)$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\cos\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\theta+\overline{T}_{p,j}\cos\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\zeta\right]/\sqrt{\,} \quad (166)$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\theta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\cos\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\theta+\overline{T}_{p,j}\sin\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\zeta\right]/\sqrt{\,} \quad (167)$$

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\sin\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\zeta+\overline{T}_{p,j}\cos\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\zeta\right]/\text{(168)}$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\sin\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\zeta+\overline{T}_{p,j}\sin\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\zeta\right]/\text{(169)}$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\sin\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\theta+\overline{T}_{p,j}\cos\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\zeta\right]/\text{(170)}$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\theta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\sin\alpha_i\nabla\theta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\theta+\overline{T}_{p,j}\sin\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\zeta\right]/\text{(171)}$$

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\cos\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\zeta-\overline{T}_{p,j}\cos\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\theta\right]/\text{(172)}$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\cos\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\zeta-\overline{T}_{p,j}\sin\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\theta\right]/\text{(173)}$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\cos\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\theta-\overline{T}_{p,j}\cos\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\theta\right]/\text{(174)}$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\zeta,e,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\cos\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\theta-\overline{T}_{p,j}\sin\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\cos\alpha_i\,\mathbf{e}_\theta\right]/\text{(175)}$$

$$\frac{\partial}{\partial A_{\theta,e,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\sin\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\zeta-\overline{T}_{p,j}\cos\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\theta\right]/\text{(176)}$$

$$\frac{\partial}{\partial A_{\theta,o,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[+\overline{T}_{l,i}\sin\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\zeta-\overline{T}_{p,j}\sin\alpha_j\nabla\theta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\theta\right]/\text{(177)}$$

$$\frac{\partial}{\partial A_{\zeta,e,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\sin\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\cos\alpha_j\,\mathbf{e}_\theta-\overline{T}_{p,j}\cos\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\theta\right]/\text{(178)}$$

$$\frac{\partial}{\partial A_{\zeta,o,j,p}}\frac{\partial}{\partial A_{\zeta,o,i,l}}\int dv\,\mathbf{A}\cdot\mathbf{B} = \int dv\left[-\overline{T}_{l,i}\sin\alpha_i\nabla\zeta\cdot\overline{T}'_{p,j}\sin\alpha_j\,\mathbf{e}_\theta-\overline{T}_{p,j}\sin\alpha_j\nabla\zeta\cdot\overline{T}'_{l,i}\sin\alpha_i\,\mathbf{e}_\theta\right]/\text{(179)}$$

- In these expressions the terms $\nabla\theta\cdot\mathbf{e}_\theta=\nabla\zeta\cdot\mathbf{e}_\zeta=1$, and $\nabla\theta\cdot\mathbf{e}_\zeta=\nabla\zeta\cdot\mathbf{e}_\theta=0$ have been included to show the structure of the derivation.

**derivatives of kinetic energy integrals**

- The first derivatives of $\int dv\,v^2$ with respect to $v_{s,e,i,l}$ etc. are

$$\frac{\partial}{\partial v_{s,e,i,l}}\int dv\,\mathbf{v}\cdot\mathbf{v} = 2\int dv\,\mathbf{v}\cdot\overline{T}_{l,i}\cos\alpha_i\nabla s \tag{180}$$

$$\frac{\partial}{\partial v_{s,o,i,l}}\int dv\,\mathbf{v}\cdot\mathbf{v} = 2\int dv\,\mathbf{v}\cdot\overline{T}_{l,i}\sin\alpha_i\nabla s \tag{181}$$

$$\frac{\partial}{\partial v_{\theta,e,i,l}}\int dv\,\mathbf{v}\cdot\mathbf{v} = 2\int dv\,\mathbf{v}\cdot\overline{T}_{l,i}\cos\alpha_i\nabla\theta \tag{182}$$

$$\frac{\partial}{\partial v_{\theta,o,i,l}}\int dv\,\mathbf{v}\cdot\mathbf{v} = 2\int dv\,\mathbf{v}\cdot\overline{T}_{l,i}\sin\alpha_i\nabla\theta \tag{183}$$

$$\frac{\partial}{\partial v_{\zeta,e,i,l}}\int dv\,\mathbf{v}\cdot\mathbf{v} = 2\int dv\,\mathbf{v}\cdot\overline{T}_{l,i}\cos\alpha_i\nabla\zeta \tag{184}$$

$$\frac{\partial}{\partial v_{\zeta,o,i,l}}\int dv\,\mathbf{v}\cdot\mathbf{v} = 2\int dv\,\mathbf{v}\cdot\overline{T}_{l,i}\sin\alpha_i\nabla\zeta \tag{185}$$

$$\tag{186}$$

**calculation of volume-integrated basis-function-weighted metric information**

- The required geometric information is calculated in ma00aa().

**Parameters**

| in | *lvol* | |
|----|--------|--|
| in | *mn* | |
| in | *lrad* | |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::cpus, allglobal::dbdx, allglobal↩
::ddttcc, allglobal::ddttcs, allglobal::ddttsc, allglobal::ddttss, allglobal::ddtzcc, allglobal::ddtzcs, allglobal↩
::ddtzsc, allglobal::ddtzss, allglobal::ddzzcc, allglobal::ddzzcs, allglobal::ddzzsc, allglobal::ddzzss, allglobal::dma,
allglobal::dmb, allglobal::dmd, allglobal::dmg, allglobal::dtoocc, allglobal::dtoocs, allglobal::dtoosc, allglobal↩
::dtooss, allglobal::ibnc, allglobal::ibns, allglobal::im, allglobal::in, allglobal::ivnc, allglobal::ivns, allglobal↩
::lcoordinatesingularity, allglobal::lma, allglobal::lmb, allglobal::lmc, allglobal::lmd, allglobal::lme, allglobal::lmf,
allglobal::lmg, allglobal::lmh, inputlist::mpol, allglobal::myid, allglobal::nadof, allglobal::ncpu, allglobal::notstellsym,
constants::one, fileunits::ounit, allglobal::rtm, allglobal::rtt, numerical::small, allglobal::tdstcc, allglobal::tdstcs,
allglobal::tdstsc, allglobal::tdstss, allglobal::tdszcc, allglobal::tdszcs, allglobal::tdszsc, allglobal::tdszss, allglobal::tt,
allglobal::ttsscc, allglobal::ttsscs, allglobal::ttsssc, allglobal::ttssss, constants::two, inputlist::wmacros, allglobal↩
::yesstellsym, and constants::zero.

Referenced by dfp100(), get_lu_beltrami_matrices(), preset(), and writereadgf().

Here is the caller graph for this function:
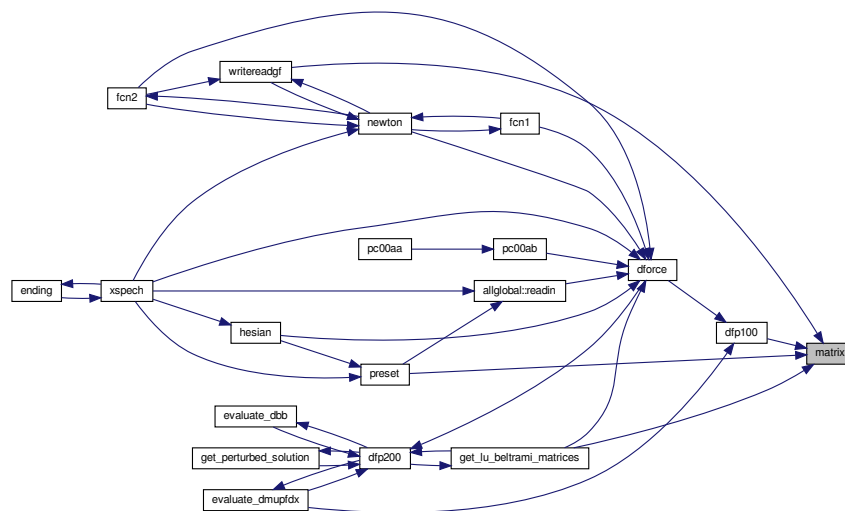
## 8.12 Metric quantities

**Functions/Subroutines**

- subroutine metrix (lquad, lvol)

  *Calculates the metric quantities, $\sqrt{g}\, g^{\mu\nu}$, which are required for the energy and helicity integrals.*

### 8.12.1 Detailed Description

### 8.12.2 Function/Subroutine Documentation

#### 8.12.2.1 metrix() 
```
subroutine metrix (
             integer, intent(in) lquad,
             integer, intent(in) lvol )
```

Calculates the metric quantities, $\sqrt{g}\, g^{\mu\nu}$, which are required for the energy and helicity integrals.

**metrics**

- The Jacobian, $\sqrt{g}$, and the "lower" metric elements, $g_{\mu\nu}$, are calculated by coords(), and are provided on a regular grid in "real-space", i.e. $(\theta, \zeta)$, at a given radial location, i.e. where $s$ is input.

**plasma region**

- In the plasma region, the required terms are $\bar{g}_{\mu\nu} \equiv g_{\mu\nu}/\sqrt{g}$.

$$
\begin{aligned}
\sqrt{g}\, g^{ss} &= \left( g_{\theta\theta}g_{\zeta\zeta} - g_{\theta\zeta}g_{\theta\zeta} \right) / \sqrt{g} \\
\sqrt{g}\, g^{s\theta} &= \left( g_{\theta\zeta}g_{s\zeta} - g_{s\theta}g_{\zeta\zeta} \right) / \sqrt{g} \\
\sqrt{g}\, g^{s\zeta} &= \left( g_{s\theta}g_{\theta\zeta} - g_{\theta\theta}g_{s\zeta} \right) / \sqrt{g} \\
\sqrt{g}\, g^{\theta\theta} &= \left( g_{\zeta\zeta}g_{ss} - g_{s\zeta}g_{s\zeta} \right) / \sqrt{g} \\
\sqrt{g}\, g^{\theta\zeta} &= \left( g_{s\zeta}g_{s\theta} - g_{\theta\zeta}g_{ss} \right) / \sqrt{g} \\
\sqrt{g}\, g^{\zeta\zeta} &= \left( g_{ss}g_{\theta\theta} - g_{s\theta}g_{s\theta} \right) / \sqrt{g}
\end{aligned}
\tag{187}
$$

**FFTs**

- After constructing the required quantities in real space, FFTs provided the required Fourier harmonics, which are returned through global.f90 . (The "extended" Fourier resolution is used.)

References allglobal::cfmn, allglobal::cpus, allglobal::dbdx, allglobal::efmn, allglobal::im, allglobal::ime, allglobal::in, allglobal::ine, allglobal::mne, allglobal::myid, allglobal::ncpu, allglobal::nt, allglobal::nz, allglobal::ofmn, constants←↩ ::one, fileunits::ounit, allglobal::sfmn, numerical::small, tfft(), and constants::zero.

Referenced by ma00aa().

Here is the call graph for this function:



Here is the caller graph for this function:
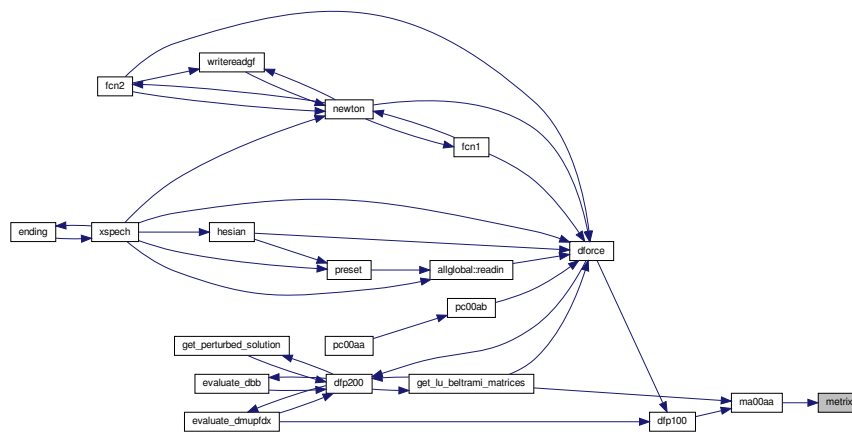
## 8.13 Solver for Beltrami (linear) system

**Functions/Subroutines**

- subroutine mp00ac (Ndof, Xdof, Fdof, Ddof, Ldfjac, iflag)

  *Solves Beltrami/vacuum (linear) system, given matrices.*
  ***unpacking fluxes, helicity multiplier***

### 8.13.1 Detailed Description

### 8.13.2 Function/Subroutine Documentation

**8.13.2.1 mp00ac()** subroutine mp00ac (
        integer, intent(in) *Ndof,*
        real, dimension(1:ndof), intent(in) *Xdof,*
        real, dimension(1:ndof) *Fdof,*
        real, dimension(1:ldfjac,1:ndof) *Ddof,*
        integer, intent(in) *Ldfjac,*
        integer *iflag* )

Solves Beltrami/vacuum (linear) system, given matrices.

**unpacking fluxes, helicity multiplier**

- The vector of "parameters", $\boldsymbol{\mu}$, is unpacked. (Recall that $\boldsymbol{\mu}$ was "packed" in ma02aa() .) In the following, $\boldsymbol{\psi} \equiv (\Delta \psi_t, \Delta \psi_p)^T$.

**construction of linear system**

- The equation $\nabla \times \mathbf{B} = \mu \mathbf{B}$ is cast as a matrix equation,

$$\mathcal{M} \cdot \mathbf{a} = \mathcal{R}, \tag{188}$$

  where $\mathbf{a}$ represents the degrees-of-freedom in the magnetic vector potential, $\mathbf{a} \equiv \{A_{\theta,e,i,l}, A_{\zeta,e,i,l}, \ldots\}$.

- The matrix $\mathcal{M}$ is constructed from $\mathcal{A} \equiv \mathrm{dMA}$ and $\mathcal{D} \equiv \mathrm{dMD}$, which were constructed in matrix() , according to

$$\mathcal{M} \equiv \mathcal{A} - \mu \mathcal{D}. \tag{189}$$

  Note that in the vacuum region, $\mu = 0$, so $\mathcal{M}$ reduces to $\mathcal{M} \equiv \mathcal{A}$.

- The construction of the vector $\mathcal{R}$ is as follows:

  - if Lcoordinatesingularity=T , then

$$\mathcal{R} \equiv -(\mathcal{B} - \mu \mathcal{E}) \cdot \boldsymbol{\psi} \tag{190}$$

  - if Lcoordinatesingularity=F and Lplasmaregion=T , then

$$\mathcal{R} \equiv -\mathcal{B} \cdot \boldsymbol{\psi} \tag{191}$$

– if Lcoordinatesingularity=F and Lvacuumregion=T , then

$$\mathcal{R} \equiv -\mathcal{G} - \mathcal{B} \cdot \boldsymbol{\psi} \tag{192}$$

The quantities $\mathcal{B} \equiv \mathrm{dMB}$, $\mathcal{E} \equiv \mathrm{dME}$ and $\mathcal{G} \equiv \mathrm{dMG}$ are constructed in matrix() .

**solving linear system**

It is *not* assumed that the linear system is positive definite. The LAPACK routine DSYSVX is used to solve the linear system.

**unpacking, ...**

- The magnetic degrees-of-freedom are unpacked by packab() .

- The error flag, ImagneticOK , is set that indicates if the Beltrami fields were successfully constructed.

**construction of "constraint" function**

- The construction of the function $\mathbf{f}(\boldsymbol{\mu})$ is required so that iterative methods can be used to construct the Beltrami field consistent with the required constraints (e.g. on the enclosed fluxes, helicity, rotational-transform, ...).

  **See also**

  ma02aa() for additional details.

  **plasma region**

  – For Lcoordinatesingularity=T , the returned function is:

  $$\mathbf{f}(\mu, \Delta\psi_p) \equiv \begin{cases} ( & 0 & , & 0 & )^T, & \text{if Lconstraint} & = & -1 \\ ( & 0 & , & 0 & )^T, & \text{if Lconstraint} & = & 0 \\ ( & \iota(+1) - \text{iota (lvol )} & , & 0 & )^T, & \text{if Lconstraint} & = & 1 \\ ( & ? & , & ? & )^T, & \text{if Lconstraint} & = & 2 \end{cases} \tag{193}$$

  – For Lcoordinatesingularity=F , the returned function is:

  $$\mathbf{f}(\mu, \Delta\psi_p) \equiv \begin{cases} ( & 0 & , & 0 & )^T, & \text{if Lconstraint} & = & -1 \\ ( & 0 & , & 0 & )^T, & \text{if Lconstraint} & = & 0 \\ ( & \iota(-1) - \text{oita(lvol-1)} & , & \iota(+1) - \text{iota(lvol)} & )^T, & \text{if Lconstraint} & = & 1 \\ ( & ? & , & ? & )^T, & \text{if Lconstraint} & = & 2 \end{cases} \tag{1}$$

  **vacuum region**

  – For the vacuum region, the returned function is:

  $$\mathbf{f}(\Delta\psi_t, \Delta\psi_p) \equiv \begin{cases} ( & 0 & , & 0 & )^T, & \text{if Lconstraint} & = & -1 \\ ( & I - \text{curtor} & , & G - \text{curpol} & )^T, & \text{if Lconstraint} & = & 0 \\ ( & \iota(-1) - \text{oita(lvol-1)} & , & G - \text{curpol} & )^T, & \text{if Lconstraint} & = & 1 \\ ( & ? & , & ? & )^T, & \text{if Lconstraint} & = & 2 \end{cases} \tag{195}$$

- The rotational-transform, $\iota$, is computed by tr00ab() ; and the enclosed currents, $I$ and $G$, are computed by curent() .

**early termination**

- If $|\mathbf{f}| < $ mupftol , then early termination is enforced (i.e., iflag is set to a negative integer). (See ma02aa() for details of how mp00ac() is called iteratively.)

**Parameters**

| | | |
|---|---|---|
| in | *Ndof* | |
| in | *Xdof* | |
| | *Fdof* | |
| | *Ddof* | |
| in | *Ldfjac* | |
| | *iflag* | indicates whether (i) iflag=1: "function" values are required; or (ii) iflag=2: "derivative" values are required |

References allglobal::cpus, curent(), inputlist::curpol, inputlist::curtor, constants::half, inputlist::helicity, allglobal↩
::im, allglobal::in, intghs(), inputlist::iota, allglobal::ivol, allglobal::lcoordinatesingularity, allglobal::lplasmaregion,
inputlist::lrad, allglobal::lvacuumregion, numerical::machprec, allglobal::mns, mtrxhs(), inputlist::mu, allglobal↩
::myid, allglobal::ncpu, allglobal::notstellsym, allglobal::nt, inputlist::ntor, allglobal::nz, inputlist::oita, constants↩
::one, fileunits::ounit, packab(), rungmres(), numerical::small, tr00ab(), inputlist::wmacros, allglobal::yesstellsym,
and constants::zero.

Referenced by ma02aa().

Here is the call graph for this function:



Here is the caller graph for this function:

## 8.14  Force-driver

**Functions/Subroutines**

- subroutine newton (NGdof, position, ihybrd)

  *Employs Newton method to find* $\mathbf{F}(\mathbf{x}) = 0$*, where* $\mathbf{x} \equiv \{\mathrm{geometry}\}$ *and* $\mathbf{F}$ *is defined in dforce() .*
- subroutine writereadgf (readorwrite, NGdof, ireadhessian)

  *read or write force-derivative matrix*
- subroutine fcn1 (NGdof, xx, fvec, irevcm)

  *fcn1*
- subroutine fcn2 (NGdof, xx, fvec, fjac, Ldfjac, irevcm)

  *fcn2*

### 8.14.1  Detailed Description

### 8.14.2  Function/Subroutine Documentation

#### 8.14.2.1  newton()  `subroutine newton (`
`        integer, intent(in) NGdof,`
`        real, dimension(0:ngdof), intent(inout) position,`
`        integer, intent(out) ihybrd )`

Employs Newton method to find $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{x} \equiv \{\mathrm{geometry}\}$ and $\mathbf{F}$ is defined in dforce() .

Solves $\mathbf{F}(\xi) = 0$, where $\mathbf{F} \equiv \{[[p + B^2/2]]_{i,l}, I_{i,l}\}$ and $\xi \equiv \{R_{i,l}, Z_{i,l}\}$.

**iterative, reverse communication loop**

- The iterative, Newton search to find $\mathbf{x} \equiv \{\mathrm{geometry}\} \equiv \{R_{i,l}, Z_{i,l}\}$ such that $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{F}$ and its derivatives, $\nabla_{\mathbf{x}}\mathbf{F}$, are calculated by dforce() , is provided by either

    - `C05NDF` if `Lfindzero=1` , which only uses function values; or
    - `C05PDF` if `Lfindzero=2`, which uses user-provided derivatives.

- The iterative search will terminate when the solution is within `c05xtol` of the true solution (see NAG documentation).

- The input variable `c05factor` is provided to determine the initial step bound (see NAG documentation).

**logic, writing/reading from file**

- Before proceeding with iterative search, dforce() is called to determine the magnitude of the initial force imbalance, and if this is less than `forcetol` then the iterative search will not be performed.

- As the iterations proceed, wrtend() will be called to save itermediate information (also see xspech() ).

- If the derivative matrix, $\nabla_{\mathbf{x}}\mathbf{F}$, is required, i.e. if `Lfindzero=2` , and if `LreadGF=T` then the derivative matrix will initially be read from .ext.sp.DF , if it exists, or from .sp.DF .

- As the iterations proceed, the derivative matrix will be written to .ext.sp.DF .

**Parameters**

| | | |
|---|---|---|
| `in` | *NGdof* | |
| `in,out` | *position* | |
| `out` | *ihybrd* | |

References allglobal::bbe, allglobal::bbo, allglobal::cpus, allglobal::dbbdmp, allglobal::dessian, allglobal↵
::dffdrz, dforce(), allglobal::dmupfdx, allglobal::energy, inputlist::ext, fcn1(), fcn2(), allglobal::forceerr, allglobal↵
::hessian, inputlist::igeometry, allglobal::iie, allglobal::iio, allglobal::im, allglobal::in, allglobal::irbc, allglobal↵
::irbs, allglobal::izbc, allglobal::izbs, newtontime::lastcpu, allglobal::lgdof, allglobal::lhessianallocated, allglobal↵
::localconstraint, allglobal::mvol, allglobal::myid, allglobal::ncpu, newtontime::ndcalls, newtontime::nfcalls,
allglobal::nfreeboundaryiterations, allglobal::notstellsym, constants::one, fileunits::ounit, numerical::sqrtmachprec,
constants::ten, constants::two, inputlist::wmacros, writereadgf(), and constants::zero.

Referenced by fcn1(), fcn2(), writereadgf(), and xspech().

Here is the call graph for this function:
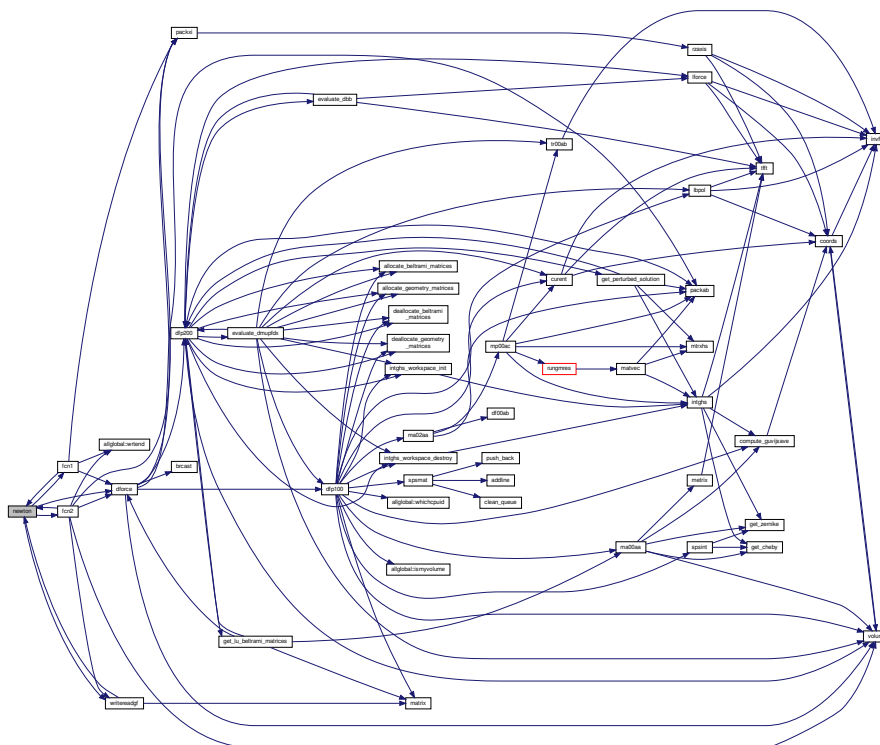
Here is the caller graph for this function:
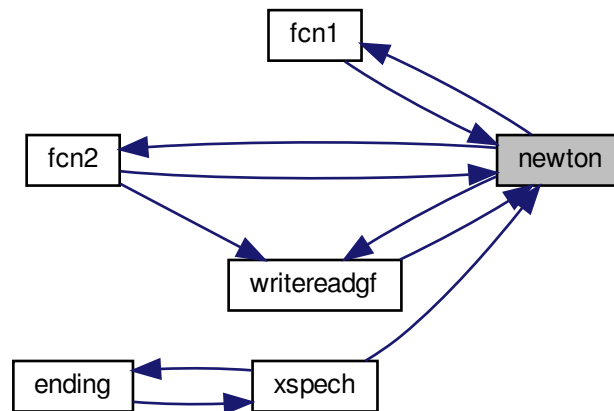


**8.14.2.2 writereadgf()** subroutine writereadgf (
        character, intent(in) *readorwrite,*
        integer, intent(in) *NGdof,*
        integer, intent(out) *ireadhessian* )

read or write force-derivative matrix

**Parameters**

| | | |
|------|--------------|--|
| in   | *readorwrite* | |
| in   | *NGdof*       | |
| out  | *ireadhessian* | |

References allglobal::cpus, fileunits::dunit, inputlist::ext, allglobal::hessian, inputlist::igeometry, allglobal::im, allglobal::in, inputlist::istellsym, inputlist::lfreebound, allglobal::lhessianallocated, matrix(), inputlist::mpol, allglobal⤶ ::myid, newton(), inputlist::ntor, inputlist::nvol, fileunits::ounit, and constants::zero.

Referenced by fcn2(), and newton().

Here is the call graph for this function:



Here is the caller graph for this function:
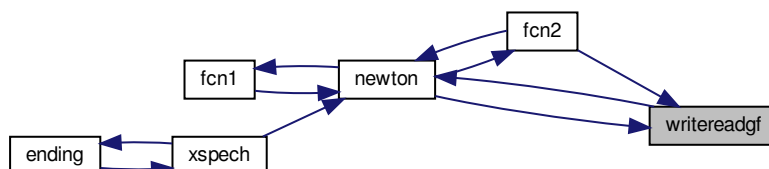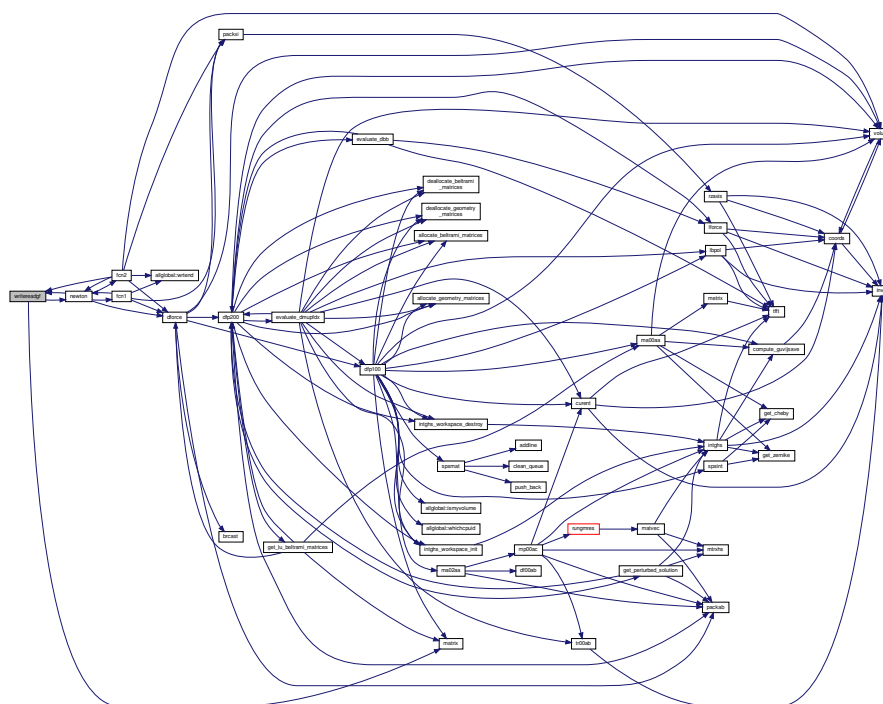


**8.14.2.3 fcn1()** `subroutine fcn1 (`
`        integer, intent(in) NGdof,`
`        real, dimension(1:ngdof), intent(in) xx,`
`        real, dimension(1:ngdof), intent(out) fvec,`
`        integer, intent(in) irevcm )`

fcn1

**Parameters**

| in  | NGdof  |  |
| --- | ------ | -- |
| in  | xx     |  |
| out | fvec   |  |
| in  | irevcm |  |

References allglobal::bbe, allglobal::bbo, allglobal::cpus, allglobal::dbbdmp, allglobal::dessian, allglobal::dffdrz, dforce(), allglobal::dmupfdx, allglobal::energy, inputlist::ext, allglobal::forceerr, allglobal::hessian, inputlist↩
::igeometry, allglobal::iie, allglobal::iio, allglobal::im, allglobal::in, allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, newtontime::lastcpu, allglobal::lgdof, allglobal::lhessianallocated, allglobal::mvol, allglobal::myid, allglobal::ncpu, newtontime::ndcalls, newton(), newtontime::nfcalls, allglobal::nfreeboundaryiterations, allglobal↩
::notstellsym, constants::one, fileunits::ounit, packxi(), numerical::sqrtmachprec, constants::ten, constants::two, inputlist::wmacros, allglobal::wrtend(), and constants::zero.

Referenced by newton().

Here is the call graph for this function:



Here is the caller graph for this function:

**8.14.2.4 fcn2()** `subroutine fcn2 (`
```
    integer, intent(in) NGdof,
    real, dimension(1:ngdof), intent(in) xx,
    real, dimension(1:ngdof), intent(out) fvec,
    real, dimension(1:ldfjac,1:ngdof), intent(out) fjac,
    integer, intent(in) Ldfjac,
    integer, intent(in) irevcm )
```

fcn2

**Parameters**

| | | |
|---|---|---|
| in | *NGdof* | |
| in | *xx* | |
| out | *fvec* | |
| out | *fjac* | |
| in | *Ldfjac* | |
| in | *irevcm* | |

References allglobal::bbe, allglobal::bbo, allglobal::cpus, allglobal::dbbdmp, allglobal::dessian, allglobal::dffdrz, dforce(), allglobal::dmupfdx, allglobal::energy, inputlist::ext, allglobal::forceerr, allglobal::hessian, inputlist↩
::igeometry, allglobal::iie, allglobal::iio, allglobal::im, allglobal::in, allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, newtontime::lastcpu, allglobal::lgdof, allglobal::lhessianallocated, allglobal::mvol, allglobal::myid, allglobal::ncpu, newtontime::ndcalls, newton(), newtontime::nfcalls, allglobal::nfreeboundaryiterations, allglobal↩
::notstellsym, constants::one, fileunits::ounit, packxi(), numerical::sqrtmachprec, constants::ten, constants::two, volume(), inputlist::wmacros, writereadgf(), allglobal::wrtend(), and constants::zero.

Referenced by newton().

Here is the call graph for this function:
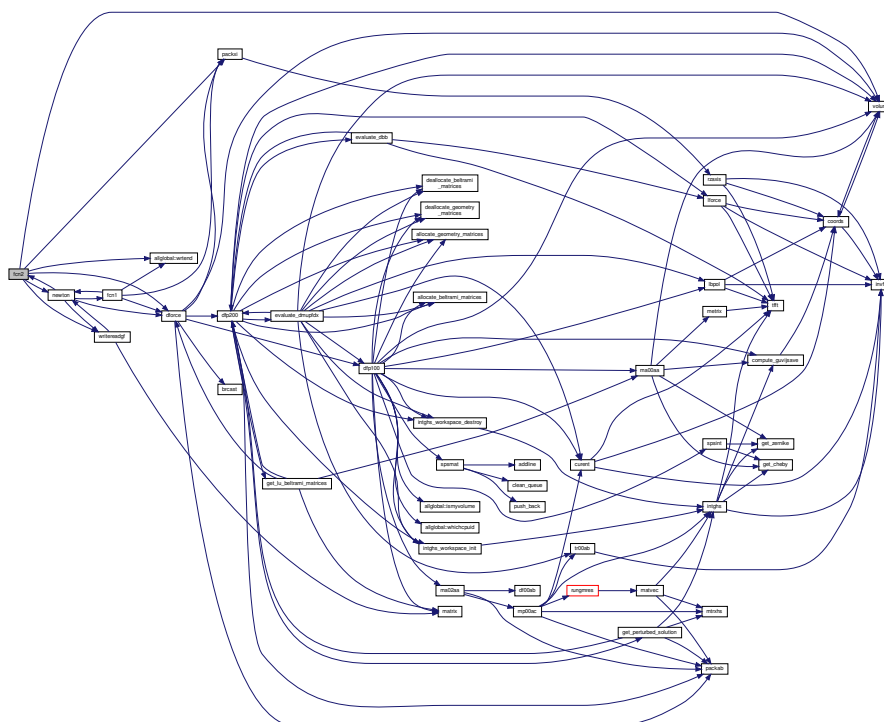
Here is the caller graph for this function:
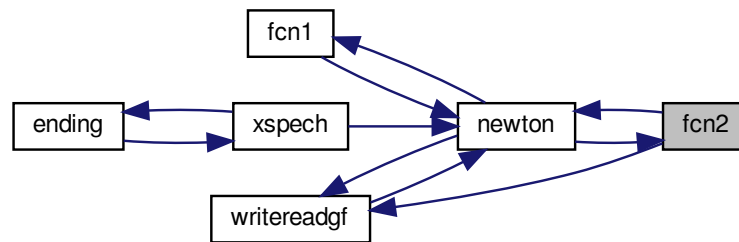
## 8.15 Some miscellaneous numerical routines

**Functions/Subroutines**

- subroutine gi00ab (Mpol, Ntor, Nfp, mn, im, in)

  *Assign Fourier mode labels.*
- subroutine tfft (Nt, Nz, ijreal, ijimag, mn, im, in, efmn, ofmn, cfmn, sfmn, ifail)

  *Forward Fourier transform (fftw wrapper)*
- subroutine invfft (mn, im, in, efmn, ofmn, cfmn, sfmn, Nt, Nz, ijreal, ijimag)

  *Inverse Fourier transform (fftw wrapper)*
- subroutine gauleg (n, weight, abscis, ifail)

  *Gauss-Legendre weights and abscissae.*

### 8.15.1 Detailed Description

### 8.15.2 Function/Subroutine Documentation

#### 8.15.2.1 gi00ab()  `subroutine gi00ab (`

```
        integer, intent(in) Mpol,
        integer, intent(in) Ntor,
        integer, intent(in) Nfp,
        integer, intent(in) mn,
        integer, dimension(mn), intent(out) im,
        integer, dimension(mn), intent(out) in )
```

Assign Fourier mode labels.

- This routine assigns the Fourier mode labels that converts a double-sum into a single sum; i.e., the $m_j$ and $n_j$ are assigned where

$$
\begin{aligned}
f(\theta, \zeta) & = \sum_{n=0}^{N} f_{0,n} \cos(-n \, N_P \, \zeta) + \sum_{m=1}^{M} \sum_{n=-N}^{N} f_{m,n} \cos(m\theta - n \, N_P \, \zeta) & (196) \\
& = \sum_{j} f_j \cos(m_j \theta - n_j \zeta), & (197)
\end{aligned}
$$

where $N \equiv \mathtt{Ntor}$ and $M \equiv \mathtt{Mpol}$ are given on input, and $N_P \equiv \mathtt{Nfp}$ is the field periodicity.

Referenced by allglobal::readin().

Here is the caller graph for this function:
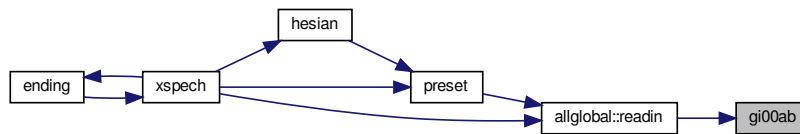
**8.15.2.2 tfft()** ```subroutine tfft (```
```
        integer Nt,
        integer Nz,
        real, dimension(1:nt*nz) ijreal,
        real, dimension(1:nt*nz) ijimag,
        integer mn,
        integer, dimension(1:mn) im,
        integer, dimension(1:mn) in,
        real, dimension(1:mn) efmn,
        real, dimension(1:mn) ofmn,
        real, dimension(1:mn) cfmn,
        real, dimension(1:mn) sfmn,
        integer ifail )
```

Forward Fourier transform (fftw wrapper)

- This constructs the "forward" Fourier transform.

- Given a set of data, $(f_i, g_i)$ for $i = 1, \ldots N_\theta N_\zeta$, on a regular two-dimensional angle grid, where $\theta_j = 2\pi j/N_\theta$ for $j = 0, N_\theta - 1$, and $\zeta_k = 2\pi k/N_\zeta$ for $k = 0, N_\zeta - 1$. The "packing" is governed by $i = 1 + j + kN_\theta$. The "discrete" resolution is $N_\theta \equiv \mathtt{Nt}$, $N_\zeta \equiv \mathtt{Nz}$ and $\mathtt{Ntz} = \mathtt{Nt} \times \mathtt{Nz}$ , which are set in preset() .

- The Fourier harmonics consistent with Eqn. (197) are constructed. The mode identification labels appearing in Eqn. (197) are $m_j \equiv \mathtt{im(j)}$ and $n_j \equiv \mathtt{in(j)}$ , which are set in readin() via a call to gi00ab() .

**Parameters**

| Nt | |
|---|---|
| Nz | |
| ijreal | |
| ijimag | |
| mn | |
| im | |
| in | |
| efmn | |
| ofmn | |
| cfmn | |
| sfmn | |
| ifail | |

References fftw_interface::cplxin, fftw_interface::cplxout, constants::half, inputlist::nfp, fileunits::ounit, constants↩
::pi2, allglobal::pi2nfp, fftw_interface::planf, and constants::zero.

Referenced by bnorml(), curent(), evaluate_dbb(), intghs(), lbpol(), lforce(), metrix(), preset(), rzaxis(), and wa00aa().

Here is the caller graph for this function:
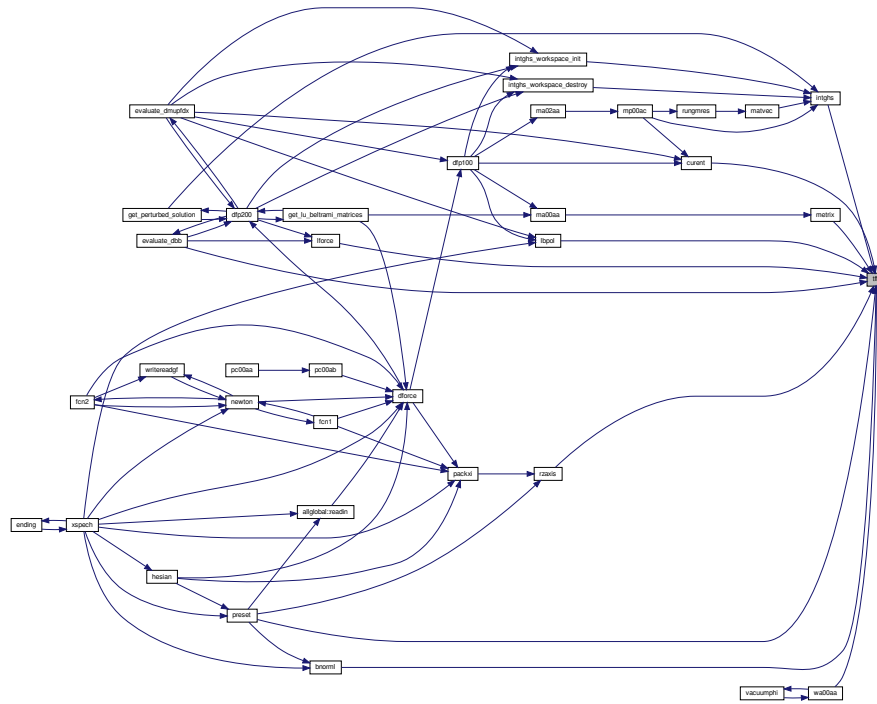


**8.15.2.3 invfft()** subroutine invfft (

```
integer, intent(in) mn,
integer, dimension(mn), intent(in) im,
integer, dimension(mn), intent(in) in,
real, dimension(mn), intent(in) efmn,
real, dimension(mn), intent(in) ofmn,
real, dimension(mn), intent(in) cfmn,
real, dimension(mn), intent(in) sfmn,
integer, intent(in) Nt,
integer, intent(in) Nz,
real, dimension(nt*nz), intent(out) ijreal,
real, dimension(nt*nz), intent(out) ijimag )
```

Inverse Fourier transform (fftw wrapper)

- Given the Fourier harmonics, the data on a regular angular grid are constructed.

- This is the inverse routine to tfft() .

**Parameters**

| in | *mn*   |  |
|----|--------|--|
| in | *im*   |  |
| in | *in*   |  |
| in | *efmn* |  |

**Parameters**

| in | *ofmn* | |
|----|--------|---|
| in | *cfmn* | |
| in | *sfmn* | |
| in | *Nt* | |
| in | *Nz* | |
| out | *ijreal* | |
| out | *ijimag* | |

References fftw_interface::cplxin, fftw_interface::cplxout, constants::half, inputlist::nfp, fftw_interface::planb, constants::two, and constants::zero.

Referenced by coords(), curent(), intghs(), jo00aa(), lbpol(), lforce(), preset(), rzaxis(), and tr00ab().

Here is the caller graph for this function:
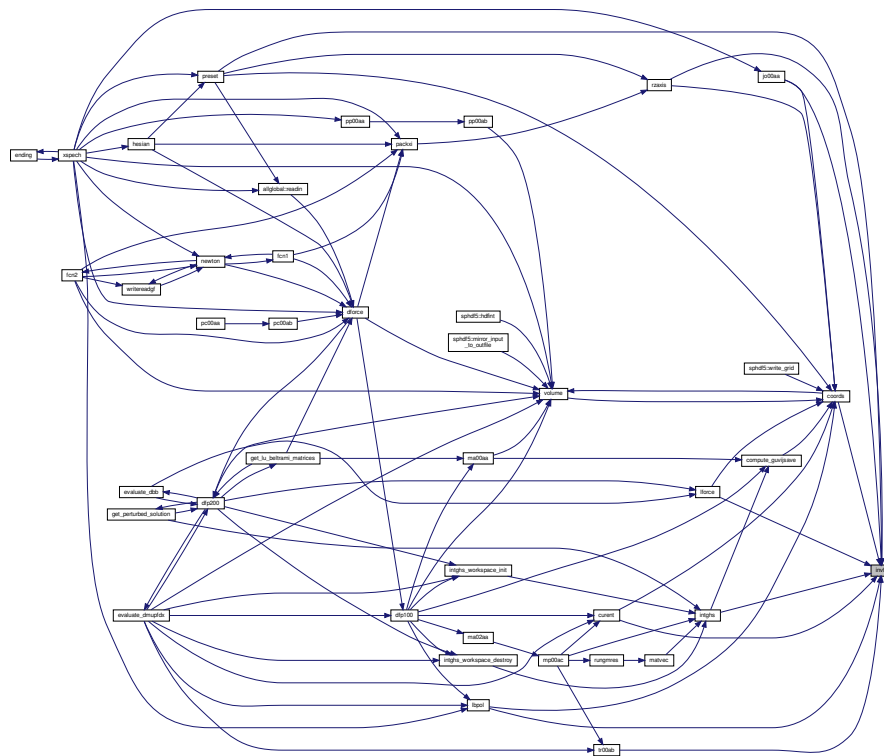


**8.15.2.4  gauleg()**  `subroutine gauleg (`
            `integer, intent(in) n,`
            `real, dimension(n), intent(out) weight,`
            `real, dimension(n), intent(out) abscis,`
            `integer, intent(out) ifail )`

Gauss-Legendre weights and abscissae.

- Compute Gaussian integration weights and abscissae.

- From Numerical Recipes.

---

**Parameters**

| | | |
|---|---|---|
| in | *n* | |
| out | *weight* | |
| out | *abscis* | |
| out | *ifail* | |

References constants::one, constants::pi, constants::two, and constants::zero.

Referenced by preset().

Here is the caller graph for this function:
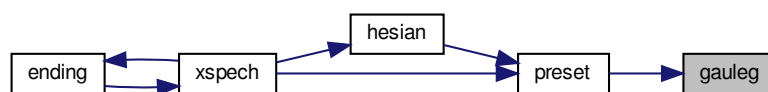
## 8.16 "packing" of Beltrami field solution vector

**Functions/Subroutines**

- subroutine packab (packorunpack, lvol, NN, solution, ideriv)

  *Packs and unpacks Beltrami field solution vector.*
- subroutine packxi (NGdof, position, Mvol, mn, iRbc, iZbs, iRbs, iZbc, packorunpack, LComputeDerivatives, LComputeAxis)

  *Packs, and unpacks, geometrical degrees of freedom; and sets coordinate axis.*

### 8.16.1 Detailed Description

### 8.16.2 Function/Subroutine Documentation

#### 8.16.2.1 packab()

```
subroutine packab (
        character, intent(in) packorunpack,
        integer, intent(in) lvol,
        integer, intent(in) NN,
        real, dimension(1:nn) solution,
        integer, intent(in) ideriv )
```

Packs and unpacks Beltrami field solution vector.

**construction of "vector" of independent degrees of freedom**

- Numerical routines for solving linear equations typically require the unknown, independent degrees of freedom to be "packed" into a vector, $\mathbf{x}$.

- The magnetic field is defined by the independent degrees of freedom in the Chebyshev-Fourier representation of the vector potential, $A_{\theta,e,i,l}$ and $A_{\zeta,e,i,l}$; and the non-stellarator-symmetric terms if relevant, $A_{\theta,o,i,l}$ and $A_{\zeta,o,i,l}$; and the Lagrange multipliers, $a_i, b_i, c_i, d_i, e_i$, etc. as required to enforce the constraints:

$$\mathbf{x} \equiv \{A_{\theta,e,i,l}, A_{\zeta,e,i,l}, A_{\theta,o,i,l}, A_{\zeta,o,i,l}, a_i, b_i, c_i, d_i, e_i, f_i, g_1, h_1\}. \tag{198}$$

- The "packing" index is assigned in preset() .

**Parameters**

| | |
|---|---|
| *packorunpack* | |
| *lvol* | |
| *NN* | |
| *solution* | |
| *ideriv* | |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::cpus, allglobal::im, allglobal::in, allglobal::lma, allglobal::lmavalue, allglobal::lmb, allglobal::lmbvalue, allglobal::lmc, allglobal::lmcvalue, allglobal←↩ ::lmd, allglobal::lmdvalue, allglobal::lme, allglobal::lmevalue, allglobal::lmf, allglobal::lmfvalue, allglobal::lmg,

allglobal::lmgvalue, allglobal::lmh, allglobal::lmhvalue, inputlist::lrad, allglobal::myid, allglobal::ncpu, allglobal←↩
::notstellsym, fileunits::ounit, numerical::small, allglobal::tt, allglobal::yesstellsym, and constants::zero.

Referenced by dforce(), dfp200(), get_perturbed_solution(), ma02aa(), matvec(), and mp00ac().

Here is the caller graph for this function:
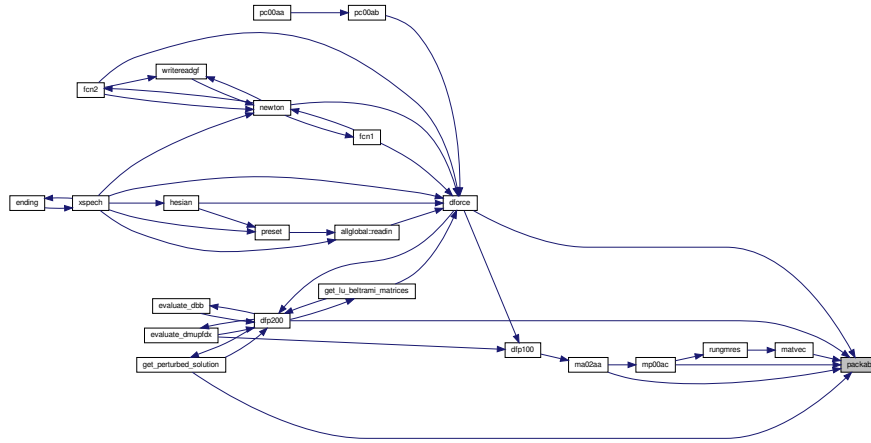


**8.16.2.2 packxi()**  subroutine packxi (
            integer, intent(in) *NGdof,*
            real, dimension(0:ngdof) *position,*
            integer, intent(in) *Mvol,*
            integer, intent(in) *mn,*
            real, dimension(1:mn,0:mvol) *iRbc,*
            real, dimension(1:mn,0:mvol) *iZbs,*
            real, dimension(1:mn,0:mvol) *iRbs,*
            real, dimension(1:mn,0:mvol) *iZbc,*
            character *packorunpack,*
            logical, intent(in) *LComputeDerivatives,*
            logical, intent(in) *LComputeAxis* )

Packs, and unpacks, geometrical degrees of freedom; and sets coordinate axis.

**geometrical degrees of freedom**

- The geometrical degrees-of-freedom, namely the $R_{j,v}$ and $Z_{j,v}$ where $v$ labels the interface and $j$ labels the Fourier harmonic, must be "packxi", and "unpackxi", into a single vector, $\boldsymbol{\xi}$, so that standard numerical routines can be called to find solutions to force-balance, i.e. $\mathbf{F}[\boldsymbol{\xi}] = 0$.

- A coordinate "pre-conditioning" factor is included:

$$\boldsymbol{\xi}_k \equiv \frac{R_{j,v}}{\Psi_{j,v}}, \tag{199}$$

where $\Psi_{j,v} \equiv \texttt{psifactor(j,v)}$, which is defined in global.f90 .

**coordinate axis**

- The coordinate axis is not an independent degree-of-freedom of the geometry. It is constructed by extrapolating the geometry of the innermost interface down to a line.

- Note that if the coordinate axis depends only on the geometry of the innermost interface then the block tridiagonal structure of the the force-derivative matrix is preserved.

- Define the arc-length weighted averages,

$$R_0(\zeta) \equiv \frac{\int_0^{2\pi} R_1(\theta, \zeta)dl}{L(\zeta)}, \qquad Z_0(\zeta) \equiv \frac{\int_0^{2\pi} Z_1(\theta, \zeta)dl}{L(\zeta)}, \tag{200}$$

  where $L(\zeta) \equiv \int_0^{2\pi} dl$ and $dl \equiv \sqrt{\partial_\theta R_1(\theta, \zeta)^2 + \partial_\theta Z_1(\theta, \zeta)^2} \, d\theta$.

- Note that if $dl$ does not depend on $\theta$, i.e. if $\theta$ is the equal arc-length angle, then the expressions simplify.

- Note that the geometry of the coordinate axis thus constructed only depends on the geometry of the innermost interface, by which I mean that the geometry of the coordinate axis is independent of the angle parameterization.

**some numerical comments**

- First, the differential poloidal length, $dl \equiv \sqrt{R_\theta^2 + Z_\theta^2}$, is computed in real space using an inverse FFT from the Fourier harmonics of $R$ and $Z$.

- Second, the Fourier harmonics of the $dl$ are computed using an FFT. The integration over $\theta$ to construct $L \equiv \int dl$ is now trivial: just multiply the $m = 0$ harmonics of $dl$ by $2\pi$. The `ajk(1:mn)` variable is used.

- Next, the weighted $R \, dl$ and $Z \, dl$ are computed in real space, and the poloidal integral is similarly taken.

- Lastly, the Fourier harmonics are constructed using an FFT after dividing in real space.

**Parameters**

| in | NGdof | |
|----|-------|--|
| | position | |
| in | Mvol | |
| in | mn | |
| | iRbc | |
| | iZbs | |
| | iRbs | |
| | iZbc | |
| | packorunpack | |
| in | LComputeDerivatives | |
| in | LComputeAxis | |

References allglobal::ajk, allglobal::cfmn, allglobal::comn, allglobal::cpus, allglobal::efmn, allglobal::evmn, inputlist::igeometry, allglobal::ijimag, allglobal::ijreal, allglobal::im, allglobal::in, allglobal::irij, allglobal::izij, allglobal←:jiimag, allglobal::jireal, inputlist::lfindzero, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, allglobal::nt, inputlist::ntor, inputlist::nvol, allglobal::nz, allglobal::odmn, allglobal::ofmn, fileunits::ounit, allglobal::psifactor, allglobal::rscale, rzaxis(), allglobal::sfmn, allglobal::simn, allglobal::trij, allglobal::tzij, allglobal::yesstellsym, and constants::zero.

Referenced by dforce(), fcn1(), fcn2(), hesian(), and xspech().

Here is the call graph for this function:
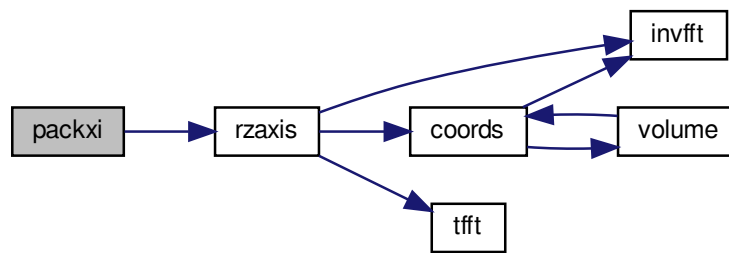


Here is the caller graph for this function:
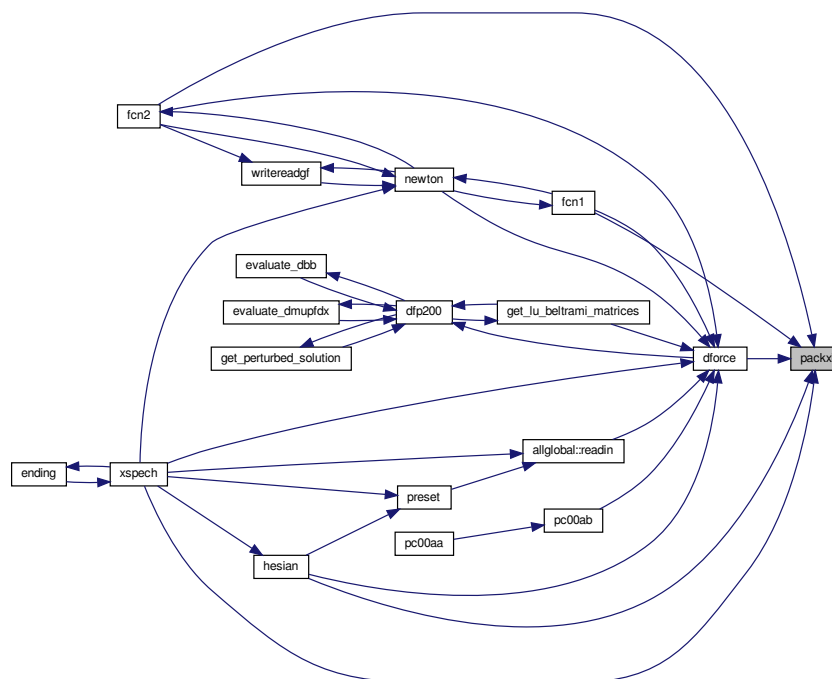
## 8.17 Conjugate-Gradient method

**Functions/Subroutines**

- subroutine pc00aa (NGdof, position, Nvol, mn, ie04dgf)

  *Use preconditioned conjugate gradient method to find minimum of energy functional.*
- subroutine pc00ab (mode, NGdof, Position, Energy, Gradient, nstate, iuser, ruser)

  *Returns the energy functional and it's derivatives with respect to geometry.*

### 8.17.1 Detailed Description

### 8.17.2 Function/Subroutine Documentation

**8.17.2.1 pc00aa()** `subroutine pc00aa (`
```
        integer, intent(in) NGdof,
        real, dimension(0:ngdof), intent(inout) position,
        integer, intent(in) Nvol,
        integer, intent(in) mn,
        integer ie04dgf )
```

Use preconditioned conjugate gradient method to find minimum of energy functional.

**energy functional**

The energy functional is described in pc00ab() .

**relevant input variables**

- The following input variables control the operation of `E04DGF` :

  - `epsilon` : weighting of "spectral energy"; see pc00ab()
  - `maxstep` : this is given to `E04DGF` for the `Maximum Step Length`
  - `maxiter` : upper limit on derivative calculations used in the conjugate gradient iterations
  - `verify` : if `verify=1`, then `E04DGF` will confirm user supplied gradients (provided by pc00ab() ) are correct;

- **Todo** Unfortunately, `E04DGF` seems to require approximately $3N$ function evaluations before proceeding to minimize the energy functional, where there are $N$ degrees of freedom. I don't know how to turn this off!

**Parameters**

| | | |
|---|---|---|
| in | *NGdof* | |
| in,out | *position* | |
| in | *Nvol* | |
| in | *mn* | |
| | *ie04dgf* | |

References allglobal::cpus, allglobal::energy, allglobal::forceerr, inputlist::forcetol, allglobal::myid, allglobal::ncpu, fileunits::ounit, pc00ab(), constants::ten, and constants::zero.

Here is the call graph for this function:



### 8.17.2.2 pc00ab()
```
subroutine pc00ab (
          integer mode,
          integer NGdof,
          real, dimension(1:ngdof) Position,
          real Energy,
          real, dimension(1:ngdof) Gradient,
          integer nstate,
          integer, dimension(1:2) iuser,
          real, dimension(1:1) ruser )
```

Returns the energy functional and it's derivatives with respect to geometry.

**Energy functional**

- The energy functional is

$$F \equiv \sum_{l=1}^{N} \int_{\mathcal{V}} \left( \frac{p}{\gamma - 1} + \frac{B^2}{2} \right) dv, \tag{201}$$

where $N \equiv \mathtt{Nvol}$ is the number of interfaces.

- Assuming that the toroidal and poloidal fluxes, $\psi_t$ and $\psi_p$, the helicity, $\mathcal{K}$, the helicity multiplier, $\mu$, and/or the interface rotational-transforms, $\iota$, are appropriately constrained, the Beltrami fields in each volume depend only the geometry of the adjacent interfaces. So, the energy functional is assumed to be a function of "position", i.e. $F = F(R_{l,j}, Z_{l,j})$.

- Introducing a ficitious time, $t$, the position may be advanced according to

$$
\begin{aligned}
\frac{\partial R_j}{\partial t} &\equiv -\frac{\partial}{\partial R_j} \sum_{l=1}^{N} \int \left( \frac{p}{\gamma-1} + \frac{B^2}{2} \right) dv, \\
\frac{\partial Z_j}{\partial t} &\equiv -\frac{\partial}{\partial Z_j} \sum_{l=1}^{N} \int \left( \frac{p}{\gamma-1} + \frac{B^2}{2} \right) dv.
\end{aligned}
\tag{202}
$$

- There remain degrees of freedom in the angle representation of the interfaces.

**Spectral energy minimization**

- Consider variations which do not affect the geometry of the surfaces,

$$
\delta R = R_\theta\, u, \tag{203}
$$
$$
\delta Z = Z_\theta\, u, \tag{204}
$$

where $u$ is a angle variation.

- The corresponding variation in each of the Fourier harmonics is

$$
\delta R_j \equiv \oint\!\!\!\oint d\theta d\zeta\ R_\theta\, u\, \cos\alpha_j, \tag{205}
$$

$$
\delta Z_j \equiv \oint\!\!\!\oint d\theta d\zeta\ Z_\theta\, u\, \sin\alpha_j, \tag{206}
$$

- Following Hirshman et al., introducing the normalized spectral width

$$
M \equiv \frac{\sum_j (m_j^p + n_j^q)(R_{l,j}^2 + Z_{l,j}^2)}{\sum_j (R_{l,j}^2 + Z_{l,j}^2)}, \tag{207}
$$

- Using the notation

$$
N \equiv \sum_j \lambda_j (R_{l,j}^2 + Z_{l,j}^2), \tag{208}
$$

$$
D \equiv \sum_j (R_{l,j}^2 + Z_{l,j}^2), \tag{209}
$$

where $\lambda_j \equiv m_j^p + n_j^q$, the variation in the normalized spectral width is

$$
\delta M = (\delta N - M\delta D)/D. \tag{210}
$$

- For tangential variations,

$$
\delta N = 2 \oint\!\!\!\oint d\theta d\zeta\ u \left( R_\theta \sum_j \lambda_j R_j \cos\alpha_j + Z_\theta \sum_j \lambda_j Z_j \sin\alpha_j \right), \tag{211}
$$

$$
\delta D = 2 \oint\!\!\!\oint d\theta d\zeta\ u \left( R_\theta \sum_j R_j \cos\alpha_j + Z_\theta \sum_j Z_j \sin\alpha_j \right). \tag{212}
$$

- The "tangential spectral-width descent direction" is thus

$$
\frac{\partial u}{\partial t} = -\left[ R_\theta \sum_j (\lambda_j - M) R_j \cos\alpha_j / D + Z_\theta \sum_j (\lambda_j - M) Z_j \sin\alpha_j / D \right]. \tag{213}
$$

- This suggests that position should be advanced according to

$$
\frac{\partial R_j}{\partial t} \equiv -\frac{\partial}{\partial R_j} \sum_{l=1}^{N} \int \left( \frac{p}{\gamma - 1} + \frac{B^2}{2} \right) dv - [R_\theta(R_\theta X + Z_\theta Y)]_j, \tag{214}
$$

$$
\frac{\partial Z_j}{\partial t} \equiv -\frac{\partial}{\partial Z_j} \sum_{l=1}^{N} \int \left( \frac{p}{\gamma - 1} + \frac{B^2}{2} \right) dv - [Z_\theta(R_\theta X + Z_\theta Y)]_j, \tag{215}
$$

where $X \equiv \sum_j (\lambda_j - M) R_j \cos \alpha_j / D$ and $Y \equiv \sum_j (\lambda_j - M) Z_j \sin \alpha_j / D$.

**numerical implementation**

- The spectral condensation terms,

$$
R_\theta(R_\theta X + Z_\theta Y) = \sum_{j,k,l} m_j m_k (\lambda_l - M) R_j (+R_k R_l \sin \alpha_j \sin \alpha_k \cos \alpha_l - Z_k Z_l \sin \alpha_j \cos \alpha_k \sin \alpha_l), \tag{216}
$$

$$
Z_\theta(R_\theta X + Z_\theta Y) = \sum_{j,k,l} m_j m_k (\lambda_l - M) Z_j (-R_k R_l \cos \alpha_j \sin \alpha_k \cos \alpha_l + Z_k Z_l \cos \alpha_j \cos \alpha_k \sin \alpha_l), \tag{217}
$$

are calculated using triple angle expressions...

**Todo** IT IS VERY LIKELY THAT FFTs WOULD BE FASTER!!!

References allglobal::cpus, allglobal::dbbdrz, dforce(), allglobal::diidrz, inputlist::epsilon, allglobal::forceerr, inputlist::forcetol, constants::half, inputlist::igeometry, allglobal::lbbintegral, allglobal::myid, inputlist::nvol, constants::one, fileunits::ounit, allglobal::yesstellsym, and constants::zero.

Referenced by pc00aa().

Here is the call graph for this function:
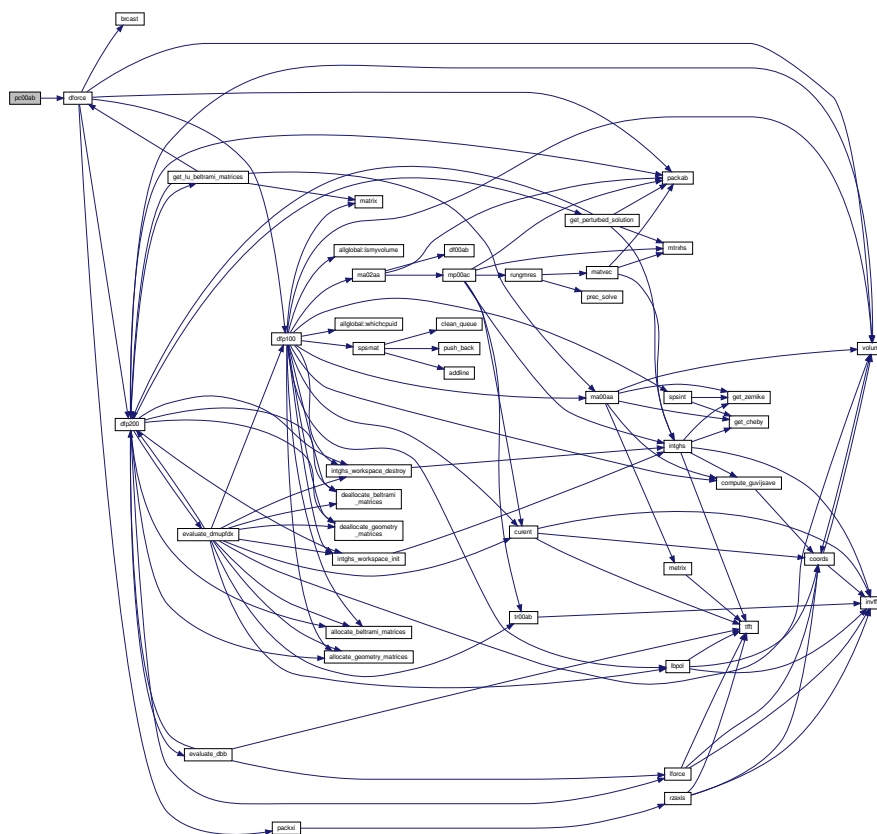
Here is the caller graph for this function:

## 8.18 Initialization of the code

### Functions/Subroutines

- subroutine preset

    *Allocates and initializes internal arrays.*

### 8.18.1 Detailed Description

### 8.18.2 Function/Subroutine Documentation

#### 8.18.2.1 preset() subroutine preset

Allocates and initializes internal arrays.

**LGdof and NGdof : number of geometrical degrees-of-freedom**

- LGdof $\equiv$ the number of degrees-of-freedom in the geometry (i.e. Fourier harmonics) of each interface

- NGdof $\equiv$ total number of degrees-of-freedom in geometry, i.e. of all interfaces

**iota and oita: rotational transform on interfaces**

- The input variables iota and oita are the rotational transform on "inner-side" and on the "outer-side" of each interface.

- These quantities are formally inputs.

- Note that if $q_l + \gamma q_r \neq 0$, then iota is given by

$$\iota \equiv \frac{p_l + \gamma p_r}{q_l + \gamma q_r}, \tag{218}$$

    where $p_l \equiv \text{pl}$, $q_l \equiv \text{ql}$ , etc.; and similarly for oita .

**dtflux(1:Mvol) and dpflux(1:Mvol): enclosed fluxes**

- dtflux $\equiv \Delta\psi_{tor}/2\pi$ and dpflux $\equiv \Delta\psi_{pol}/2\pi$ in each volume.

- Note that the total toroidal flux enclosed by the plasma boundary is $\Phi_{edge} \equiv \text{phiedge}$ .

- $\psi_{tor} \equiv \text{tflux}$ and $\psi_{pol} \equiv \text{pflux}$ are immediately normalized (in readin() ) according to $\psi_{tor,i} \rightarrow \psi_{tor,i}/\psi_0$ and $\psi_{pol,i} \rightarrow \psi_{pol,i}/\psi_0$, where $\psi_0 \equiv \psi_{tor,N}$ on input.

**sweight(1:Mvol): star-like angle constraint weight**

- the "star-like" poloidal angle constraint weights (only required for toroidal geometry, i.e. `Igeometry=3`) are given by

$$\texttt{sweight}_v \equiv \texttt{upsilon} \times (l_v/N_{vol})^w, \tag{219}$$

where $l_v$ is the volume number, and $w \equiv \texttt{wpoloidal}$.

### TT(0:Mrad,0:1,0:1): Chebyshev polynomials at inner/outer interface

- `TT(0:Lrad,0:1,0:1)` gives the Chebyshev polynomials, and their first derivative, evaluated at $s = -1$ and $s = +1$.

- Precisely, `TT(l,i,d)` $\equiv T_l^{(d)}(s_i)$ for $s_0 = -1$ and $s_1 = +1$.

- Note that $T_l^{(0)}(s) = s^l$ and $T_l^{(1)}(s) = s^{l+1}l^2$ for $s = \pm 1$.

- Note that

$$T_l(-1) = \begin{cases} +1, & \text{if } l \text{ is even,} \\ -1, & \text{if } l \text{ is odd;} \end{cases} \qquad T_l(+1) = \begin{cases} +1, & \text{if } l \text{ is even,} \\ +1, & \text{if } l \text{ is odd;} \end{cases} \tag{220}$$

$$T_l'(-1) = \begin{cases} -l^2, & \text{if } l \text{ is even,} \\ +l^2, & \text{if } l \text{ is odd;} \end{cases} \qquad T_l'(+1) = \begin{cases} +l^2, & \text{if } l \text{ is even,} \\ +l^2, & \text{if } l \text{ is odd.} \end{cases} \tag{221}$$

- `TT(0:Mrad,0:1,0:1)` is used in routines that explicity require interface information, such as

    - the interface force-balance routine, lforce()
    - the virtual casing routine, casing()
    - computing the rotational-transform on the interfaces, tr00ab()
    - computing the covariant components of the interface magnetic field, sc00aa()
    - enforcing the constraints on the Beltrami fields, matrix() and
    - computing the enclosed currents of the vacuum field, curent().

### ImagneticOK(1:Mvol): Beltrami/vacuum error flag

- error flags that indicate if the magnetic field in each volume has been successfully constructed

- `ImagneticOK` is initialized to .false. in dforce() before the Beltrami solver routines are called. If the construction of the Beltrami field is successful (in either ma02aa() or mp00ac() ) then `ImagneticOK` is set to .true. .

### Lhessianallocated

- The internal logical variable, `Lhessianallocated`, indicates whether the ``Hessian'' matrix of second-partial derivatives (really, the first derivatives of the force-vector) has been allocated, or not!

### ki(1:mn,0:1): Fourier identification

- Consider the "abbreviated" representation for a double Fourier series,

$$\sum_i f_i \cos(m_i\theta - n_i\zeta) \equiv \sum_{n=0}^{N_0} f_{0,n}\cos(-n\zeta) + \sum_{m=1}^{M_0}\sum_{n=-N_0}^{N_0} f_{m,n}\cos(m\theta - n\zeta), \tag{222}$$

and the same representation but with enhanced resolution,

$$\sum_k \bar{f}_k \cos(\bar{m}_k\theta - \bar{n}_k\zeta) \equiv \sum_{n=0}^{N_1} f_{0,n}\cos(-n\zeta) + \sum_{m=1}^{M_1}\sum_{n=-N_1}^{N_1} f_{m,n}\cos(m\theta - n\zeta), \tag{223}$$

with $M_1 \geq M_0$ and $N_1 \geq N_0$; then $k_i \equiv \texttt{ki(i,0)}$ is defined such that $\bar{m}_{k_i} = m_i$ and $\bar{n}_{k_i} = n_i$.

**kija(1:mn,1:mn,0:1), kijs(1:mn,1:mn,0:1): Fourier identification**

- Consider the following quantities, which are computed in [ma00aa()](), where $\bar{g}^{\mu\nu} = \sum_k \bar{g}_k^{\mu\nu} \cos\alpha_k$ for $\alpha_k \equiv m_k\theta - n_k\zeta$,

$$\oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} \cos\alpha_i \; \cos\alpha_j \;\; = \;\; \frac{1}{2} \oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} (+\cos\alpha_{k_{ij+}} + \cos\alpha_{k_{ij-}}), \tag{224}$$

$$\oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} \cos\alpha_i \; \sin\alpha_j \;\; = \;\; \frac{1}{2} \oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} (+\sin\alpha_{k_{ij+}} - \sin\alpha_{k_{ij-}}), \tag{225}$$

$$\oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} \sin\alpha_i \; \cos\alpha_j \;\; = \;\; \frac{1}{2} \oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} (+\sin\alpha_{k_{ij+}} + \sin\alpha_{k_{ij-}}), \tag{226}$$

$$\oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} \sin\alpha_i \; \sin\alpha_j \;\; = \;\; \frac{1}{2} \oint\oint d\theta d\zeta \; \bar{g}^{\mu\nu} (-\cos\alpha_{k_{ij+}} + \cos\alpha_{k_{ij-}}), \tag{227}$$

  where $(m_{k_{ij+}}, n_{k_{ij+}}) = (m_i + m_j, n_i + n_j)$ and $(m_{k_{ij-}}, n_{k_{ij-}}) = (m_i - m_j, n_i - n_j)$; then `kija(i,j,0)` $\equiv k_{ij+}$ and `kijs(i,j,0)` $\equiv k_{ij-}$.

- Note that Eqn. (223) does not include $m < 0$; so, if $m_i - m_j < 0$ then $k_{ij-}$ is re-defined such that $(m_{k_{ij-}}, n_{k_{ij-}}) = (m_j - m_i, n_j - n_i)$; and similarly for the case $m = 0$ and $n < 0$. Also, take care that the sign of the sine harmonics in the above expressions will change for these cases.

**djkp**

**iotakki**

**cheby(0:Lrad,0:2): Chebyshev polynomial workspace**

- `cheby(0:Lrad,0:2)` is global workspace for computing the Chebyshev polynomials, and their derivatives, using the recurrence relations $T_0(s) = 1$, $T_1(s) = s$ and $T_l(s) = 2\, s\, T_{l-1}(s) - T_{l-2}(s)$.

- These are computed as required, i.e. for arbitrary $s$, in [bfield()](), [jo00aa()]() and [ma00aa()]().

- Note that the quantities required for [ma00aa()]() are for fixed $s$, and so these quantities should be precomputed.

**Iquad, gaussianweight, gaussianabscissae: Gauss-Legendre quadrature**

- The volume integrals are computed using a "Fourier" integration over the angles and by Gauss-Legendre quadrature over the radial, i.e. $\int f(s)ds = \sum_k \omega_k f(s_k)$.

- The quadrature resolution in each volume is give by `Iquad(1:Mvol)` which is determined as follows:

    - if `Nquad.gt.0`, then `Iquad(vvol)=Nquad`
    - if `Nquad.le.0` and `.not.Lcoordinatesingularity`, then `Iquad(vvol)=2*Lrad`(vvol)-Nquad
    - if `Nquad.le.0` and `Lcoordinatesingularity`, then `Iquad(vvol)=2*Lrad`(vvol)-Nquad+Mpol

- The Gaussian weights and abscissae are given by `gaussianweight(1:maxIquad,1:Mvol)` and `gaussianabscissae(1:maxIquad,1:Mvol)`, which are computed using modified Numerical Recipes routine [gauleg()]().

- `Iquad`$_v$ is passed through to [ma00aa()]() to compute the volume integrals of the metric elements; also see [jo00aa()](), where `Iquad`$_v$ is used to compute the volume integrals of $||\nabla \times \mathbf{B} - \mu\mathbf{B}||$.

**LBsequad, LBnewton and LBlinear**

- `LBsequad`, `LBnewton` and `LBlinear` depend simply on `LBeltrami` , which is described in global.f90
.

**BBweight(1:mn): weighting of force-imbalance harmonics**

- weight on force-imbalance harmonics;

$$\texttt{BBweight}_i \equiv \texttt{opsilon} \times \exp\left[-\texttt{escale} \times (m_i^2 + n_i^2)\right] \tag{228}$$

- this is only used in dforce() in constructing the force-imbalance vector

**mmpp(1:mn): spectral condensation weight factors**

- spectral condensation weight factors;

$$\texttt{mmpp(i)} \equiv m_i^p, \tag{229}$$

where $p \equiv \texttt{pcondense}$ .

**NAdof, Ate, Aze, Ato and Azo: degrees-of-freedom in magnetic vector potential**

- `NAdof(1:Mvol)` $\equiv$ total number of degrees-of-freedom in magnetic vector potential, including Lagrange multipliers, in each volume. This can de deduced from matrix().

- The components of the vector potential, $\mathbf{A} = A_\theta \nabla + A_\zeta \nabla\zeta$, are

$$A_\theta(s,\theta,\zeta) = \sum_{i,l} A_{\theta,e,i,l}\, \overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} A_{\theta,o,i,l}\, \overline{T}_{l,i}(s)\sin\alpha_i, \tag{230}$$

$$A_\zeta(s,\theta,\zeta) = \sum_{i,l} A_{\zeta,e,i,l}\, \overline{T}_{l,i}(s)\cos\alpha_i + \sum_{i,l} A_{\zeta,o,i,l}\, \overline{T}_{l,i}(s)\sin\alpha_i, \tag{231}$$

where $\overline{T}_{l,i}(s) \equiv \bar{s}^{m_i/2}\, T_l(s)$, $T_l(s)$ is the Chebyshev polynomial, and $\alpha_j \equiv m_j\theta - n_j\zeta$. The regularity factor, $\bar{s}^{m_i/2}$, where $\bar{s} \equiv (1+s)/2$, is only included if there is a coordinate singularity in the domain (i.e. only in the innermost volume, and only in cylindrical and toroidal geometry.)

- The Chebyshev-Fourier harmonics of the covariant components of the magnetic vector potential are kept in

$$A_{\theta,e,i,l} \equiv \texttt{Ate(v,0,j)\%s(l)}, \tag{232}$$

$$A_{\zeta,e,i,l} \equiv \texttt{Aze(v,0,j)\%s(l)}, \tag{233}$$

$$A_{\theta,o,i,l} \equiv \texttt{Ato(v,0,j)\%s(l)}, \text{and} \tag{234}$$

$$A_{\zeta,o,i,l} \equiv \texttt{Azo(v,0,j)\%s(l)}; \tag{235}$$

where $v = 1, \texttt{Mvol}$ labels volume, $j = 1, \texttt{mn}$ labels Fourier harmonic, and $l = 0, \texttt{Lrad}\,(v)$ labels Chebyshev polynomial. (These arrays also contains derivative information.)

- If `Linitguess=1` , a guess for the initial state for the Beltrami fields is constructed. An initial state is required for iterative solvers of the Beltrami fields, see `LBeltrami` .

- If `Linitguess=2` , the initial state for the Beltrami fields is read from file (see ra00aa() ). An initial state is required for iterative solvers of the Beltrami fields, see `LBeltrami` .

**workspace**

**goomne, goomno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**gssmne, gssmno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**gstmne, gstmno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**gszmne, gszmno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**gttmne, gttmno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**gtzmne, gtzmno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**gzzmne, gzzmno: metric information** These are defined in [metrix()](), and used in [ma00aa()]().

**cosi(1:Ntz,1:mn) and sini(1:Ntz,1:mn)**

- Trigonometric factors used in various Fast Fourier transforms, where

$$
\begin{aligned}
\texttt{cosi}_{j,i} &= \cos(m_i\theta_j - n_i\zeta_j), \quad\quad (236)\\
\texttt{sini}_{j,i} &= \sin(m_i\theta_j - n_i\zeta_j). \quad\quad (237)
\end{aligned}
$$

**psifactor(1:mn,1:Mvol): coordinate "pre-conditioning" factor**

- In toroidal geometry, the coordinate "pre-conditioning" factor is

$$
f_{j,v} \equiv \begin{cases} \psi_{t,v}^{0} & , \quad \text{for } m_j = 0, \\ \psi_{t,v}^{m_j/2} & , \quad \text{otherwise.} \end{cases} \quad\quad (238)
$$

where $\psi_{t,v} \equiv \texttt{tflux}$ is the (normalized?) toroidal flux enclosed by the $v$-th interface.

- $\texttt{psifactor}$ is used in [packxi()](), [dforce()]() and [hesian()]().

- $\texttt{inifactor}$ is similarly constructed, with

$$
f_{j,v} \equiv \begin{cases} \psi_{t,v}^{1/2} & , \quad \text{for } m_j = 0, \\ \psi_{t,v}^{m_j/2} & , \quad \text{otherwise.} \end{cases} \quad\quad (239)
$$

and used only for the initialization of the surfaces taking into account axis information if provided.

**Bsupumn and Bsupvmn**

**diotadxup and glambda: transformation to straight fieldline angle**

- Given the Beltrami fields in any volume, the rotational-transform on the adjacent interfaces may be determined (in [tr00ab()]()) by constructing the straight fieldline angle on the interfaces.

- The rotational transform on the inner or outer interface of a given volume depends on the magnetic field in that volume, i.e. $\iota_{\pm} = \iota(\mathbf{B}_{\pm})$, so that

$$
\delta\iota_{\pm} = \frac{\partial \iota_{\pm}}{\partial \mathbf{B}_{\pm}} \cdot \delta\mathbf{B}_{\pm}. \quad\quad (240)
$$

- The magnetic field depends on the Fourier harmonics of both the inner and outer interface geometry (represented here as $x_j$), the helicity multiplier, and the enclosed poloidal flux, i.e. $\mathbf{B}_\pm = \mathbf{B}_\pm(x_j, \mu, \Delta\psi_p)$, so that

$$\delta\mathbf{B}_\pm = \frac{\partial\mathbf{B}_\pm}{\partial x_j}\delta x_j + \frac{\partial\mathbf{B}_\pm}{\partial\mu}\delta\mu + \frac{\partial\mathbf{B}_\pm}{\partial\Delta\psi_p}\delta\Delta\psi_p. \tag{241}$$

- The rotational-transforms, thus, can be considered to be functions of the geometry, the helicity-multiplier and the enclosed poloidal flux, $\iota_\pm = \iota_\pm(x_j, \mu, \Delta\psi_p)$.

- The rotational-transform, and its derivatives, on the inner and outer interfaces of each volume is stored in `diotadxup(0:1,-1:2,1:Mvol)` . The indices label:

   - the first index labels the inner or outer interface,
   - the the second one labels derivative, with
      * `-1` : indicating the derivative with respect to the interface geometry, i.e. $\dfrac{\partial\iota_\pm}{\partial x_j}$,
      * `0` : the rotational-transform itself,
      * `1,2` : the derivatives with respec to $\mu$ and $\Delta\psi_p$, i.e. $\dfrac{\partial\iota_\pm}{\partial\mu}$ and $\dfrac{\partial\iota_\pm}{\partial\Delta\psi_p}$;
   - The third index labels volume.

- The values of `diotadxup` are assigned in mp00aa() after calling tr00ab().

**vvolume, lBBintegral and lABintegral**

- volume integrals

$$\texttt{vvolume(i)} \;=\; \int_{\mathcal{V}_i} dv \tag{242}$$

$$\texttt{lBBintegral(i)} \;=\; \int_{\mathcal{V}_i} \mathbf{B}\cdot\mathbf{B}\,dv \tag{243}$$

$$\texttt{lABintegral(i)} \;=\; \int_{\mathcal{V}_i} \mathbf{A}\cdot\mathbf{B}\,dv \tag{244}$$

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::bbe, allglobal::bbo, allglobal↩
::bbweight, allglobal::bemn, allglobal::bloweremn, allglobal::bloweromn, bnorml(), allglobal::bomn, allglobal↩
::bsupumn, allglobal::bsupvmn, allglobal::btemn, allglobal::btomn, allglobal::bzemn, allglobal::bzomn, allglobal↩
::cfmn, allglobal::cheby, allglobal::comn, coords(), allglobal::cosi, fftw_interface::cplxin, fftw_interface::cplxout,
allglobal::cpus, allglobal::diotadxup, allglobal::ditgpdxtp, allglobal::djkm, allglobal::djkp, allglobal::dpflux, allglobal↩
::dradr, allglobal::dradz, allglobal::drij, allglobal::drodr, allglobal::drodz, allglobal::dtflux, allglobal::dxyz, allglobal↩
::dzadr, allglobal::dzadz, allglobal::dzij, allglobal::dzodr, allglobal::dzodz, allglobal::efmn, inputlist::escale,
allglobal::evmn, allglobal::fse, allglobal::fso, gauleg(), allglobal::gaussianabscissae, allglobal::gaussianweight,
get_cheby(), get_zernike(), get_zernike_rm(), getimn(), allglobal::glambda, allglobal::gmreslastsolution, allglobal↩
::goomne, allglobal::goomno, allglobal::gssmne, allglobal::gssmno, allglobal::gstmne, allglobal::gstmno, allglobal↩
::gszmne, allglobal::gszmno, allglobal::gteta, allglobal::gttmne, allglobal::gttmno, allglobal::gtzmne, allglobal↩
::gtzmno, allglobal::guvij, allglobal::gvuij, allglobal::gzeta, allglobal::gzzmne, allglobal::gzzmno, allglobal::halfmm,
inputlist::helicity, allglobal::hnt, allglobal::hnz, allglobal::iemn, inputlist::igeometry, allglobal::iie, allglobal::iio,
allglobal::ijimag, allglobal::ijreal, allglobal::im, allglobal::imagneticok, allglobal::ims, allglobal::in, allglobal::inifactor,
allglobal::ins, invfft(), allglobal::iomn, inputlist::iota, allglobal::iotakadd, allglobal::iotakkii, allglobal::iotaksgn,
allglobal::iotaksub, allglobal::ipdt, allglobal::ipdtdpf, allglobal::iquad, allglobal::irbc, allglobal::irbs, allglobal↩
::irij, inputlist::ivolume, allglobal::izbc, allglobal::izbs, allglobal::izij, allglobal::jiimag, allglobal::jireal, allglobal↩
::jkimag, allglobal::jkreal, allglobal::jxyz, allglobal::ki, allglobal::kija, allglobal::kijs, allglobal::kjimag, allglobal::kjreal,
allglobal::labintegral, allglobal::lbbintegral, inputlist::lbeltrami, allglobal::lblinear, allglobal::lbnewton, allglobal↩
::lbsequad, inputlist::lconstraint, allglobal::lcoordinatesingularity, inputlist::lfindzero, inputlist::lfreebound, allglobal↩
::lgdof, inputlist::lgmresprec, allglobal::lhessianallocated, allglobal::liluprecond, inputlist::linitgues, inputlist↩
::linitialize, allglobal::lma, inputlist::lmatsolver, allglobal::lmavalue, allglobal::lmb, allglobal::lmbvalue, allglobal::lmc,

allglobal::lmcvalue, allglobal::lmd, allglobal::lmdvalue, allglobal::lme, allglobal::lmevalue, allglobal::lmf, allglobal←
::lmfvalue, allglobal::lmg, allglobal::lmgvalue, allglobal::lmh, allglobal::lmhvalue, allglobal::lmns, allglobal::lmpol,
allglobal::lntor, allglobal::localconstraint, inputlist::lp, inputlist::lq, inputlist::lrad, matrix(), inputlist::maxrndgues,
allglobal::mmpp, allglobal::mne, allglobal::mns, inputlist::mpol, inputlist::mu, constants::mu0, allglobal::mvol,
allglobal::myid, allglobal::nadof, inputlist::ndiscrete, allglobal::ndmas, allglobal::ndmasmax, allglobal::nfielddof,
inputlist::nfp, allglobal::ngdof, allglobal::notmatrixfree, allglobal::notstellsym, inputlist::nquad, allglobal::nt, inputlist←
::ntor, inputlist::nvol, allglobal::nxyz, allglobal::nz, allglobal::odmn, allglobal::ofmn, inputlist::oita, constants←
::one, inputlist::opsilon, fileunits::ounit, inputlist::pcondense, allglobal::pemn, inputlist::pflux, inputlist::phiedge,
allglobal::pi2nfp, allglobal::pi2pi2nfp, allglobal::pi2pi2nfphalf, allglobal::pi2pi2nfpquart, inputlist::pl, fftw_interface←
::planb, fftw_interface::planf, allglobal::pomn, inputlist::pr, allglobal::psifactor, inputlist::ql, inputlist::qr, ra00aa(),
allglobal::readin(), allglobal::rij, inputlist::rp, inputlist::rq, allglobal::rscale, allglobal::rtm, allglobal::rtt, rzaxis(),
allglobal::semn, allglobal::sfmn, allglobal::sg, allglobal::simn, allglobal::sini, numerical::small, allglobal::smpol,
allglobal::sntor, allglobal::somn, allglobal::sontz, numerical::sqrtmachprec, allglobal::sweight, tfft(), inputlist::tflux,
allglobal::trij, allglobal::tt, allglobal::tzij, inputlist::upsilon, numerical::vsmall, allglobal::vvolume, inputlist::wpoloidal,
allglobal::yesstellsym, allglobal::zernike, constants::zero, and allglobal::zij.

Referenced by hesian(), and xspech().

Here is the call graph for this function:
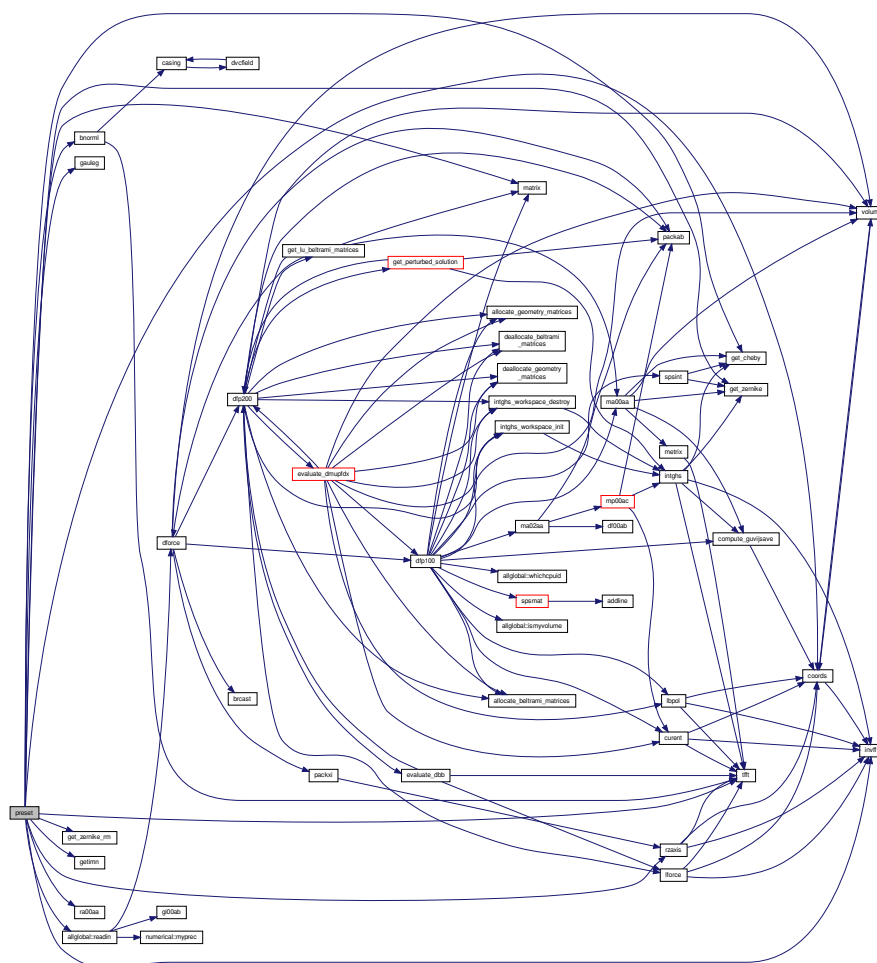
Here is the caller graph for this function:
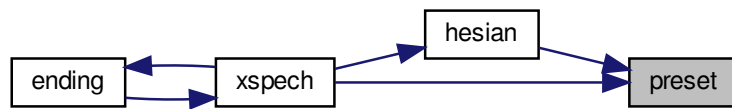
## 8.19 Output file(s)

**Functions/Subroutines**

- subroutine ra00aa (writeorread)

    *Writes vector potential to .ext.sp.A .*

### 8.19.1 Detailed Description

### 8.19.2 Function/Subroutine Documentation

**8.19.2.1 ra00aa()** `subroutine ra00aa (`

`            character, intent(in) writeorread )`

Writes vector potential to .ext.sp.A .

**representation of vector potential**

- The components of the vector potential, $\mathbf{A} = A_\theta \nabla + A_\zeta \nabla \zeta$, are

$$
\begin{array}{rcl}
A_\theta(s, \theta, \zeta) & = & \displaystyle\sum_{i,l} A_{\theta,e,i,l}\, \overline{T}_{l,i}(s) \cos \alpha_i + \sum_{i,l} A_{\theta,o,i,l}\, \overline{T}_{l,i}(s) \sin \alpha_i,
\end{array}
\tag{245}
$$

$$
\begin{array}{rcl}
A_\zeta(s, \theta, \zeta) & = & \displaystyle\sum_{i,l} A_{\zeta,e,i,l}\, \overline{T}_{l,i}(s) \cos \alpha_i + \sum_{i,l} A_{\zeta,o,i,l}\, \overline{T}_{l,i}(s) \sin \alpha_i,
\end{array}
\tag{246}
$$

where $\overline{T}_{l,i}(s) \equiv \bar{s}^{m_i/2}\, T_l(s)$, $T_l(s)$ is the Chebyshev polynomial, and $\alpha_j \equiv m_j \theta - n_j \zeta$. The regularity factor, $\bar{s}^{m_i/2}$, where $\bar{s} \equiv (1+s)/2$, is only included if there is a coordinate singularity in the domain (i.e. only in the innermost volume, and only in cylindrical and toroidal geometry.)

**file format**

- The format of the files containing the vector potential is as follows:

```
open(aunit, file="."//trim(ext)//".sp.A", status="replace", form="unformatted" )
write(aunit) mvol, mpol, ntor, mn, nfp ! integers;
write(aunit) im(1:mn) ! integers; poloidal modes;
write(aunit) in(1:mn) ! integers; toroidal modes;
do vvol = 1, mvol ! integers; loop over volumes;
write(aunit) lrad(vvol) ! integers; the radial resolution in each volume may be different;
do ii = 1, mn
write(aunit) ate(vvol,ii)%s(0:lrad(vvol)) ! reals;
write(aunit) aze(vvol,ii)%s(0:lrad(vvol)) ! reals;
write(aunit) ato(vvol,ii)%s(0:lrad(vvol)) ! reals;
write(aunit) azo(vvol,ii)%s(0:lrad(vvol)) ! reals;
enddo ! end of do ii;
enddo ! end of do vvol;
close(aunit)
```

**Parameters**

| | | |
|---|---|---|
| `in` | *writeorread* | 'W' to write the vector potential; 'R' to read it |

References allglobal::ate, allglobal::ato, fileunits::aunit, allglobal::aze, allglobal::azo, allglobal::cpus, inputlist::ext,

allglobal::im, allglobal::in, inputlist::lrad, inputlist::mpol, allglobal::mvol, allglobal::myid, allglobal::ncpu, inputlist::nfp, inputlist::ntor, fileunits::ounit, inputlist::wmacros, and constants::zero.

Referenced by preset(), and xspech().

Here is the caller graph for this function:
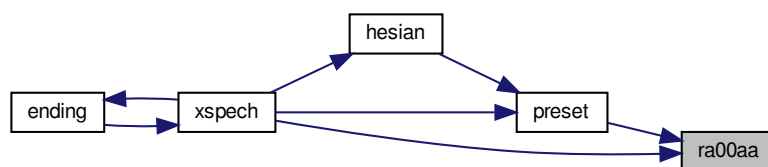
## 8.20 Coordinate axis

**Functions/Subroutines**

- subroutine rzaxis (Mvol, mn, inRbc, inZbs, inRbs, inZbc, ivol, LcomputeDerivatives)

  *The coordinate axis is assigned via a poloidal average over an arbitrary surface.*

### 8.20.1 Detailed Description

### 8.20.2 Function/Subroutine Documentation

#### 8.20.2.1 rzaxis() `subroutine rzaxis (`

```
          integer, intent(in) Mvol,
          integer, intent(in) mn,
          real, dimension(1:mn,0:mvol) inRbc,
          real, dimension(1:mn,0:mvol) inZbs,
          real, dimension(1:mn,0:mvol) inRbs,
          real, dimension(1:mn,0:mvol) inZbc,
          integer, intent(in) ivol,
          logical, intent(in) LcomputeDerivatives )
```

The coordinate axis is assigned via a poloidal average over an arbitrary surface.

Specifies position of coordinate axis; $\mathbf{x}_a(\zeta) \equiv \int \mathbf{x}_1(\theta, \zeta) dl \, / \int dl$.

**coordinate axis**

- The coordinate axis is *not* an independent degree-of-freedom of the geometry. It is constructed by extrapolating the geometry of a given interface, as determined by $i \equiv \text{ivol}$ which is given on input, down to a line.

- If the coordinate axis depends only on the *geometry* of the interface and not the angle parameterization, then the block tri-diagonal structure of the the force-derivative matrix is preserved.

- Define the arc-length-weighted averages,

$$
R_0(\zeta) \equiv \frac{\int_0^{2\pi} R_i(\theta, \zeta) \, dl}{\int_0^{2\pi} dl}, \qquad Z_0(\zeta) \equiv \frac{\int_0^{2\pi} Z_i(\theta, \zeta) \, dl}{\int_0^{2\pi} dl}, \tag{247}
$$

  where $dl \equiv \dot{l} \, d\theta = \sqrt{\partial_\theta R_i(\theta, \zeta)^2 + \partial_\theta Z_i(\theta, \zeta)^2} \, d\theta$.

- (Note that if $\dot{l}$ does not depend on $\theta$, i.e. if $\theta$ is the equal arc-length angle, then the expressions simplify. This constraint is not enforced.)

- The geometry of the coordinate axis thus constructed only depends on the geometry of the interface, i.e. the angular parameterization of the interface is irrelevant.

**coordinate axis: derivatives**

- The derivatives of the coordinate axis with respect to the Fourier harmonics of the given interface are given by

$$\frac{\partial R_0}{\partial R_{i,j}^c} = \int \left( \cos \alpha_j \, \dot{l} - \Delta R_i R_{i,\theta} \, m_j \sin \alpha_j / \, \dot{l} \right) d\theta / L \tag{248}$$

$$\frac{\partial R_0}{\partial R_{i,j}^s} = \int \left( \sin \alpha_j \, \dot{l} + \Delta R_i R_{i,\theta} \, m_j \cos \alpha_j / \, \dot{l} \right) d\theta / L \tag{249}$$

$$\frac{\partial R_0}{\partial Z_{i,j}^c} = \int \left( \qquad -\Delta R_i Z_{i,\theta} \, m_j \sin \alpha_j / \, \dot{l} \right) d\theta / L \tag{250}$$

$$\frac{\partial R_0}{\partial Z_{i,j}^s} = \int \left( \qquad +\Delta R_i Z_{i,\theta} \, m_j \cos \alpha_j / \, \dot{l} \right) d\theta / L \tag{251}$$

$$\frac{\partial Z_0}{\partial R_{i,j}^c} = \int \left( \qquad -\Delta Z_i R_{i,\theta} \, m_j \sin \alpha_j / \, \dot{l} \right) d\theta / L \tag{252}$$

$$\frac{\partial Z_0}{\partial R_{i,j}^s} = \int \left( \qquad +\Delta Z_i R_{i,\theta} \, m_j \cos \alpha_j / \, \dot{l} \right) d\theta / L \tag{253}$$

$$\frac{\partial Z_0}{\partial Z_{i,j}^c} = \int \left( \cos \alpha_j \, \dot{l} - \Delta Z_i Z_{i,\theta} \, m_j \sin \alpha_j / \, \dot{l} \right) d\theta / L \tag{254}$$

$$\frac{\partial Z_0}{\partial Z_{i,j}^s} = \int \left( \sin \alpha_j \, \dot{l} + \Delta Z_i Z_{i,\theta} \, m_j \cos \alpha_j / \, \dot{l} \right) d\theta / L \tag{255}$$

where $L(\zeta) \equiv \int_0^{2\pi} dl$.

**some numerical comments**

- First, the differential poloidal length, $\dot{l} \equiv \sqrt{R_\theta^2 + Z_\theta^2}$, is computed in real space using an inverse FFT from the Fourier harmonics of $R$ and $Z$.

- Second, the Fourier harmonics of $dl$ are computed using an FFT. The integration over $\theta$ to construct $L \equiv \int dl$ is now trivial: just multiply the $m = 0$ harmonics of $dl$ by $2\pi$. The `ajk(1:mn)` variable is used, and this is assigned in readin() .

- Next, the weighted $R \, dl$ and $Z \, dl$ are computed in real space, and the poloidal integral is similarly taken.

- Last, the Fourier harmonics are constructed using an FFT after dividing in real space.

**Parameters**

| | | |
|---|---|---|
| in | *Mvol* | |
| in | *mn* | |
| | *iRbc* | |
| | *iZbs* | |
| | *iRbs* | |
| | *iZbc* | |
| in | *ivol* | |
| | *LcomputeDerivatives* | |

References allglobal::ajk, allglobal::cfmn, allglobal::comn, coords(), allglobal::cosi, allglobal::cpus, allglobal::dbdx, allglobal::dradr, allglobal::dradz, allglobal::drodr, allglobal::drodz, allglobal::dzadr, allglobal::dzadz, allglobal::dzodr, allglobal::dzodz, allglobal::efmn, allglobal::evmn, constants::half, inputlist::igeometry, allglobal::ijimag, allglobal↩::ijreal, allglobal::im, allglobal::in, invfft(), allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal::izbs, allglobal::jiimag, allglobal::jireal, allglobal::jkimag, allglobal::jkreal, allglobal::kjimag, allglobal::kjreal, inputlist::lcheck, allglobal↩::lcoordinatesingularity, inputlist::lreflect, inputlist::lrzaxis, allglobal::myid, allglobal::ncpu, allglobal::notstellsym,

allglobal::nt, inputlist::ntor, inputlist::ntoraxis, allglobal::nz, allglobal::odmn, allglobal::ofmn, constants::one, fileunits::ounit, allglobal::rij, allglobal::sfmn, allglobal::sg, allglobal::simn, allglobal::sini, tfft(), constants::two, numerical::vsmall, inputlist::wmacros, allglobal::yesstellsym, constants::zero, and allglobal::zij.

Referenced by packxi(), and preset().

Here is the call graph for this function:
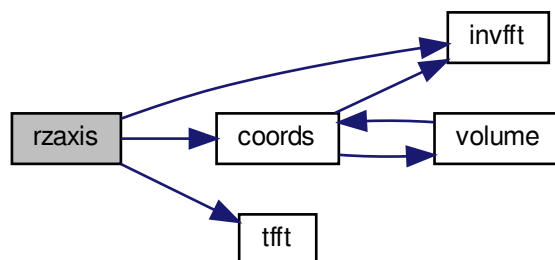


Here is the caller graph for this function:
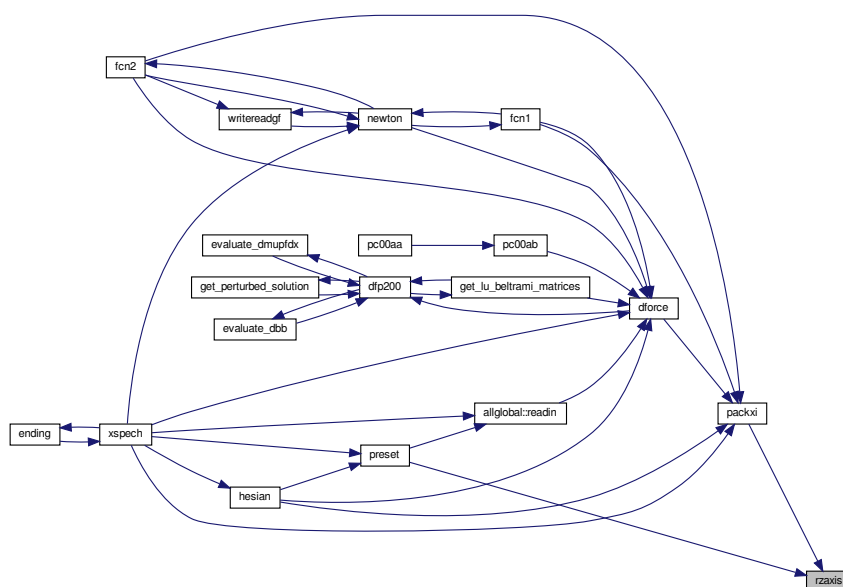
## 8.21 Rotational Transform

**Functions/Subroutines**

- subroutine tr00ab (lvol, mn, NN, Nt, Nz, iflag, ldiota)

  *Calculates rotational transform given an arbitrary tangential field.*

### 8.21.1 Detailed Description

### 8.21.2 Function/Subroutine Documentation

**8.21.2.1 tr00ab()** `subroutine tr00ab (`
```
        integer, intent(in) lvol,
        integer, intent(in) mn,
        integer, intent(in) NN,
        integer, intent(in) Nt,
        integer, intent(in) Nz,
        integer, intent(in) iflag,
        real, dimension(0:1,-1:2), intent(inout) ldiota )
```

Calculates rotational transform given an arbitrary tangential field.

Calculates transform, $\iota = \dot{\theta}(1 + \lambda_\theta) + \lambda_\zeta$, given $\mathbf{B}|_{\mathcal{I}}$.

**constructing straight field line angle on interfaces**

- The algorithm stems from introducing a straight field line angle $\theta_s = \theta + \lambda(\theta, \zeta)$, where

$$\lambda = \sum_j \lambda_{o,j} \sin(m_j\theta - n_j\zeta) + \sum_j \lambda_{e,j} \cos(m_j\theta - n_j\zeta) \tag{256}$$

and insisting that

$$\frac{\mathbf{B} \cdot \nabla\theta_s}{\mathbf{B} \cdot \nabla\zeta} = \dot{\theta}(1 + \lambda_\theta) + \lambda_\zeta = \iota, \tag{257}$$

where $\iota$ is a constant that is to be determined.

- Writing $\dot{\theta} = -\partial_s A_\zeta / \partial_s A_\theta$, we have

$$\partial_s A_\theta \, \iota + \partial_s A_\zeta \, \lambda_\theta - \partial_s A_\theta \, \lambda_\zeta = -\partial_s A_\zeta \tag{258}$$

- Expanding this equation we obtain

$$
\begin{aligned}
& \left(A'_{\theta,e,k}\cos\alpha_k + A'_{\theta,o,k}\sin\alpha_k\right) \, \iota \\
+ \ & \left(A'_{\zeta,e,k}\cos\alpha_k + A'_{\zeta,o,k}\sin\alpha_k\right)\left(+m_j\lambda_{o,j}\cos\alpha_j - m_j\lambda_{e,j}\sin\alpha_j\right) \\
- \ & \left(A'_{\theta,e,k}\cos\alpha_k + A'_{\theta,o,k}\sin\alpha_k\right)\left(-n_j\lambda_{o,j}\cos\alpha_j + n_j\lambda_{e,j}\sin\alpha_j\right) \\
= \ - & \left(A'_{\zeta,e,k}\cos\alpha_k + A'_{\zeta,o,k}\sin\alpha_k\right),
\end{aligned} \tag{259}
$$

where summation over $k = 1, \mathrm{mn}$ and $j = 2, \mathrm{mns}$ is implied

- After applying double angle formulae,

$$
\begin{aligned}
& \left(A'_{\theta,e,k}\cos\alpha_k + A'_{\theta,o,k}\sin\alpha_k\right)\ \mathsf{t} \\
+\ & \lambda_{o,j}\left(+m_j A'_{\zeta,e,k} + n_j A'_{\theta,e,k}\right)\left[+\cos(\alpha_k+\alpha_j)+\cos(\alpha_k-\alpha_j)\right]/2 \\
+\ & \lambda_{e,j}\left(-m_j A'_{\zeta,e,k} - n_j A'_{\theta,e,k}\right)\left[+\sin(\alpha_k+\alpha_j)-\sin(\alpha_k-\alpha_j)\right]/2 \\
+\ & \lambda_{o,j}\left(+m_j A'_{\zeta,o,k} + n_j A'_{\theta,o,k}\right)\left[+\sin(\alpha_k+\alpha_j)+\sin(\alpha_k-\alpha_j)\right]/2 \\
+\ & \lambda_{e,j}\left(-m_j A'_{\zeta,o,k} - n_j A'_{\theta,o,k}\right)\left[-\cos(\alpha_k+\alpha_j)+\cos(\alpha_k-\alpha_j)\right]/2 \\
=\ -\ & \left(A'_{\zeta,e,k}\cos\alpha_k + A'_{\zeta,o,k}\sin\alpha_k\right),
\end{aligned}
\tag{260}
$$

and equating coefficients, an equation of the form $\mathbf{A}\cdot\mathbf{x}=\mathbf{b}$ is obtained, where

$$
\mathbf{x} = (\ \underbrace{\mathsf{t}}_{\texttt{x[1]}}\ ,\ \underbrace{\lambda_{o,2},\lambda_{o,3},\dots}_{\texttt{x[ 2: N ]}}\ ,\ \underbrace{\lambda_{e,2},\lambda_{e,3},\dots}_{\texttt{x[N+1:2N-1]}}\ )^T.
\tag{261}
$$

**alternative iterative method**

- Consider the equation $\dot\theta(1+\lambda_\theta)+\lambda_\zeta = \mathsf{t}$, where $\lambda = \sum_j \lambda_j \sin\alpha_j$, given on a grid

$$
\dot\theta_i + \dot\theta_i \sum_j m_j \cos\alpha_{i,j}\lambda_j - \sum_j n_j \cos\alpha_{i,j}\lambda_j = \mathsf{t},
\tag{262}
$$

where $i$ labels the grid point.

- This is a matrix equation...

**Parameters**

| | |
|---|---|
| *lvol* | |
| *mn* | |
| *NN* | |
| *Nt* | |
| *Nz* | |
| *iflag* | |
| *ldiota* | |

References allglobal::cpus, allglobal::glambda, constants::goldenmean, constants::half, allglobal::im, inputlist↵::imethod, allglobal::ims, allglobal::in, allglobal::ins, invfft(), inputlist::iorder, inputlist::iotatol, inputlist::iprecon, inputlist::lrad, inputlist::lsparse, inputlist::lsvdiota, numerical::machprec, allglobal::mns, inputlist::mpol, allglobal↵::mvol, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, inputlist::ntor, inputlist::nvol, constants::one, fileunits↵::ounit, constants::pi2, allglobal::pi2nfp, numerical::small, numerical::sqrtmachprec, constants::third, constants::two, numerical::vsmall, inputlist::wmacros, allglobal::yesstellsym, and constants::zero.

Referenced by evaluate_dmupfdx(), and mp00ac().

Here is the call graph for this function:

Here is the caller graph for this function:
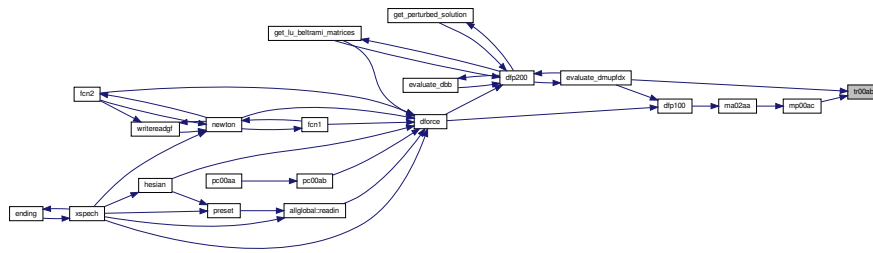
## 8.22 Plasma volume

**Functions/Subroutines**

- subroutine volume (lvol, vflag)

   *Computes volume of each region; and, if required, the derivatives of the volume with respect to the interface geometry.*

### 8.22.1 Detailed Description

### 8.22.2 Function/Subroutine Documentation

**8.22.2.1 volume()** ` subroutine volume (`
   ` integer, intent(in) ` *lvol,*
   ` integer ` *vflag* ` )`

Computes volume of each region; and, if required, the derivatives of the volume with respect to the interface geometry.

Calculates volume of each region; $\mathcal{V}_i \equiv \int dv$.

**volume integral**

- The volume enclosed by the $v$-th interface is given by the integral

$$V = \int_{\mathcal{V}} dv = \frac{1}{3} \int_{\mathcal{V}} \nabla \cdot \mathbf{x} \, dv = \frac{1}{3} \int_{\mathcal{S}} \mathbf{x} \cdot d\mathbf{s} = \frac{1}{3} \int_0^{2\pi} d\theta \int_0^{2\pi/N} d\zeta \quad \mathbf{x} \cdot \mathbf{x}_\theta \times \mathbf{x}_\zeta \big|^s \tag{263}$$

   where we have used $\nabla \cdot \mathbf{x} = 3$, and have assumed that the domain is periodic in the angles.

**representation of surfaces**

- The coordinate functions are

$$R(\theta, \zeta) = \sum_i R_{e,i} \, \cos\alpha_i + \sum_i R_{o,i} \, \sin\alpha_i \tag{264}$$

$$Z(\theta, \zeta) = \sum_i Z_{e,i} \, \cos\alpha_i + \sum_i Z_{o,i} \, \sin\alpha_i, \tag{265}$$

   where $\alpha_i \equiv m_i\theta - n_i\zeta$.

**geometry**

- The geometry is controlled by the input parameter `Igeometry` as follows:

- `Igeometry.eq.1` : Cartesian : $\sqrt{g} = R_s$

$$\begin{aligned} V &= \int_0^{2\pi} d\theta \int_0^{2\pi/N} d\zeta \, R \\ &= 2\pi \, \frac{2\pi}{N} \, R_{e,1} \end{aligned} \tag{266}$$

- `Igeometry.eq.2` : cylindrical : $\sqrt{g} = RR_s = \frac{1}{2}\partial_s(R^2)$

$$
\begin{aligned}
V &= \frac{1}{2}\int_0^{2\pi} d\theta \int_0^{2\pi/N} d\zeta \ R^2 \\
&= \frac{1}{2} \ 2\pi \ \frac{2\pi}{N} \ \frac{1}{2} \ \sum_i \sum_j R_{e,i} R_{e,j} \left[\cos(\alpha_i - \alpha_j) + \cos(\alpha_i + \alpha_j)\right] \\
&+ \frac{1}{2} \ 2\pi \ \frac{2\pi}{N} \ \frac{1}{2} \ \sum_i \sum_j R_{o,i} R_{o,j} \left[\cos(\alpha_i - \alpha_j) - \cos(\alpha_i + \alpha_j)\right]
\end{aligned}
\tag{267}
$$

- `Igeometry.eq.3` : toroidal : $\mathbf{x} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta = R(ZR_\theta - RZ_\theta)$ This is computed by fast Fourier transform:

$$
\begin{aligned}
V &= \frac{1}{3}\int_0^{2\pi} d\theta \int_0^{2\pi/N} d\zeta \ R\left(ZR_\theta - RZ_\theta\right) \\
&= \frac{1}{3} \sum_i \sum_j \sum_k R_{e,i}\left(Z_{e,j}R_{o,k} - R_{e,j}Z_{o,k}\right)(+m_k)\iint d\theta d\zeta \ \cos\alpha_i \cos\alpha_j \cos\alpha_k \\
&+ \frac{1}{3} \sum_i \sum_j \sum_k R_{e,i}\left(Z_{o,j}R_{e,k} - R_{o,j}Z_{e,k}\right)(-m_k)\iint d\theta d\zeta \ \cos\alpha_i \sin\alpha_j \sin\alpha_k \\
&+ \frac{1}{3} \sum_i \sum_j \sum_k R_{o,i}\left(Z_{e,j}R_{e,k} - R_{e,j}Z_{e,k}\right)(-m_k)\iint d\theta d\zeta \ \sin\alpha_i \cos\alpha_j \sin\alpha_k \\
&+ \frac{1}{3} \sum_i \sum_j \sum_k R_{o,i}\left(Z_{o,j}R_{o,k} - R_{o,j}Z_{o,k}\right)(+m_k)\iint d\theta d\zeta \ \sin\alpha_i \sin\alpha_j \cos\alpha_k
\end{aligned}
\tag{268}
$$

- (Recall that the integral over an odd function is zero, so various terms in the above expansion have been ignored.)

- The trigonometric terms are

$$
\begin{aligned}
4\cos\alpha_i \cos\alpha_j \cos\alpha_k &= + \ \cos(\alpha_i + \alpha_j + \alpha_k) \ + \ \cos(\alpha_i + \alpha_j - \alpha_k) \ + \ \cos(\alpha_i - \alpha_j + \alpha_k) \ + \ \cos(\alpha_i - \alpha_j - \alpha_k) \\
4\cos\alpha_i \sin\alpha_j \sin\alpha_k &= - \ \cos(\alpha_i + \alpha_j + \alpha_k) \ + \ \cos(\alpha_i + \alpha_j - \alpha_k) \ + \ \cos(\alpha_i - \alpha_j + \alpha_k) \ - \ \cos(\alpha_i - \alpha_j - \alpha_k) \\
4\sin\alpha_i \cos\alpha_j \sin\alpha_k &= - \ \cos(\alpha_i + \alpha_j + \alpha_k) \ + \ \cos(\alpha_i + \alpha_j - \alpha_k) \ - \ \cos(\alpha_i - \alpha_j + \alpha_k) \ + \ \cos(\alpha_i - \alpha_j - \alpha_k) \\
4\sin\alpha_i \sin\alpha_j \cos\alpha_k &= - \ \cos(\alpha_i + \alpha_j + \alpha_k) \ - \ \cos(\alpha_i + \alpha_j - \alpha_k) \ + \ \cos(\alpha_i - \alpha_j + \alpha_k) \ + \ \cos(\alpha_i - \alpha_j - \alpha_k)
\end{aligned}
$$

- The required derivatives are

$$
\begin{aligned}
3\frac{\partial V}{\partial R_{e,i}} &= \left(+Z_{e,j}R_{o,k}m_k - R_{e,j}Z_{o,k}m_k - R_{e,j}Z_{o,k}m_k\right) \quad \iint d\theta d\zeta \ \cos\alpha_i \cos\alpha_j \cos\alpha_k \\
&+ \left(-Z_{o,j}R_{e,k}m_k + R_{o,j}Z_{e,k}m_k + R_{o,j}Z_{e,k}m_k\right) \quad \iint d\theta d\zeta \ \cos\alpha_i \sin\alpha_j \sin\alpha_k \\
&+ \left(-R_{o,k}Z_{e,j}m_i\right) \qquad\qquad\qquad\qquad\qquad \iint d\theta d\zeta \ \sin\alpha_i \cos\alpha_j \sin\alpha_k \\
&+ \left(-R_{e,k}Z_{o,j}m_i\right) \qquad\qquad\qquad\qquad\qquad \iint d\theta d\zeta \ \sin\alpha_i \sin\alpha_j \cos\alpha_k
\end{aligned}
\tag{270}
$$

$$
\begin{aligned}
3\frac{\partial V}{\partial Z_{o,i}} &= \left(-R_{e,k}R_{e,j}m_i\right) \quad \iint d\theta d\zeta \ \cos\alpha_i \cos\alpha_j \cos\alpha_k \\
&+ \left(-R_{o,k}R_{o,j}m_i\right) \quad \iint d\theta d\zeta \ \cos\alpha_i \sin\alpha_j \sin\alpha_k \\
&+ \left(-R_{e,j}R_{e,k}m_k\right) \quad \iint d\theta d\zeta \ \sin\alpha_i \cos\alpha_j \sin\alpha_k \\
&+ \left(+R_{o,j}R_{o,k}m_k\right) \quad \iint d\theta d\zeta \ \sin\alpha_i \sin\alpha_j \cos\alpha_k
\end{aligned}
\tag{271}
$$

References coords(), allglobal::cosi, allglobal::cpus, allglobal::dbdx, allglobal::djkm, allglobal::djkp, allglobal↩
::dvolume, constants::four, constants::half, inputlist::igeometry, allglobal::im, allglobal::in, allglobal::irbc, allglobal↩
::irbs, allglobal::izbc, allglobal::izbs, allglobal::mvol, allglobal::myid, inputlist::nvol, constants::one, fileunits::ounit,
constants::pi2, allglobal::pi2nfp, allglobal::pi2pi2nfp, allglobal::pi2pi2nfpquart, inputlist::pscale, constants::quart,
allglobal::rij, allglobal::sini, numerical::small, constants::third, constants::two, numerical::vsmall, allglobal::vvolume,
allglobal::yesstellsym, constants::zero, and allglobal::zij.

Referenced by coords(), dforce(), dfp100(), dfp200(), evaluate_dmupfdx(), fcn2(), sphdf5::hdfint(), ma00aa(),
sphdf5::mirror_input_to_outfile(), pp00ab(), and xspech().

Here is the call graph for this function:
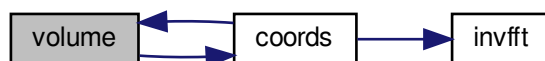


Here is the caller graph for this function:
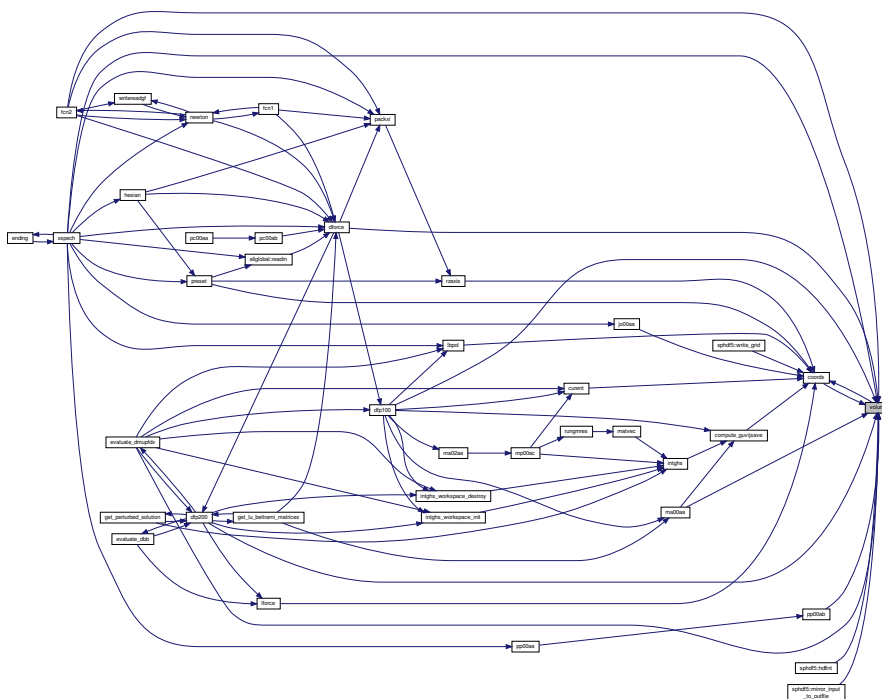
## 8.23 Smooth boundary

**Functions/Subroutines**

- subroutine wa00aa (iwa00aa)

  *Constructs smooth approximation to wall.*

- subroutine vacuumphi (Nconstraints, rho, fvec, iflag)

  *Compute vacuum magnetic scalar potential (?)*

### 8.23.1 Detailed Description

### 8.23.2 Function/Subroutine Documentation

#### 8.23.2.1 wa00aa()  `subroutine wa00aa (`
`            integer iwa00aa )`

Constructs smooth approximation to wall.

**solution of Laplace's equation in two-dimensions**

- The wall is given by a discrete set of points.

- The points must go anti-clockwise.

References allglobal::cpus, fileunits::gunit, constants::half, allglobal::im, allglobal::in, allglobal::irbc, allglobal↩
::irbs, allglobal::izbc, allglobal::izbs, allglobal::lcoordinatesingularity, inputlist::mpol, allglobal::mvol, allglobal↩
::myid, allglobal::ncpu, allglobal::nt, inputlist::ntor, inputlist::nvol, allglobal::nz, inputlist::odetol, constants::one,
fileunits::ounit, constants::pi2, allglobal::rij, constants::ten, tfft(), vacuumphi(), numerical::vsmall, inputlist::wmacros,
allglobal::yesstellsym, constants::zero, and allglobal::zij.

Referenced by vacuumphi().

Here is the call graph for this function:
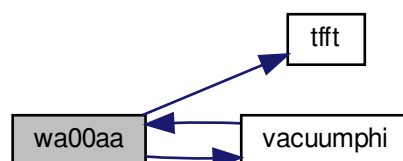
Here is the caller graph for this function:



**8.23.2.2  vacuumphi()**  `subroutine vacuumphi (`
`        integer Nconstraints,`
`        real, dimension(1:nconstraints) rho,`
`        real, dimension(1:nconstraints) fvec,`
`        integer iflag )`

Compute vacuum magnetic scalar potential (?)

**Parameters**

| Nconstraints | |
|---|---|
| rho | |
| fvec | |
| iflag | |

References allglobal::cpus, constants::half, allglobal::myid, allglobal::ncpu, constants::one, fileunits::ounit, constants::pi2, allglobal::rij, wa00aa(), inputlist::wmacros, constants::zero, and allglobal::zij.

Referenced by wa00aa().

Here is the call graph for this function:

Here is the caller graph for this function:
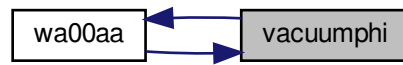
## 8.24 physicslist

The namelist `physicslist` controls the geometry, profiles, and numerical resolution.

Collaboration diagram for physicslist:



**Variables**

- integer inputlist::igeometry = 3

    *selects Cartesian, cylindrical or toroidal geometry;*
- integer inputlist::istellsym = 1

    *stellarator symmetry is enforced if `Istellsym==1`*
- integer inputlist::lfreebound = 0

    *compute vacuum field surrounding plasma*
- real inputlist::phiedge = 1.0

    *total enclosed toroidal magnetic flux;*
- real inputlist::curtor = 0.0

    *total enclosed (toroidal) plasma current;*
- real inputlist::curpol = 0.0

    *total enclosed (poloidal) linking current;*
- real inputlist::gamma = 0.0

    *adiabatic index; cannot set $|\gamma| = 1$*
- integer inputlist::nfp = 1

    *field periodicity*
- integer inputlist::nvol = 1

    *number of volumes*
- integer inputlist::mpol = 0

    *number of poloidal Fourier harmonics*
- integer inputlist::ntor = 0

    *number of toroidal Fourier harmonics*
- integer, dimension(1:mnvol+1) inputlist::lrad = 4

    *Chebyshev resolution in each volume.*
- integer inputlist::lconstraint = -1

    *selects constraints; primarily used in ma02aa() and mp00ac().*
- real, dimension(1:mnvol+1) inputlist::tflux = 0.0

    *toroidal flux, $\psi_t$, enclosed by each interface*
- real, dimension(1:mnvol+1) inputlist::pflux = 0.0

    *poloidal flux, $\psi_p$, enclosed by each interface*
- real, dimension(1:mnvol) inputlist::helicity = 0.0

    *helicity, $\mathcal{K}$, in each volume, $\mathcal{V}_i$*

- real inputlist::pscale = 0.0

  *pressure scale factor*
- real, dimension(1:mnvol+1) inputlist::pressure = 0.0

  *pressure in each volume*
- integer inputlist::ladiabatic = 0

  *logical flag*
- real, dimension(1:mnvol+1) inputlist::adiabatic = 0.0

  *adiabatic constants in each volume*
- real, dimension(1:mnvol+1) inputlist::mu = 0.0

  *helicity-multiplier, $\mu$, in each volume*
- real, dimension(1:mnvol+1) inputlist::ivolume = 0.0

  *Toroidal current constraint normalized by $\mu_0$ ( $I_{volume} = \mu_0 \cdot [A]$), in each volume. This is a cumulative quantity: $I_{\mathcal{V},i} = \int_0^{\psi_{t,i}} \mathbf{J} \cdot \mathbf{dS}$. Physically, it represents the sum of all non-pressure driven currents.*
- real, dimension(1:mnvol) inputlist::isurf = 0.0

  *Toroidal current normalized by $\mu_0$ at each interface (cumulative). This is the sum of all pressure driven currents.*
- integer, dimension(0:mnvol) inputlist::pl = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::ql = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::pr = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::qr = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- real, dimension(0:mnvol) inputlist::iota = 0.0

  *rotational-transform, $\iota$, on inner side of each interface*
- integer, dimension(0:mnvol) inputlist::lp = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::lq = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::rp = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::rq = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- real, dimension(0:mnvol) inputlist::oita = 0.0

  *rotational-transform, $\iota$, on outer side of each interface*
- real inputlist::mupftol = 1.0e-14

  *accuracy to which $\mu$ and $\Delta\psi_p$ are required*
- integer inputlist::mupfits = 8

  *an upper limit on the transform/helicity constraint iterations;*
- real inputlist::rpol = 1.0

  *poloidal extent of slab (effective radius)*
- real inputlist::rtor = 1.0

  *toroidal extent of slab (effective radius)*
- integer inputlist::lreflect = 0

  *=1 reflect the upper and lower bound in slab, =0 do not reflect*
- real, dimension( 0:mntor) inputlist::rac = 0.0

  *stellarator symmetric coordinate axis;*
- real, dimension( 0:mntor) inputlist::zas = 0.0

  *stellarator symmetric coordinate axis;*
- real, dimension( 0:mntor) inputlist::ras = 0.0

  *non-stellarator symmetric coordinate axis;*

- real, dimension( 0:mntor) inputlist::zac = 0.0

  *non-stellarator symmetric coordinate axis;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::rbc = 0.0

  *stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::zbs = 0.0

  *stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::rbs = 0.0

  *non-stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::zbc = 0.0

  *non-stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::rwc = 0.0

  *stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::zws = 0.0

  *stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::rws = 0.0

  *non-stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::zwc = 0.0

  *non-stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::vns = 0.0

  *stellarator symmetric normal field at boundary; vacuum component;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::bns = 0.0

  *stellarator symmetric normal field at boundary; plasma component;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::vnc = 0.0

  *non-stellarator symmetric normal field at boundary; vacuum component;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) inputlist::bnc = 0.0

  *non-stellarator symmetric normal field at boundary; plasma component;*

### 8.24.1 Detailed Description

The namelist `physicslist` controls the geometry, profiles, and numerical resolution.

### 8.24.2 Variable Documentation

#### 8.24.2.1 igeometry `integer inputlist::igeometry = 3`

selects Cartesian, cylindrical or toroidal geometry;

- `Igeometry=1` : Cartesian; geometry determined by $R$;

- `Igeometry=2` : cylindrical; geometry determined by $R$;

- `Igeometry=3` : toroidal; geometry determined by $R$ *and* $Z$;

Referenced by bnorml(), coords(), dforce(), dfp100(), dfp200(), dvcfield(), evaluate_dbb(), evaluate_dmupfdx(), fcn1(), fcn2(), hesian(), jo00aa(), lbpol(), lforce(), sphdf5::mirror_input_to_outfile(), newton(), packxi(), pc00ab(), pp00aa(), preset(), allglobal::readin(), rzaxis(), stzxyz(), volume(), sphdf5::write_grid(), writereadgf(), allglobal←↩ ::wrtend(), and xspech().

**8.24.2.2 nfp** `integer inputlist::nfp = 1`

field periodicity

- all Fourier representations are of the form $\cos(m\theta - nN\zeta)$, $\sin(m\theta - nN\zeta)$, where $N \equiv$ `Nfp`

- constraint: `Nfp` $>=$ 1

Referenced by invfft(), jo00aa(), sphdf5::mirror_input_to_outfile(), preset(), ra00aa(), allglobal::readin(), tfft(), allglobal::wrtend(), and xspech().

**8.24.2.3 nvol** `integer inputlist::nvol = 1`

number of volumes

- each volume $\mathcal{V}_l$ is bounded by the $\mathcal{I}_{l-1}$ and $\mathcal{I}_l$ interfaces

- note that in cylindrical or toroidal geometry, $\mathcal{I}_0$ is the degenerate coordinate axis

- constraint: `Nvol<=MNvol`

Referenced by brcast(), df00ab(), dforce(), dfp100(), dfp200(), dvcfield(), evaluate_dbb(), evaluate_dmupfdx(), sphdf5::hdfint(), hesian(), jo00aa(), lforce(), sphdf5::mirror_input_to_outfile(), packxi(), pc00ab(), pp00aa(), pp00ab(), preset(), allglobal::readin(), stzxyz(), tr00ab(), volume(), wa00aa(), sphdf5::write_grid(), writereadgf(), allglobal::wrtend(), and xspech().

**8.24.2.4 mpol** `integer inputlist::mpol = 0`

number of poloidal Fourier harmonics

- all Fourier representations of doubly-periodic functions are of the form

$$f(\theta,\zeta) \;\; = \;\; \sum_{n=0}^{\text{Ntor}} f_{0,n} \cos(-n\,\text{Nfp}\,\zeta) + \sum_{m=1}^{\text{Mpol}} \sum_{n=-\text{Ntor}}^{\text{Ntor}} f_{m,n} \cos(m\theta - n\,\text{Nfp}\,\zeta), \qquad (272)$$

Internally these "double" summations are written as a "single" summation, e.g. $f(\theta,\zeta) = \sum_j f_j \cos(m_j\theta - n_j\zeta)$.

Referenced by allocate_geometry_matrices(), bfield(), bfield_tangent(), dfp200(), intghs(), intghs_workspace_↩ init(), jo00aa(), ma00aa(), matrix(), sphdf5::mirror_input_to_outfile(), mtrxhs(), preset(), ra00aa(), allglobal::readin(), spsint(), spsmat(), tr00ab(), wa00aa(), writereadgf(), and allglobal::wrtend().

### 8.24.2.5 ntor `integer inputlist::ntor = 0`

number of toroidal Fourier harmonics

- all Fourier representations of doubly-periodic functions are of the form

$$f(\theta, \zeta) = \sum_{n=0}^{\text{Ntor}} f_{0,n} \cos(-n \, \text{Nfp} \, \zeta) + \sum_{m=1}^{\text{Mpol}} \sum_{n=-\text{Ntor}}^{\text{Ntor}} f_{m,n} \cos(m\theta - n \, \text{Nfp} \, \zeta), \quad (273)$$

Internally these "double" summations are written as a "single" summation, e.g. $f(\theta, \zeta) = \sum_j f_j \cos(m_j \theta - n_j \zeta)$.

Referenced by coords(), dforce(), dfp200(), evaluate_dbb(), sphdf5::mirror_input_to_outfile(), mp00ac(), packxi(), preset(), ra00aa(), allglobal::readin(), rzaxis(), stzxyz(), tr00ab(), wa00aa(), writereadgf(), and allglobal::wrtend().

### 8.24.2.6 lrad `integer, dimension(1:mnvol+1) inputlist::lrad = 4`

Chebyshev resolution in each volume.

- constraint : `Lrad(1:Mvol)` $>=$ 2

Referenced by allocate_geometry_matrices(), bfield(), bfield_tangent(), bnorml(), brcast(), curent(), dforce(), dfp100(), dfp200(), dvcfield(), evaluate_dbb(), evaluate_dmupfdx(), get_lu_beltrami_matrices(), get_perturbed_↩ solution(), sphdf5::hdfint(), intghs_workspace_init(), jo00aa(), lbpol(), lforce(), ma02aa(), matvec(), sphdf5::mirror↩ _input_to_outfile(), mp00ac(), packab(), pp00aa(), preset(), ra00aa(), allglobal::readin(), tr00ab(), sphdf5::write_↩ grid(), allglobal::wrtend(), and xspech().

### 8.24.2.7 lconstraint `integer inputlist::lconstraint = -1`

selects constraints; primarily used in ma02aa() and mp00ac().

- if `Lconstraint==-1`, then in the plasma regions $\Delta\psi_t$, $\mu$ and $\Delta\psi_p$ are *not* varied and in the vacuum region (only for free-boundary) $\Delta\psi_t$ and $\Delta\psi_p$ are *not* varied, and $\mu = 0$.

- if `Lconstraint==0`, then in the plasma regions $\Delta\psi_t$, $\mu$ and $\Delta\psi_p$ are *not* varied and in the vacuum region (only for free-boundary) $\Delta\psi_t$ and $\Delta\psi_p$ are varied to match the prescribed plasma current, `curtor`, and the "linking" current, `curpol`, and $\mu = 0$

- if `Lconstraint==1`, then in the plasma regions $\mu$ and $\Delta\psi_p$ are adjusted in order to satisfy the inner and outer interface transform constraints (except in the simple torus, where the enclosed poloidal flux is irrelevant, and only $\mu$ is varied to satisfy the outer interface transform constraint); and in the vacuum region $\Delta\psi_t$ and $\Delta\psi_p$ are varied to match the transform constraint on the boundary and to obtain the prescribed linking current, `curpol`, and $\mu = 0$.

- **Todo** if `Lconstraint==2`, under reconstruction.

- if `Lconstraint.eq.3` , then the $\mu$ and $\psi_p$ variables are adjusted in order to satisfy the volume and surface toroidal current computed with lbpol() (excepted in the inner most volume, where the volume current is irrelevant). Not implemented yet in free boundary.

Referenced by brcast(), dforce(), dfp100(), dfp200(), evaluate_dbb(), evaluate_dmupfdx(), get_lu_beltrami_↩ matrices(), get_perturbed_solution(), ma02aa(), sphdf5::mirror_input_to_outfile(), pp00aa(), preset(), allglobal↩ ::readin(), allglobal::wrtend(), and xspech().

**8.24.2.8 tflux** `real, dimension(1:mnvol+1) inputlist::tflux = 0.0`

toroidal flux, $\psi_t$, enclosed by each interface

- For each of the plasma volumes, this is a constraint: `tflux` is *not* varied

- For the vacuum region (only if `Lfreebound==1`), `tflux` may be allowed to vary to match constraints

- Note that `tflux` will be normalized so that `tflux(Nvol) = 1.0`, so that `tflux` is arbitrary up to a scale factor

  **See also**

- phiedge

Referenced by dfp200(), sphdf5::hdfint(), sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), allglobal←
::wrtend(), and xspech().

**8.24.2.9 helicity** `real, dimension(1:mnvol) inputlist::helicity = 0.0`

helicity, $\mathcal{K}$, in each volume, $\mathcal{V}_i$

- on exit, `helicity` is set to the computed values of $\mathcal{K} \equiv \int \mathbf{A} \cdot \mathbf{B} \, dv$

Referenced by brcast(), df00ab(), sphdf5::hdfint(), hesian(), ma02aa(), sphdf5::mirror_input_to_outfile(), mp00ac(),
preset(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.24.2.10 pscale** `real inputlist::pscale = 0.0`

pressure scale factor

- the initial pressure profile is given by `pscale * pressure`

Referenced by dfp200(), evaluate_dbb(), lforce(), sphdf5::mirror_input_to_outfile(), allglobal::readin(), volume(),
allglobal::wrtend(), and xspech().

**8.24.2.11 pressure** `real, dimension(1:mnvol+1) inputlist::pressure = 0.0`

pressure in each volume

- The pressure is *not* held constant, but $p_l V_l^{\gamma} = P_l$ *is* held constant, where $P_l$ is determined by the initial pressures and the initial volumes, $V_l$.

- Note that if `gamma==0.0`, then $p_l \equiv P_l$.

- On output, the pressure is given by $p_l = P_l / V_l^{\gamma}$, where $V_l$ is the final volume.

- `pressure` is only used in calculation of interface force-balance.

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.24.2.12 ladiabatic** `integer inputlist::ladiabatic = 0`

logical flag

- If `Ladiabatic==0`, the adiabatic constants are determined by the initial pressure and volume.

- If `Ladiabatic==1`, the adiabatic constants are determined by the given input `adiabatic`.

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.24.2.13 adiabatic** `real, dimension(1:mnvol+1) inputlist::adiabatic = 0.0`

adiabatic constants in each volume

- The pressure is *not* held constant, but $p_l V_l^\gamma = P_l \equiv$ `adiabatic` is constant.

- Note that if `gamma==0.0`, then `pressure==adiabatic`.

- `pressure` is only used in calculation of interface force-balance.

Referenced by dfp200(), evaluate_dbb(), sphdf5::hdfint(), lforce(), sphdf5::mirror_input_to_outfile(), allglobal←
::readin(), allglobal::wrtend(), and xspech().

**8.24.2.14 pl** `integer, dimension(0:mnvol) inputlist::pl = 0`

"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (inside) interface rotational-transform is defined by `iota` .

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.15 ql** `integer, dimension(0:mnvol) inputlist::ql = 0`

"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (inside) interface rotational-transform is defined by `iota` .

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.16 pr** `integer, dimension(0:mnvol) inputlist::pr = 0`

"inside" interface rotational-transform is $t = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (inside) interface rotational-transform is defined by `iota` .

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.17 qr** `integer, dimension(0:mnvol) inputlist::qr = 0`

"inside" interface rotational-transform is $t = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (inside) interface rotational-transform is defined by `iota` .

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.18 iota** `real, dimension(0:mnvol) inputlist::iota = 0.0`

rotational-transform, $t$, on inner side of each interface

- only relevant if illogical input for `ql` and `qr` are provided

Referenced by sphdf5::mirror_input_to_outfile(), mp00ac(), pp00aa(), preset(), allglobal::readin(), and allglobal←
::wrtend().

**8.24.2.19 lp** `integer, dimension(0:mnvol) inputlist::lp = 0`

"outer" interface rotational-transform is $t = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (outer) interface rotational-transform is defined by `oita` .

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.20 lq** `integer, dimension(0:mnvol) inputlist::lq = 0`

"outer" interface rotational-transform is $t = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (outer) interface rotational-transform is defined by `oita` .

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.21 rp** `integer, dimension(0:mnvol) inputlist::rp = 0`

"outer" interface rotational-transform is $ι = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (outer) interface rotational-transform is defined by `oita`.

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.22 rq** `integer, dimension(0:mnvol) inputlist::rq = 0`

"outer" interface rotational-transform is $ι = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.

If both $q_l = 0$ *and* $q_r = 0$, then the (outer) interface rotational-transform is defined by `oita`.

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.23 oita** `real, dimension(0:mnvol) inputlist::oita = 0.0`

rotational-transform, $ι$, on outer side of each interface

- only relevant if illogical input for `ql` and `qr` are provided

Referenced by sphdf5::mirror_input_to_outfile(), mp00ac(), pp00aa(), preset(), allglobal::readin(), and allglobal←↩::wrtend().

**8.24.2.24 mupftol** `real inputlist::mupftol = 1.0e-14`

accuracy to which $\mu$ and $\Delta\psi_p$ are required

- only relevant if constraints on transform, enclosed currents etc. are to be satisfied iteratively, see `Lconstraint`

Referenced by dforce(), evaluate_dmupfdx(), ma02aa(), sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.25 mupfits** `integer inputlist::mupfits = 8`

an upper limit on the transform/helicity constraint iterations;

- only relevant if constraints on transform, enclosed currents etc. are to be satisfied iteratively, see `Lconstraint`
- constraint: `mupfits` $> 0$

Referenced by ma02aa(), sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

**8.24.2.26 rpol** `real inputlist::rpol = 1.0`

poloidal extent of slab (effective radius)

- only relevant if `Igeometry==1`
- poloidal size is $L = 2\pi * \texttt{rpol}$

Referenced by coords(), sphdf5::mirror_input_to_outfile(), allglobal::readin(), sphdf5::write_grid(), and allglobal$\hookleftarrow$
::wrtend().

**8.24.2.27 rtor** `real inputlist::rtor = 1.0`

toroidal extent of slab (effective radius)

- only relevant if `Igeometry==1`
- toroidal size is $L = 2\pi * \texttt{rtor}$

Referenced by coords(), sphdf5::mirror_input_to_outfile(), allglobal::readin(), sphdf5::write_grid(), and allglobal$\hookleftarrow$
::wrtend().

## 8.25 numericlist

The namelist `numericlist` controls internal resolution parameters that the user rarely needs to consider.

Collaboration diagram for numericlist:



### Variables

- integer inputlist::linitialize = 0

  *Used to initialize geometry using a regularization / extrapolation method.*
- integer inputlist::lautoinitbn = 1

  *Used to initialize $B_{ns}$ using an initial fixed-boundary calculation.*
- integer inputlist::lzerovac = 0

  *Used to adjust vacuum field to cancel plasma field on computational boundary.*
- integer inputlist::ndiscrete = 2

  *resolution of the real space grid on which fast Fourier transforms are performed is given by `Ndiscrete*Mpol*4`*
- integer inputlist::nquad = -1

  *Resolution of the Gaussian quadrature.*
- integer inputlist::impol = -4

  *Fourier resolution of straight-fieldline angle on interfaces.*
- integer inputlist::intor = -4

  *Fourier resolution of straight-fieldline angle on interfaces;.*
- integer inputlist::lsparse = 0

  *controls method used to solve for rotational-transform on interfaces*
- integer inputlist::lsvdiota = 0

  *controls method used to solve for rotational-transform on interfaces; only relevant if `Lsparse = 0`*
- integer inputlist::imethod = 3

  *controls iterative solution to sparse matrix arising in real-space transformation to the straight-fieldline angle; only relevant if `Lsparse.eq.2;`*
- integer inputlist::iorder = 2

  *controls real-space grid resolution for constructing the straight-fieldline angle; only relevant if `Lsparse>0`*
- integer inputlist::iprecon = 0

  *controls iterative solution to sparse matrix arising in real-space transformation to the straight-fieldline angle; only relevant if `Lsparse.eq.2;`*
- real inputlist::iotatol = -1.0

  *tolerance required for iterative construction of straight-fieldline angle; only relevant if `Lsparse.ge.2`*
- integer inputlist::lextrap = 0

  *geometry of innermost interface is defined by extrapolation*
- integer inputlist::mregular = -1

  *maximum regularization factor*
- integer inputlist::lrzaxis = 1

  *controls the guess of geometry axis in the innermost volume or initialization of interfaces*
- integer inputlist::ntoraxis = 3

  *the number of $n$ harmonics used in the Jacobian $m = 1$ harmonic elimination method; only relevant if `Lrzaxis.↩ ge.1.`*

### 8.25.1 Detailed Description

The namelist `numericlist` controls internal resolution parameters that the user rarely needs to consider.

### 8.25.2 Variable Documentation

#### 8.25.2.1 linitialize `integer inputlist::linitialize = 0`

Used to initialize geometry using a regularization / extrapolation method.

- if `Linitialize` $= -I$, where $I$ is a positive integer, the geometry of the $i = 1, N_V - I$ surfaces constructed by an extrapolation

- if `Linitialize` = 0, the geometry of the interior surfaces is provided after the namelists in the input file

- if `Linitialize` = 1, the interior surfaces will be intialized as $R_{l,m,n} = R_{N,m,n} \psi_{t,l}^{m/2}$, where $R_{N,m,n}$ is the plasma boundary and $\psi_{t,l}$ is the given toroidal flux enclosed by the $l$-th interface, normalized to the total enclosed toroidal flux; a similar extrapolation is used for $Z_{l,m,n}$

- Note that the Fourier harmonics of the boundary is *always* given by the `Rbc` and `Zbs` given in `physicslist`.

- if `Linitialize` = 2, the interior surfaces *and the plasma boundary* will be intialized as $R_{l,m,n} = R_{W,m,n} \psi_{t,l}^{m/2}$, where $R_{W,m,n}$ is the computational boundary and $\psi_{t,l}$ is the given toroidal flux enclosed by the $l$-th interface, normalized to the total enclosed toroidal flux; a similar extrapolation is used for $Z_{l,m,n}$

- Note that, for free-boundary calculations, the Fourier harmonics of the computational boundary are *always* given by the `Rwc` and `Zws` given in `physicslist`.

- if `Linitialize` = 1, 2, it is not required to provide the geometry of the interfaces after the namelists

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

#### 8.25.2.2 lautoinitbn `integer inputlist::lautoinitbn = 1`

Used to initialize $B_{ns}$ using an initial fixed-boundary calculation.

- only relevant if `Lfreebound` = 1

- user-supplied `Bns` will only be considered if `LautoinitBn` = 0

Referenced by allglobal::readin(), allglobal::wrtend(), and xspech().

**8.25.2.3  lzerovac**  `integer inputlist::lzerovac = 0`

Used to adjust vacuum field to cancel plasma field on computational boundary.

- only relevant if `Lfreebound` = 1

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.25.2.4  ndiscrete**  `integer inputlist::ndiscrete = 2`

resolution of the real space grid on which fast Fourier transforms are performed is given by `Ndiscrete*Mpol*4`

- constraint `Ndiscrete>0`

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.25.2.5  nquad**  `integer inputlist::nquad = -1`

Resolution of the Gaussian quadrature.

- The resolution of the Gaussian quadrature, $\int f(s)ds = \sum_k \omega_k f(s_k)$, in each volume is given by `Iquad` $_v$,
- `Iquad` $_v$ is set in [preset()](#)

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.25.2.6  impol**  `integer inputlist::impol = -4`

Fourier resolution of straight-fieldline angle on interfaces.

- the rotational-transform on the interfaces is determined by a transformation to the straight-fieldline angle, with poloidal resolution given by `iMpol`
- if `iMpol<=0`, then `iMpol` = Mpol - iMpol

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

**8.25.2.7  intor**  `integer inputlist::intor = -4`

Fourier resolution of straight-fieldline angle on interfaces;.

- the rotational-transform on the interfaces is determined by a transformation to the straight-fieldline angle, with toroidal resolution given by `iNtor`

- if `iNtor<=0` then `iNtor` = Ntor - iNtor

- if `Ntor==0`, then the toroidal resolution of the angle transformation is set `lNtor` = 0

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

**8.25.2.8  lsparse**  `integer inputlist::lsparse = 0`

controls method used to solve for rotational-transform on interfaces

- if `Lsparse` = 0, the transformation to the straight-fieldline angle is computed in Fourier space using a dense matrix solver, `F04AAF`

- if `Lsparse` = 1, the transformation to the straight-fieldline angle is computed in real space using a dense matrix solver, `F04ATF`

- if `Lsparse` = 2, the transformation to the straight-fieldline angle is computed in real space using a sparse matrix solver, `F11DEF`

- if `Lsparse` = 3, the different methods for constructing the straight-fieldline angle are compared

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), tr00ab(), and allglobal::wrtend().

**8.25.2.9  lsvdiota**  `integer inputlist::lsvdiota = 0`

controls method used to solve for rotational-transform on interfaces; only relevant if `Lsparse` = 0

- if `Lsvdiota` = 0, use standard linear solver to construct straight fieldline angle transformation

- if `Lsvdiota` = 1, use SVD method to compute rotational-transform

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), tr00ab(), and allglobal::wrtend().

**8.25.2.10 imethod** `integer inputlist::imethod = 3`

controls iterative solution to sparse matrix arising in real-space transformation to the straight-fieldline angle; only relevant if `Lsparse.eq.2`;

**See also**

> [tr00ab()](#) for details
>
> - if `imethod` = 1, the method is `RGMRES`
> - if `imethod` = 2, the method is `CGS`
> - if `imethod` = 3, the method is `BICGSTAB`

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), tr00ab(), and allglobal::wrtend().

**8.25.2.11 iorder** `integer inputlist::iorder = 2`

controls real-space grid resolution for constructing the straight-fieldline angle; only relevant if `Lsparse`>0

determines order of finite-difference approximation to the derivatives

- if `iorder` = 2,
- if `iorder` = 4,
- if `iorder` = 6,

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), tr00ab(), and allglobal::wrtend().

**8.25.2.12 iprecon** `integer inputlist::iprecon = 0`

controls iterative solution to sparse matrix arising in real-space transformation to the straight-fieldline angle; only relevant if `Lsparse.eq.2`;

**See also**

> [tr00ab()](#) for details
>
> - if `iprecon` = 0, the preconditioner is 'N'
> - if `iprecon` = 1, the preconditioner is 'J'
> - if `iprecon` = 2, the preconditioner is 'S'

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), tr00ab(), and allglobal::wrtend().

**8.25.2.13  mregular** `integer inputlist::mregular = -1`

maximum regularization factor

- if `Mregular.ge.2`, then `regumm`$_i$ = `Mregular` $/2$ where `m`$_i$ > `Mregular`

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

**8.25.2.14  lrzaxis** `integer inputlist::lrzaxis = 1`

controls the guess of geometry axis in the innermost volume or initialization of interfaces

- if `iprecon` = 1, the centroid is used

- if `iprecon` = 2, the Jacobian $m = 1$ harmonic elimination method is used

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), rzaxis(), and allglobal::wrtend().

## 8.26 locallist

The namelist `locallist` controls the construction of the Beltrami fields in each volume.

Collaboration diagram for locallist:



**Variables**

- integer inputlist::lbeltrami = 4

  *Control flag for solution of Beltrami equation.*
- integer inputlist::linitgues = 1

  *controls how initial guess for Beltrami field is constructed*
- integer inputlist::lposdef = 0

  *redundant;*
- real inputlist::maxrndgues = 1.0

  *the maximum random number of the Beltrami field if* `Linitgues = 3`
- integer inputlist::lmatsolver = 3

  *1 for LU factorization, 2 for GMRES, 3 for GMRES matrix-free*
- integer inputlist::nitergmres = 200

  *number of max iteration for GMRES*
- real inputlist::epsgmres = 1e-14

  *the precision of GMRES*
- integer inputlist::lgmresprec = 1

  *type of preconditioner for GMRES, 1 for ILU sparse matrix*
- real inputlist::epsilu = 1e-12

  *the precision of incomplete LU factorization for preconditioning*

### 8.26.1 Detailed Description

The namelist `locallist` controls the construction of the Beltrami fields in each volume.

The transformation to straight-fieldline coordinates is singular when the rotational-transform of the interfaces is rational; however, the rotational-transform is still well defined.

### 8.26.2 Variable Documentation

### 8.26.2.1   lbeltrami   `integer inputlist::lbeltrami = 4`

Control flag for solution of Beltrami equation.

- if `LBeltrami` = 1,3,5 or 7, (SQP) then the Beltrami field in each volume is constructed by minimizing the magnetic energy with the constraint of fixed helicity; this is achieved by using sequential quadratic programming as provided by `E04UFF`. This approach has the benefit (in theory) of robustly constructing minimum energy solutions when multiple, i.e. bifurcated, solutions exist.

- if `LBeltrami` = 2,3,6 or 7, (Newton) then the Beltrami fields are constructed by employing a standard Newton method for locating an extremum of $F \equiv \int B^2 dv - \mu(\int \mathbf{A} \cdot \mathbf{B} dv - \mathcal{K})$, where $\mu$ is treated as an independent degree of freedom similar to the parameters describing the vector potential and $\mathcal{K}$ is the required value of the helicity; this is the standard Lagrange multipler approach for locating the constrained minimum; this method cannot distinguish saddle-type extrema from minima, and which solution that will be obtained depends on the initial guess;

- if `LBeltrami` = 4,5,6 or 7, (linear) it is assumed that the Beltrami fields are parameterized by $\mu$; in this case, it is only required to solve $\nabla \times \mathbf{B} = \mu \mathbf{B}$ which reduces to a system of linear equations; $\mu$ may or may not be adjusted iteratively, depending on `Lconstraint`, to satisfy either rotational-transform or helicity constraints;

- for flexibility and comparison, each of the above methods can be employed; for example:

    - if `LBeltrami` = 1, only the SQP method will be employed;
    - if `LBeltrami` = 2, only the Newton method will be employed;
    - if `LBeltrami` = 4, only the linear method will be employed;
    - if `LBeltrami` = 3, the SQP and the Newton method are used;
    - if `LBeltrami` = 5, the SQP and the linear method are used;
    - if `LBeltrami` = 6, the Newton and the linear method are used;
    - if `LBeltrami` = 7, all three methods will be employed;

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

### 8.26.2.2   linitgues   `integer inputlist::linitgues = 1`

controls how initial guess for Beltrami field is constructed

- only relevant for routines that require an initial guess for the Beltrami fields, such as the SQP and Newton methods, or the sparse linear solver;

- if `Linitgues` = 0, the initial guess for the Beltrami field is trivial

- if `Linitgues` = 1, the initial guess for the Beltrami field is an integrable approximation

- if `Linitgues` = 2, the initial guess for the Beltrami field is read from file

- if `Linitgues` = 3, the initial guess for the Beltrami field will be randomized with the maximum `maxrndgues`

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

## 8.27 globallist

The namelist `globallist` controls the search for global force-balance.

Collaboration diagram for globallist:



**Variables**

- integer inputlist::lfindzero = 0

  *use Newton methods to find zero of force-balance, which is computed by dforce()*

- real inputlist::escale = 0.0

  *controls the weight factor, `BBweight`, in the force-imbalance harmonics*

- real inputlist::opsilon = 1.0

  *weighting of force-imbalance*

- real inputlist::pcondense = 2.0

  *spectral condensation parameter*

- real inputlist::epsilon = 0.0

  *weighting of spectral-width constraint*

- real inputlist::wpoloidal = 1.0

  *"star-like" poloidal angle constraint radial exponential factor used in preset() to construct `sweight`*

- real inputlist::upsilon = 1.0

  *weighting of "star-like" poloidal angle constraint used in preset() to construct `sweight`*

- real inputlist::forcetol = 1.0e-10

  *required tolerance in force-balance error; only used as an initial check*

- real inputlist::c05xmax = 1.0e-06

  *required tolerance in position, $\mathbf{x} \equiv \{R_{i,v}, Z_{i,v}\}$*

- real inputlist::c05xtol = 1.0e-12

  *required tolerance in position, $\mathbf{x} \equiv \{R_{i,v}, Z_{i,v}\}$*

- real inputlist::c05factor = 1.0e-02

  *used to control initial step size in `C05NDF` and `C05PDF`*

- logical inputlist::lreadgf = .true.

  *read $\nabla_{\mathbf{x}}\mathbf{F}$ from file `ext.GF`*

- integer inputlist::mfreeits = 0

  *maximum allowed free-boundary iterations*

- real inputlist::bnstol = 1.0e-06

  *redundant;*

- real inputlist::bnsblend = 0.666

  *redundant;*

- real inputlist::gbntol = 1.0e-06

  *required tolerance in free-boundary iterations*

- real inputlist::gbnbld = 0.666

    *normal blend*
- real inputlist::vcasingeps = 1.e-12

    *regularization of Biot-Savart; see bnorml(), casing()*
- real inputlist::vcasingtol = 1.e-08

    *accuracy on virtual casing integral; see bnorml(), casing()*
- integer inputlist::vcasingits = 8

    *minimum number of calls to adaptive virtual casing routine; see casing()*
- integer inputlist::vcasingper = 1

    *periods of integragion in adaptive virtual casing routine; see casing()*
- integer inputlist::mcasingcal = 8

    *minimum number of calls to adaptive virtual casing routine; see casing(); redundant;*

### 8.27.1 Detailed Description

The namelist `globallist` controls the search for global force-balance.

Comments:

- The "force" vector, $\mathbf{F}$, which is constructed in dforce(), is a combination of pressure-imbalance Fourier harmonics,

$$F_{i,v} \equiv [[p + B^2/2]]_{i,v} \times \exp\left[-\texttt{escale}(m_i^2 + n_i^2)\right] \times \texttt{opsilon}, \tag{274}$$

    and spectral-condensation constraints, $I_{i,v}$, and the "star-like" angle constraints, $S_{i,v,}$, (see lforce() for details)

$$F_{i,v} \equiv \texttt{epsilon} \times I_{i,v} + \texttt{upsilon} \times \left(\psi_v^\omega S_{i,v,1} - \psi_{v+1}^\omega S_{i,v+1,0}\right), \tag{275}$$

    where $\psi_v \equiv$ normalized toroidal flux, `tflux`, and $\omega \equiv \texttt{wpoloidal}$.

### 8.27.2 Variable Documentation

#### 8.27.2.1 lfindzero    `integer inputlist::lfindzero = 0`

use Newton methods to find zero of force-balance, which is computed by dforce()

- if `Lfindzero = 0`, then dforce() is called once to compute the Beltrami fields consistent with the given geometry and constraints

- if `Lfindzero = 1`, then call `C05NDF` (uses function values only), which iteratively calls dforce()

- if `Lfindzero = 2`, then call `C05PDF` (uses derivative information), which iteratively calls dforce()

Referenced by brcast(), dfp200(), hesian(), sphdf5::mirror_input_to_outfile(), packxi(), preset(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.27.2.2 escale** `real inputlist::escale = 0.0`

controls the weight factor, `BBweight`, in the force-imbalance harmonics

- `BBweight(i)` $\equiv$ `opsilon` $\times \exp\left[-\text{escale} \times (m_i^2 + n_i^2)\right]$
- defined in preset() ; used in dforce()
- also see Eqn. (274)

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.27.2.3 opsilon** `real inputlist::opsilon = 1.0`

weighting of force-imbalance

- used in dforce(); also see Eqn. (274)

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.27.2.4 pcondense** `real inputlist::pcondense = 2.0`

spectral condensation parameter

- used in preset() to define `mmpp(i)` $\equiv m_i^p$, where $p \equiv$ `pcondense`
- the angle freedom is exploited to minimize `epsilon` $\sum_i m_i^p (R_i^2 + Z_i^2)$ with respect to tangential variations in the interface geometry
- also see Eqn. (275)

Referenced by sphdf5::mirror_input_to_outfile(), preset(), allglobal::readin(), and allglobal::wrtend().

**8.27.2.5 epsilon** `real inputlist::epsilon = 0.0`

weighting of spectral-width constraint

- used in dforce(); also see Eqn. (275)

Referenced by dforce(), dfp200(), evaluate_dbb(), sphdf5::mirror_input_to_outfile(), pc00ab(), allglobal::readin(), and allglobal::wrtend().

### 8.27.2.6  forcetol  `real inputlist::forcetol = 1.0e-10`

required tolerance in force-balance error; only used as an initial check

- if the initially supplied interfaces are consistent with force-balance to within `forcetol` then the geometry of the interfaces is not altered
- if not, then the geometry of the interfaces is changed in order to bring the configuration into force balance so that the geometry of interfaces is within `c05xtol`, defined below, of the true solution
- to force execution of either `C05NDF` or `C05PDF`, regardless of the initial force imbalance, set `forcetol` $<$ 0

Referenced by sphdf5::mirror_input_to_outfile(), pc00aa(), pc00ab(), allglobal::readin(), and allglobal::wrtend().

### 8.27.2.7  c05xtol  `real inputlist::c05xtol = 1.0e-12`

required tolerance in position, $\mathbf{x} \equiv \{R_{i,v}, Z_{i,v}\}$

- used by both `C05NDF` and `C05PDF`; see the NAG documents for further details on how the error is defined
- constraint `c05xtol` $>$ 0.0

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

### 8.27.2.8  c05factor  `real inputlist::c05factor = 1.0e-02`

used to control initial step size in `C05NDF` and `C05PDF`

- constraint `c05factor` $>$ 0.0
- only relevant if `Lfindzero` $>$ 0

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

### 8.27.2.9  lreadgf  `logical inputlist::lreadgf = .true.`

read $\nabla_{\mathbf{x}}\mathbf{F}$ from file `ext.GF`

- only used if `Lfindzero` = 2
- only used in newton()

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), and allglobal::wrtend().

**8.27.2.10 mfreeits** `integer inputlist::mfreeits = 0`

maximum allowed free-boundary iterations

- only used if `Lfreebound` = 1

- only used in xspech()

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.27.2.11 gbntol** `real inputlist::gbntol = 1.0e-06`

required tolerance in free-boundary iterations

- only used if `Lfreebound` = 1

- only used in xspech()

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), allglobal::wrtend(), and xspech().

**8.27.2.12 gbnbld** `real inputlist::gbnbld = 0.666`

normal blend

- The "new" magnetic field at the computational boundary produced by the plasma currents is updated using a Picard scheme:

$$(\mathbf{B} \cdot \mathbf{n})^{j+1} = \texttt{gBnbld} \times (\mathbf{B} \cdot \mathbf{n})^j + (1 - \texttt{gBnbld}) \times (\mathbf{B} \cdot \mathbf{n})^*, \tag{276}$$

where $j$ labels free-boundary iterations, and $(\mathbf{B} \cdot \mathbf{n})^*$ is computed by virtual casing.

- only used if `Lfreebound` = 1

- only used in xspech()

Referenced by sphdf5::mirror_input_to_outfile(), allglobal::readin(), allglobal::wrtend(), and xspech().

## 8.28 diagnosticslist

The namelist `diagnosticslist` controls post-processor diagnostics, such as Poincaré plot resolution, etc.

Collaboration diagram for diagnosticslist:

```
┌─────────────────────┐         ┌─────────────────────┐
│ Input namelists and │◀────────│   diagnosticslist   │
│   global variables  │         │                     │
└─────────────────────┘         └─────────────────────┘
```
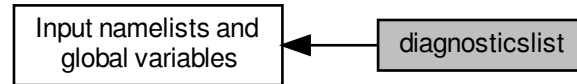
**Variables**

- real inputlist::odetol = 1.0e-07

  *o.d.e. integration tolerance for all field line tracing routines*
- real inputlist::absreq = 1.0e-08

  *redundant*
- real inputlist::relreq = 1.0e-08

  *redundant*
- real inputlist::absacc = 1.0e-04

  *redundant*
- real inputlist::epsr = 1.0e-08

  *redundant*
- integer inputlist::nppts = 0

  *number of toroidal transits used (per trajectory) in following field lines for constructing Poincaré plots; if `nPpts<1`, no Poincaré plot is constructed;*
- real inputlist::ppts = 0.0

  *stands for Poincare plot theta start. Chose at which angle (normalized over $\pi$) the Poincare field-line tracing start.*
- integer, dimension(1:mnvol+1) inputlist::nptrj = -1

  *number of trajectories in each annulus to be followed in constructing Poincaré plot*
- logical inputlist::lhevalues = .false.

  *to compute eigenvalues of $\nabla \mathbf{F}$*
- logical inputlist::lhevectors = .false.

  *to compute eigenvectors (and also eigenvalues) of $\nabla \mathbf{F}$*
- logical inputlist::lhmatrix = .false.

  *to compute and write to file the elements of $\nabla \mathbf{F}$*
- integer inputlist::lperturbed = 0

  *to compute linear, perturbed equilibrium*
- integer inputlist::dpp = -1

  *perturbed harmonic*
- integer inputlist::dqq = -1

  *perturbed harmonic*
- integer inputlist::lerrortype = 0

  *the type of error output for Lcheck=1*
- integer inputlist::ngrid = -1

  *the number of points to output in the grid, -1 for Lrad(vvol)*

- real inputlist::drz = 1E-5

    *difference in geometry for finite difference estimate (debug only)*
- integer inputlist::lcheck = 0

    *implement various checks*
- logical inputlist::ltiming = .false.

    *to check timing*
- real inputlist::fudge = 1.0e-00

    *redundant*
- real inputlist::scaling = 1.0e-00

    *redundant*

### 8.28.1 Detailed Description

The namelist `diagnosticslist` controls post-processor diagnostics, such as Poincaré plot resolution, etc.

### 8.28.2 Variable Documentation

#### 8.28.2.1 nptrj `integer, dimension(1:mnvol+1) inputlist::nptrj = -1`

number of trajectories in each annulus to be followed in constructing Poincaré plot

- if `nPtrj(l) < 0`, then `nPtrj(l)` = Ni(l), where `Ni(l)` is the grid resolution used to construct the Beltrami field in volume $l$

Referenced by sphdf5::mirror_input_to_outfile(), pp00aa(), allglobal::readin(), allglobal::wrtend(), and xspech().

#### 8.28.2.2 lcheck `integer inputlist::lcheck = 0`

implement various checks

- if `Lcheck` = 0, no additional check on the calculation is performed

- if `Lcheck` = 1, the error in the current, i.e. $\nabla \times \mathbf{B} - \mu \mathbf{B}$ is computed as a post-diagnostic

- if `Lcheck` = 2, the analytic derivatives of the interface transform w.r.t. the helicity multiplier, $\mu$, and the enclosed poloidal flux, $\Delta \psi_p$, are compared to a finite-difference estimate

    - only if `Lconstraint==1`
    - only for `dspec` executable, i.e. must compile with `DFLAGS` = "-D DEBUG"

- if `Lcheck` = 3, the analytic derivatives of the volume w.r.t. interface Fourier harmonic is compared to a finite-difference estimate

    - must set `Lfindzero` $= 2$
    - set `forcetol` sufficiently small and set `LreadGF` = F, so that the matrix of second derivatives is calculated

– only for `dspec` executable, i.e. must compile with `DFLAGS` = "-D DEBUG"

- if `Lcheck` = 4, the analytic calculation of the derivatives of the magnetic field, $B^2$, at the interfaces is compared to a finite-difference estimate

    – must set $\mathtt{Lfindzero} = 2$

    – set `forcetol` sufficiently small

    – set `LreadGF=F`

    – only for `dspec` executable, i.e. must compile with `DFLAGS` = "-D DEBUG"

- if `Lcheck` = 5, the analytic calculation of the matrix of the derivatives of the force imbalance is compared to a finite-difference estimate

- if `Lcheck` = 6, the virtual casing calculation is compared to `xdiagno` (Lazerson 2013 [7])

    – the input file for `xdiagno` is written by bnorml()

    – this provides the Cartesian coordinates on the computational boundary where the virtual casing routine casing() computes the magnetic field, with the values of the magnetic field being written to the screen for comparison

    – must set $\mathtt{Freebound=1, Lfindzero>0, mfreeits!=0}$

    – `xdiagno` must be executed manually

Referenced by bnorml(), dforce(), dfp200(), evaluate_dbb(), evaluate_dmupfdx(), sphdf5::hdfint(), hesian(), lbpol(), lforce(), ma02aa(), sphdf5::mirror_input_to_outfile(), allglobal::readin(), rzaxis(), allglobal::wrtend(), and xspech().

## 8.29  screenlist

The namelist `screenlist` controls screen output. Every subroutine, e.g. `xy00aa.h`, has its own write flag, `Wxy00aa`.

Collaboration diagram for screenlist:

```
┌─────────────────────┐        ┌─────────────┐
│ Input namelists and │ ◄───── │  screenlist │
│   global variables  │        └─────────────┘
└─────────────────────┘
```

**Variables**

- logical inputlist::wbuild_vector_potential = .false.
- logical inputlist::wreadin = .false.

  *write screen output of readin()*
- logical inputlist::wwrtend = .false.

  *write screen output of wrtend()*
- logical inputlist::wmacros = .false.

  *write screen output from expanded macros*

### 8.29.1  Detailed Description

The namelist `screenlist` controls screen output. Every subroutine, e.g. `xy00aa.h`, has its own write flag, `Wxy00aa`.

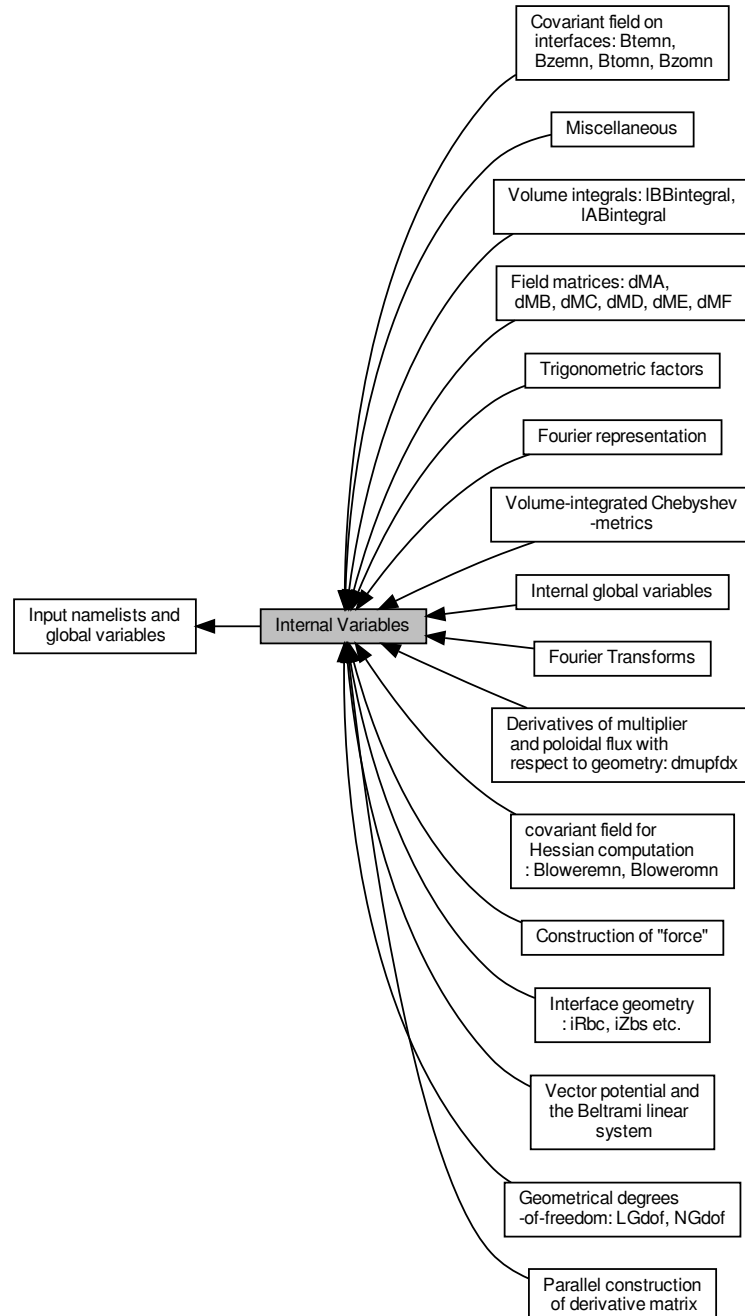### 8.29.2  Variable Documentation

#### 8.29.2.1  wbuild_vector_potential  `logical inputlist::wbuild_vector_potential = .false.`

**Todo**  : what is this?

## 8.30 Internal Variables

Collaboration diagram for Internal Variables:



**Modules**

- Fourier representation
- Interface geometry: iRbc, iZbs etc.

> *The Fourier harmonics of the interfaces are contained in* `iRbc(1:mn,0:Mvol)` *and* `iZbs(1:mn,0:Mvol)`, *where* `iRbc(l,j)`, `iZbs(l,j)` *contains the Fourier harmonics,* $R_j$, $Z_j$, *of the* $l$-*th interface.*

- Fourier Transforms

  > *The coordinate geometry and fields are mapped to/from Fourier space and real space using FFTW3. The resolution of the real space grid is given by* `Nt=Ndiscrete*4*Mpol` *and* `Nz=Ndiscrete*4*Ntor`.

- Volume-integrated Chebyshev-metrics

  > *These are allocated in* dforce()*, defined in* ma00aa()*, and are used in* matrix() *to construct the matrices.*

- Vector potential and the Beltrami linear system
- Field matrices: dMA, dMB, dMC, dMD, dME, dMF
- Construction of "force"

  > *The force vector is comprised of* `Bomn` *and* `Iomn`.

- Covariant field on interfaces: Btemn, Bzemn, Btomn, Bzomn

  > *The covariant field.*

- covariant field for Hessian computation: Bloweremn, Bloweromn
- Geometrical degrees-of-freedom: LGdof, NGdof

  > *The geometrical degrees-of-freedom.*

- Parallel construction of derivative matrix
- Derivatives of multiplier and poloidal flux with respect to geometry: dmupfdx
- Trigonometric factors
- Volume integrals: IBBintegral, IABintegral
- Internal global variables

  > *internal global variables; internal logical variables; default values are provided here; these may be changed according to input values*

- Miscellaneous

  > *The following are miscellaneous flags required for the virtual casing field, external (vacuum) field integration, ...*

## Data Types

- type allglobal::derivative

  $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ *(?)* *More...*

## Variables

- logical allglobal::derivative::l

  *what is this?*

- integer allglobal::derivative::vol

  *Used in* coords()*; required for global constraint force gradient evaluation.*

- integer allglobal::derivative::innout

  *what is this?*

- integer allglobal::derivative::ii

  *what is this?*

- integer allglobal::derivative::irz

  *what is this?*

- integer allglobal::derivative::issym

  *what is this?*

- type(derivative) allglobal::dbdx

  $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ *(?)*

### 8.30.1  Detailed Description

### 8.30.2  Data Type Documentation

#### 8.30.2.1  type allglobal::derivative    $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ (?)

**Class Members**

| logical | l | what is this? |
|---|---|---|
| integer | vol | Used in [coords()](); required for global constraint force gradient evaluation. |
| integer | innout | what is this? |
| integer | ii | what is this? |
| integer | irz | what is this? |
| integer | issym | what is this? |

### 8.30.3  Variable Documentation

**8.30.3.1  l**  `logical allglobal::derivative::l`

what is this?

**8.30.3.2  vol**  `integer allglobal::derivative::vol`

Used in [coords](); required for global constraint force gradient evaluation.

**8.30.3.3  innout**  `integer allglobal::derivative::innout`

what is this?

**8.30.3.4  ii**  `integer allglobal::derivative::ii`

what is this?

**8.30.3.5  irz**  `integer allglobal::derivative::irz`

what is this?

**8.30.3.6  issym**  `integer allglobal::derivative::issym`

what is this?

## 8.31 Fourier representation

Collaboration diagram for Fourier representation:
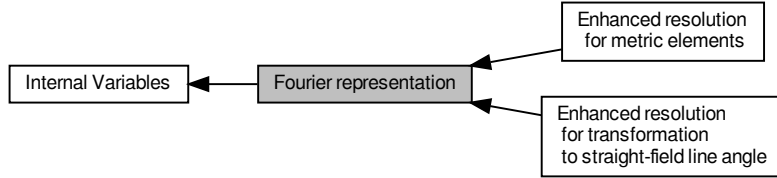


### Modules

- Enhanced resolution for metric elements

  *Enhanced resolution is required for the metric elements, $g_{ij}/\sqrt{g}$, which is given by mne, ime, and ine. The Fourier resolution here is determined by `lMpol=2*Mpol` and `lNtor=2*Ntor`.*
- Enhanced resolution for transformation to straight-field line angle

  *Enhanced resolution is required for the transformation to straight-field line angle on the interfaces, which is given by mns, ims and ins. The Fourier resolution here is determined by `iMpol` and `iNtor`.*

### Variables

- integer allglobal::mn

  *total number of Fourier harmonics for coordinates/fields; calculated from Mpol, Ntor in readin()*
- integer, dimension(:), allocatable allglobal::im

  *poloidal mode numbers for Fourier representation*
- integer, dimension(:), allocatable allglobal::in

  *toroidal mode numbers for Fourier representation*
- real, dimension(:), allocatable allglobal::halfmm

  *I saw this already somewhere...*
- real, dimension(:), allocatable allglobal::regumm

  *I saw this already somewhere...*
- real allglobal::rscale

  *no idea*
- real, dimension(:,:), allocatable allglobal::psifactor

  *no idea*
- real, dimension(:,:), allocatable allglobal::inifactor

  *no idea*
- real, dimension(:), allocatable allglobal::bbweight

  *weight on force-imbalance harmonics; used in dforce()*
- real, dimension(:), allocatable allglobal::mmpp

  *spectral condensation factors*
- integer allglobal::lmpol

  *what is this?*
- integer allglobal::lntor

  *what is this?*

- integer allglobal::smpol

     *what is this?*
- integer allglobal::sntor

     *what is this?*
- real allglobal::xoffset = 1.0

     *used to normalize NAG routines (which ones exacly where?)*

### 8.31.1 Detailed Description

## 8.32 Enhanced resolution for metric elements

Enhanced resolution is required for the metric elements, $g_{ij}/\sqrt{g}$, which is given by mne, ime, and ine. The Fourier resolution here is determined by `lMpol=2*Mpol` and `lNtor=2*Ntor`.

Collaboration diagram for Enhanced resolution for metric elements:
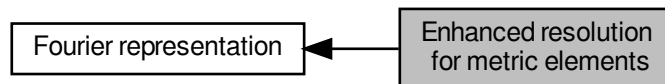


**Variables**

- integer allglobal::mne

  *enhanced resolution for metric elements*
- integer, dimension(:), allocatable allglobal::ime

  *enhanced poloidal mode numbers for metric elements*
- integer, dimension(:), allocatable allglobal::ine

  *enhanced toroidal mode numbers for metric elements*

### 8.32.1 Detailed Description

Enhanced resolution is required for the metric elements, $g_{ij}/\sqrt{g}$, which is given by mne, ime, and ine. The Fourier resolution here is determined by `lMpol=2*Mpol` and `lNtor=2*Ntor`.

## 8.33 Enhanced resolution for transformation to straight-field line angle

Enhanced resolution is required for the transformation to straight-field line angle on the interfaces, which is given by mns, ims and ins. The Fourier resolution here is determined by `iMpol` and `iNtor`.

Collaboration diagram for Enhanced resolution for transformation to straight-field line angle:
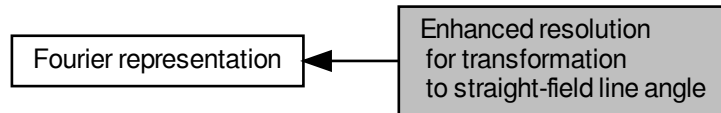


**Variables**

- integer allglobal::mns

  *enhanced resolution for straight field line transformation*
- integer, dimension(:), allocatable allglobal::ims

  *enhanced poloidal mode numbers for straight field line transformation*
- integer, dimension(:), allocatable allglobal::ins

  *enhanced toroidal mode numbers for straight field line transformation*

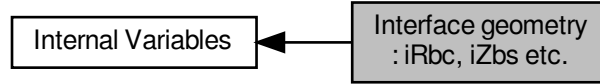### 8.33.1 Detailed Description

Enhanced resolution is required for the transformation to straight-field line angle on the interfaces, which is given by mns, ims and ins. The Fourier resolution here is determined by `iMpol` and `iNtor`.

## 8.34 Interface geometry: iRbc, iZbs etc.

The Fourier harmonics of the interfaces are contained in `iRbc(1:mn,0:Mvol)` and `iZbs(1:mn,0:Mvol)`, where `iRbc(l,j)`, `iZbs(l,j)` contains the Fourier harmonics, $R_j$, $Z_j$, of the $l$-th interface.

Collaboration diagram for Interface geometry: iRbc, iZbs etc.:



**Variables**

- real, dimension(:,:), allocatable allglobal::irbc

  *cosine R harmonics of interface surface geometry; stellarator symmetric*
- real, dimension(:,:), allocatable allglobal::izbs

  *sine Z harmonics of interface surface geometry; stellarator symmetric*
- real, dimension(:,:), allocatable allglobal::irbs

  *sine R harmonics of interface surface geometry; non-stellarator symmetric*
- real, dimension(:,:), allocatable allglobal::izbc

  *cosine Z harmonics of interface surface geometry; non-stellarator symmetric*
- real, dimension(:,:), allocatable allglobal::drbc

  *cosine R harmonics of interface surface geometry; stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable allglobal::dzbs

  *sine Z harmonics of interface surface geometry; stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable allglobal::drbs

  *sine R harmonics of interface surface geometry; non-stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable allglobal::dzbc

  *cosine Z harmonics of interface surface geometry; non-stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable allglobal::irij

  *interface surface geometry; real space*
- real, dimension(:,:), allocatable allglobal::izij

  *interface surface geometry; real space*
- real, dimension(:,:), allocatable allglobal::drij

  *interface surface geometry; real space*
- real, dimension(:,:), allocatable allglobal::dzij

  *interface surface geometry; real space*
- real, dimension(:,:), allocatable allglobal::trij

  *interface surface geometry; real space*
- real, dimension(:,:), allocatable allglobal::tzij

  *interface surface geometry; real space*
- real, dimension(:), allocatable allglobal::ivns

  *sine harmonics of vacuum normal magnetic field on interfaces; stellarator symmetric*
- real, dimension(:), allocatable allglobal::ibns

  *sine harmonics of plasma normal magnetic field on interfaces; stellarator symmetric*

- real, dimension(:), allocatable allglobal::ivnc

  *cosine harmonics of vacuum normal magnetic field on interfaces; non-stellarator symmetric*

- real, dimension(:), allocatable allglobal::ibnc

  *cosine harmonics of plasma normal magnetic field on interfaces; non-stellarator symmetric*

- real, dimension(:), allocatable allglobal::lrbc

  *local workspace*

- real, dimension(:), allocatable allglobal::lzbs

  *local workspace*

- real, dimension(:), allocatable allglobal::lrbs

  *local workspace*

- real, dimension(:), allocatable allglobal::lzbc

  *local workspace*

### 8.34.1 Detailed Description

The Fourier harmonics of the interfaces are contained in `iRbc(1:mn,0:Mvol)` and `iZbs(1:mn,0:Mvol)`, where `iRbc(l,j)`, `iZbs(l,j)` contains the Fourier harmonics, $R_j$, $Z_j$, of the $l$-th interface.

## 8.35 Fourier Transforms

The coordinate geometry and fields are mapped to/from Fourier space and real space using FFTW3. The resolution of the real space grid is given by `Nt=Ndiscrete*4*Mpol` and `Nz=Ndiscrete*4*Ntor`.

Collaboration diagram for Fourier Transforms:

```
┌──────────────────────┐      ┌──────────────────────┐
│  Internal Variables  │◀─────│  Fourier Transforms  │
└──────────────────────┘      └──────────────────────┘
```

**Variables**

- integer allglobal::nt

  *discrete resolution along $\theta$ of grid in real space*
- integer allglobal::nz

  *discrete resolution along $\zeta$ of grid in real space*
- integer allglobal::ntz

  *discrete resolution; Ntz=Nt*Nz shorthand*
- integer allglobal::hnt

  *discrete resolution; Ntz=Nt*Nz shorthand*
- integer allglobal::hnz

  *discrete resolution; Ntz=Nt*Nz shorthand*
- real allglobal::sontz

  *one / sqrt (one*Ntz); shorthand*
- real, dimension(:,:,:), allocatable allglobal::rij

  *real-space grid; R*
- real, dimension(:,:,:), allocatable allglobal::zij

  *real-space grid; Z*
- real, dimension(:,:,:), allocatable allglobal::xij

  *what is this?*
- real, dimension(:,:,:), allocatable allglobal::yij

  *what is this?*
- real, dimension(:,:), allocatable allglobal::sg

  *real-space grid; jacobian and its derivatives*
- real, dimension(:,:,:,:), allocatable allglobal::guvij

  *real-space grid; metric elements*
- real, dimension(:,:,:,:), allocatable allglobal::gvuij

  *real-space grid; metric elements (?); 10 Dec 15;*
- real, dimension(:,:,:,:), allocatable allglobal::guvijsave

  *what is this?*
- integer, dimension(:,:), allocatable allglobal::ki

  *identification of Fourier modes*
- integer, dimension(:,:,:), allocatable allglobal::kijs

  *identification of Fourier modes*

- integer, dimension(:,:,:), allocatable allglobal::kija

    *identification of Fourier modes*
- integer, dimension(:), allocatable allglobal::iotakkii

    *identification of Fourier modes*
- integer, dimension(:,:), allocatable allglobal::iotaksub

    *identification of Fourier modes*
- integer, dimension(:,:), allocatable allglobal::iotakadd

    *identification of Fourier modes*
- integer, dimension(:,:), allocatable allglobal::iotaksgn

    *identification of Fourier modes*
- real, dimension(:), allocatable allglobal::efmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::ofmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::cfmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::sfmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::evmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::odmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::comn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::simn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable allglobal::ijreal

    *what is this ?*
- real, dimension(:), allocatable allglobal::ijimag

    *what is this ?*
- real, dimension(:), allocatable allglobal::jireal

    *what is this ?*
- real, dimension(:), allocatable allglobal::jiimag

    *what is this ?*
- real, dimension(:), allocatable allglobal::jkreal

    *what is this ?*
- real, dimension(:), allocatable allglobal::jkimag

    *what is this ?*
- real, dimension(:), allocatable allglobal::kjreal

    *what is this ?*
- real, dimension(:), allocatable allglobal::kjimag

    *what is this ?*
- real, dimension(:,:,:), allocatable allglobal::bsupumn

    *tangential field on interfaces; $\theta$-component; required for virtual casing construction of field; 11 Oct 12*
- real, dimension(:,:,:), allocatable allglobal::bsupvmn

    *tangential field on interfaces; $\zeta$-component; required for virtual casing construction of field; 11 Oct 12*
- real, dimension(:,:), allocatable allglobal::goomne

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::goomno

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gssmne

*described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gssmno

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gstmne

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gstmno

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gszmne

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gszmno

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gttmne

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gttmno

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gtzmne

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gtzmno

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gzzmne

  *described in* *preset()*

- real, dimension(:,:), allocatable allglobal::gzzmno

  *described in* *preset()*

### 8.35.1 Detailed Description

The coordinate geometry and fields are mapped to/from Fourier space and real space using FFTW3. The resolution of the real space grid is given by `Nt=Ndiscrete*4*Mpol` and `Nz=Ndiscrete*4*Ntor`.

Various workspace arrays are allocated. These include `Rij(1:Ntz,0:3,0:3)` and `Zij(1:Ntz,0:3,0:3)`, which contain the coordinates in real space and their derivatives; `sg(0:3,Ntz)`, which contains the Jacobian and its derivatives; and `guv(0:6,0:3,1:Ntz)`, which contains the metric elements and their derivatives.

## 8.36 Volume-integrated Chebyshev-metrics

These are allocated in [dforce()](#), defined in [ma00aa()](#), and are used in [matrix()](#) to construct the matrices.

Collaboration diagram for Volume-integrated Chebyshev-metrics:

```
┌──────────────────┐     ┌──────────────────────────┐
│ Internal Variables│◄────│ Volume-integrated Chebyshev│
│                  │     │         -metrics          │
└──────────────────┘     └──────────────────────────┘
```

**Variables**

- real, dimension(:,:,:,:,:), allocatable [allglobal::dtoocc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::dtoocs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::dtoosc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::dtooss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::ttsscc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::ttsscs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::ttsssc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::ttssss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdstcc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdstcs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdstsc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdstss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdszcc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdszcs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdszsc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::tdszss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*
- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttcc](#)

*volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttcs](allglobal::ddttcs)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttsc](allglobal::ddttsc)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttss](allglobal::ddttss)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzcc](allglobal::ddtzcc)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzcs](allglobal::ddtzcs)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzsc](allglobal::ddtzsc)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzss](allglobal::ddtzss)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzcc](allglobal::ddzzcc)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzcs](allglobal::ddzzcs)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzsc](allglobal::ddzzsc)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzss](allglobal::ddzzss)

   *volume-integrated Chebychev-metrics; see [matrix()](matrix())*

- real, dimension(:,:), allocatable [allglobal::tsc](allglobal::tsc)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::tss](allglobal::tss)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::dtc](allglobal::dtc)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::dts](allglobal::dts)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::dzc](allglobal::dzc)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::dzs](allglobal::dzs)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::ttc](allglobal::ttc)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::tzc](allglobal::tzc)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::tts](allglobal::tts)

   *what is this?*

- real, dimension(:,:), allocatable [allglobal::tzs](allglobal::tzs)

   *what is this?*

- real, dimension(:), allocatable [allglobal::dtflux](allglobal::dtflux)

   $\delta\psi_{toroidal}$ *in each annulus*

- real, dimension(:), allocatable [allglobal::dpflux](allglobal::dpflux)

   $\delta\psi_{poloidal}$ *in each annulus*

- real, dimension(:), allocatable [allglobal::sweight](allglobal::sweight)

   *minimum poloidal length constraint weight*

### 8.36.1 Detailed Description

These are allocated in [dforce()](dforce()), defined in [ma00aa()](ma00aa()), and are used in [matrix()](matrix()) to construct the matrices.

## 8.37    Vector potential and the Beltrami linear system

Collaboration diagram for Vector potential and the Beltrami linear system:
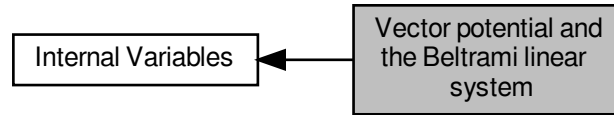


**Variables**

- integer, dimension(:), allocatable allglobal::nadof

  *degrees of freedom in Beltrami fields in each annulus*
- integer, dimension(:), allocatable allglobal::nfielddof

  *degrees of freedom in Beltrami fields in each annulus, field only, no Lagrange multipliers*
- type(subgrid), dimension(:,:,:), allocatable allglobal::ate

  *magnetic vector potential cosine Fourier harmonics; stellarator-symmetric*
- type(subgrid), dimension(:,:,:), allocatable allglobal::aze

  *magnetic vector potential cosine Fourier harmonics; stellarator-symmetric*
- type(subgrid), dimension(:,:,:), allocatable allglobal::ato

  *magnetic vector potential sine Fourier harmonics; non-stellarator-symmetric*
- type(subgrid), dimension(:,:,:), allocatable allglobal::azo

  *magnetic vector potential sine Fourier harmonics; non-stellarator-symmetric*
- integer, dimension(:,:), allocatable allglobal::lma

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmb

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmc

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmd

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lme

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmf

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmg

  *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmh

  *Lagrange multipliers (?)*
- real, dimension(:,:), allocatable allglobal::lmavalue

  *what is this?*
- real, dimension(:,:), allocatable allglobal::lmbvalue

  *what is this?*
- real, dimension(:,:), allocatable allglobal::lmcvalue

*what is this?*

- real, dimension(:,:), allocatable allglobal::lmdvalue

  *what is this?*

- real, dimension(:,:), allocatable allglobal::lmevalue

  *what is this?*

- real, dimension(:,:), allocatable allglobal::lmfvalue

  *what is this?*

- real, dimension(:,:), allocatable allglobal::lmgvalue

  *what is this?*

- real, dimension(:,:), allocatable allglobal::lmhvalue

  *what is this?*

- integer, dimension(:,:), allocatable allglobal::fso

  *what is this?*

- integer, dimension(:,:), allocatable allglobal::fse

  *what is this?*

- logical allglobal::lcoordinatesingularity

  *set by* `LREGION` *macro; true if inside the innermost volume*

- logical allglobal::lplasmaregion

  *set by* `LREGION` *macro; true if inside the plasma region*

- logical allglobal::lvacuumregion

  *set by* `LREGION` *macro; true if inside the vacuum region*

- logical allglobal::lsavedguvij

  *flag used in matrix free*

- logical allglobal::localconstraint

  *what is this?*

### 8.37.1 Detailed Description

- In each volume, the total degrees of freedom in the Beltrami linear system is `NAdof(1:Nvol)`. This depends on `Mpol`, `Ntor` and `Lrad(vvol)`.

- The covariant components of the vector potential are written as

$$A_\theta \;=\; \sum_i \sum_{l=0}^{L} A_{\theta,e,i,l}\, T_l(s)\cos\alpha_i + \sum_i \sum_{l=0}^{L} A_{\theta,o,i,l}\, T_l(s)\sin\alpha_i \tag{277}$$

$$A_\zeta \;=\; \sum_i \sum_{l=0}^{L} A_{\zeta,e,i,l}\, T_l(s)\cos\alpha_i + \sum_i \sum_{l=0}^{L} A_{\zeta,o,i,l}\, T_l(s)\sin\alpha_i, \tag{278}$$

where $T_l(s)$ are the Chebyshev polynomials and $\alpha_i \equiv m_i\theta - n_i\zeta$.

- The following internal arrays are declared in preset() :

`dAte(0,i)%s(l)` $\equiv A_{\theta,e,i,l}$

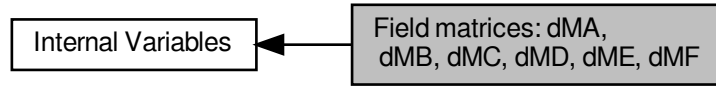`dAze(0,i)%s(l)` $\equiv A_{\zeta,e,i,l}$

`dAto(0,i)%s(l)` $\equiv A_{\theta,o,i,l}$

`dAzo(0,i)%s(l)` $\equiv A_{\zeta,o,i,l}$

## 8.38   Field matrices: dMA, dMB, dMC, dMD, dME, dMF

Collaboration diagram for Field matrices: dMA, dMB, dMC, dMD, dME, dMF:



**Variables**

- real, dimension(:,:), allocatable allglobal::dma

    *energy and helicity matrices; quadratic forms*
- real, dimension(:,:), allocatable allglobal::dmb

    *energy and helicity matrices; quadratic forms*
- real, dimension(:,:), allocatable allglobal::dmd

    *energy and helicity matrices; quadratic forms*
- real, dimension(:), allocatable allglobal::dmas

    *sparse version of dMA, data*
- real, dimension(:), allocatable allglobal::dmds

    *sparse version of dMD, data*
- integer, dimension(:), allocatable allglobal::idmas

    *sparse version of dMA and dMD, indices*
- integer, dimension(:), allocatable allglobal::jdmas

    *sparse version of dMA and dMD, indices*
- integer, dimension(:), allocatable allglobal::ndmasmax

    *number of elements for sparse matrices*
- integer, dimension(:), allocatable allglobal::ndmas

    *number of elements for sparse matrices*
- real, dimension(:), allocatable allglobal::dmg

    *what is this?*
- real, dimension(:,:), allocatable allglobal::solution

    *this is allocated in dforce; used in mp00ac and ma02aa; and is passed to packab*
- real, dimension(:,:,:), allocatable allglobal::gmreslastsolution

    *used to store the last solution for restarting GMRES*
- real, dimension(:), allocatable allglobal::mbpsi

    *matrix vector products*
- logical allglobal::liluprecond

    *whether to use ILU preconditioner for GMRES*
- real, dimension(:,:), allocatable allglobal::beltramiinverse

    *what is this?*
- real, dimension(:,:,:), allocatable allglobal::diotadxup

    *measured rotational transform on inner/outer interfaces for each volume; d(transform)/dx; (see dforce)*
- real, dimension(:,:,:), allocatable allglobal::ditgpdxtp

    *measured toroidal and poloidal current on inner/outer interfaces for each volume; d(Itor,Gpol)/dx; (see dforce)*
- real, dimension(:,:,:,:), allocatable allglobal::glambda

    *save initial guesses for iterative calculation of rotational-transform*
- integer allglobal::lmns

    *what is this?*

### 8.38.1  Detailed Description

- The energy, $W \equiv \int dv\ \mathbf{B} \cdot \mathbf{B}$, and helicity, $K \equiv \int dv\ \mathbf{A} \cdot \mathbf{B}$, functionals may be written

$$W = \frac{1}{2}\ a_i\ A_{i,j}\ a_j + a_i\ B_{i,j}\ \psi_j + \frac{1}{2}\ \psi_i\ C_{i,j}\ \psi_j \tag{279}$$

$$K = \frac{1}{2}\ a_i\ D_{i,j}\ a_j + a_i\ E_{i,j}\ \psi_j + \frac{1}{2}\ \psi_i\ F_{i,j}\ \psi_j \tag{280}$$
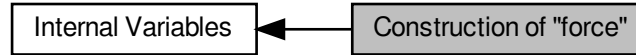
where $\mathbf{a} \equiv \{A_{\theta,e,i,l}, A_{\zeta,e,i,l}, A_{\theta,o,i,l}, A_{\zeta,o,i,l}, f_{e,i}, f_{o,i}\}$ contains the independent degrees of freedom and $\boldsymbol{\psi} \equiv \{\Delta\psi_t, \Delta\psi_p\}$.

- These are allocated and deallocated in dforce(), assigned in matrix(), and used in mp00ac() and (?) df00aa().

## 8.39 Construction of "force"

The force vector is comprised of `Bomn` and `Iomn`.

Collaboration diagram for Construction of "force":

```
┌─────────────────────┐        ┌─────────────────────────┐
│  Internal Variables │◄───────│  Construction of "force" │
└─────────────────────┘        └─────────────────────────┘
```

**Variables**

- real, dimension(:,:,:), allocatable allglobal::bemn

  *force vector; stellarator-symmetric (?)*
- real, dimension(:,:), allocatable allglobal::iomn

  *force vector; stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::somn

  *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::pomn

  *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::bomn

  *force vector; stellarator-symmetric (?)*
- real, dimension(:,:), allocatable allglobal::iemn

  *force vector; stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::semn

  *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::pemn

  *force vector; non-stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::bbe

  *force vector (?); stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::iio

  *force vector (?); stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::bbo

  *force vector (?); non-stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::iie

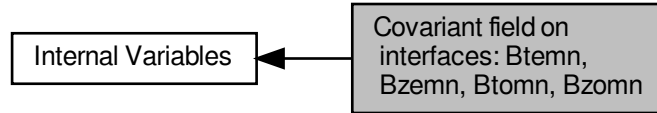  *force vector (?); non-stellarator-symmetric (?)*

### 8.39.1 Detailed Description

The force vector is comprised of `Bomn` and `Iomn`.

## 8.40 Covariant field on interfaces: Btemn, Bzemn, Btomn, Bzomn

The covariant field.

Collaboration diagram for Covariant field on interfaces: Btemn, Bzemn, Btomn, Bzomn:
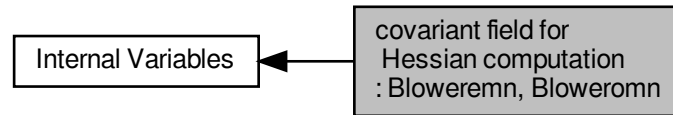


**Variables**

- real, dimension(:,:,:), allocatable allglobal::btemn

  *covariant $\theta$ cosine component of the tangential field on interfaces; stellarator-symmetric*
- real, dimension(:,:,:), allocatable allglobal::bzemn

  *covariant $\zeta$ cosine component of the tangential field on interfaces; stellarator-symmetric*
- real, dimension(:,:,:), allocatable allglobal::btomn

  *covariant $\theta$ sine component of the tangential field on interfaces; non-stellarator-symmetric*
- real, dimension(:,:,:), allocatable allglobal::bzomn

  *covariant $\zeta$ sine component of the tangential field on interfaces; non-stellarator-symmetric*

### 8.40.1 Detailed Description

The covariant field.

## 8.41 covariant field for Hessian computation: Bloweremn, Bloweromn

Collaboration diagram for covariant field for Hessian computation: Bloweremn, Bloweromn:



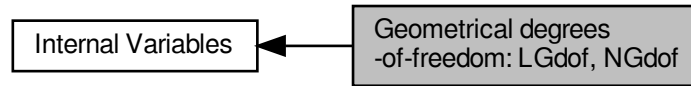**Variables**

- real, dimension(:,:), allocatable allglobal::bloweremn

  *covariant field for Hessian computation*
- real, dimension(:,:), allocatable allglobal::bloweromn

  *covariant field for Hessian computation*

### 8.41.1 Detailed Description

## 8.42 Geometrical degrees-of-freedom: LGdof, NGdof

The geometrical degrees-of-freedom.

Collaboration diagram for Geometrical degrees-of-freedom: LGdof, NGdof:



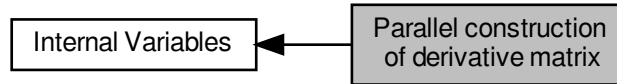**Variables**

- integer allglobal::lgdof

  *geometrical degrees of freedom associated with each interface*
- integer allglobal::ngdof

  *total geometrical degrees of freedom*

### 8.42.1 Detailed Description

The geometrical degrees-of-freedom.

## 8.43  Parallel construction of derivative matrix

Collaboration diagram for Parallel construction of derivative matrix:

```
┌──────────────────────┐        ┌──────────────────────┐
│  Internal Variables  │◀───────│  Parallel construction│
│                      │        │  of derivative matrix │
└──────────────────────┘        └──────────────────────┘
```

**Variables**
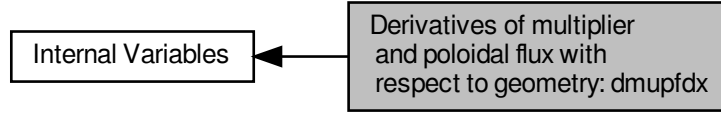
- real, dimension(:,:,:), allocatable allglobal::dbbdrz

    *derivative of magnetic field w.r.t. geometry (?)*

- real, dimension(:,:), allocatable allglobal::diidrz

    *derivative of spectral constraints w.r.t. geometry (?)*

- real, dimension(:,:,:,:,:), allocatable allglobal::dffdrz

    *derivatives of $B^{\wedge}2$ at the interfaces wrt geometry*

- real, dimension(:,:,:,:), allocatable allglobal::dbbdmp

    *derivatives of $B^{\wedge}2$ at the interfaces wrt mu and dpflux*

### 8.43.1  Detailed Description

- The derivatives of force-balance, $[[p + B^2/2]]$, and the spectral constraints (see sw03aa()), with respect to the interface geometry is constructed in parallel by dforce().

- force-balance across the $l$-th interface depends on the fields in the adjacent interfaces.

## 8.44 Derivatives of multiplier and poloidal flux with respect to geometry: dmupfdx

Collaboration diagram for Derivatives of multiplier and poloidal flux with respect to geometry: dmupfdx:



**Variables**

- real, dimension(:,:,:,:,:), allocatable allglobal::dmupfdx

  *derivatives of mu and dpflux wrt geometry at constant interface transform*
- logical allglobal::lhessianallocated

  *flag to indicate that force gradient matrix is allocated (?)*
- real, dimension(:,:), allocatable allglobal::hessian

  *force gradient matrix (?)*
- real, dimension(:,:), allocatable allglobal::dessian

  *derivative of force gradient matrix (?)*

### 8.44.1 Detailed Description

- The information in dmupfdx describes how the helicity multiplier, $\mu$, and the enclosed poloidal flux, $\Delta \psi_p$, must vary as the geometry is varied in order to satisfy the interface transform constraint.

- The internal variable dmupfdx(1:Mvol,1:2,1:LGdof,0:1) is allocated/deallocated in newton(), and hesian() if selected.

- The magnetic field depends on the Fourier harmonics of both the inner and outer interface geometry (represented here as $x_j$), the helicity multiplier, and the enclosed poloidal flux, i.e. $\mathbf{B}_\pm = \mathbf{B}_\pm(x_j, \mu, \Delta \psi_p)$, so that

$$\delta \mathbf{B}_\pm = \frac{\partial \mathbf{B}_\pm}{\partial x_j} \delta x_j + \frac{\partial \mathbf{B}_\pm}{\partial \mu} \delta \mu + \frac{\partial \mathbf{B}_\pm}{\partial \Delta \psi_p} \delta \Delta \psi_p. \tag{281}$$

- This information is used to adjust the calculation of how force-balance, i.e. $B^2$ at the interfaces, varies with geometry at fixed interface rotational transform. Given

$$B_\pm^2 = B_\pm^2(x_j, \mu, \Delta \psi_p), \tag{282}$$

we may derive

$$\frac{\partial B_\pm^2}{\partial x_j} = \frac{\partial B_\pm^2}{\partial x_j} + \frac{\partial B_\pm^2}{\partial \mu} \frac{\partial \mu}{\partial x_j} + \frac{\partial B_\pm^2}{\partial \Delta \psi_p} \frac{\partial \Delta \psi_p}{\partial x_j} \tag{283}$$
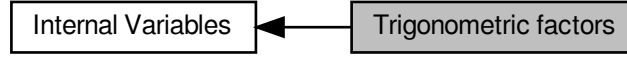
- The constraint to be enforced is that $\mu$ and $\Delta\psi_p$ must generally vary as the geometry is varied if the value of the rotational-transform constraint on the inner/outer interface is to be preserved, i.e.

$$
\begin{pmatrix}
\dfrac{\partial \iota_-}{\partial \mathbf{B}_-} \cdot \dfrac{\partial \mathbf{B}_-}{\partial \mu} & , & \dfrac{\partial \iota_-}{\partial \mathbf{B}_-} \cdot \dfrac{\partial \mathbf{B}_-}{\partial \Delta\psi_p} \\
\dfrac{\partial \iota_+}{\partial \mathbf{B}_+} \cdot \dfrac{\partial \mathbf{B}_+}{\partial \mu} & , & \dfrac{\partial \iota_+}{\partial \mathbf{B}_+} \cdot \dfrac{\partial \mathbf{B}_+}{\partial \Delta\psi_p}
\end{pmatrix}
\begin{pmatrix}
\dfrac{\partial \mu}{\partial x_j} \\
\dfrac{\partial \Delta\psi_p}{\partial x_j}
\end{pmatrix}
= -
\begin{pmatrix}
\dfrac{\partial \iota_-}{\partial \mathbf{B}_-} \cdot \dfrac{\partial \mathbf{B}_-}{\partial x_j} \\
\dfrac{\partial \iota_+}{\partial \mathbf{B}_+} \cdot \dfrac{\partial \mathbf{B}_+}{\partial x_j}
\end{pmatrix}.
\tag{284}
$$

- This $2 \times 2$ linear equation is solved in dforce() and the derivatives of the rotational-transform are given in `diotadxup`, see preset.f90 .

- A finite-difference estimate is computed if `Lcheck==4`.

## 8.45 Trigonometric factors

Collaboration diagram for Trigonometric factors:

```
┌────────────────────┐      ┌────────────────────────┐
│ Internal Variables │◄─────│ Trigonometric factors  │
└────────────────────┘      └────────────────────────┘
```

**Variables**

- real, dimension(:,:), allocatable allglobal::cosi

  *some precomputed cosines*
- real, dimension(:,:), allocatable allglobal::sini

  *some precomputed sines*
- real, dimension(:), allocatable allglobal::gteta

  *something related to $\sqrt{g}$ and $\theta$ ?*
- real, dimension(:), allocatable allglobal::gzeta

  *something related to $\sqrt{g}$ and $\zeta$ ?*
- real, dimension(:), allocatable allglobal::ajk

  *definition of coordinate axis*
- real, dimension(:,:,:,:), allocatable allglobal::dradr

  *derivatives of coordinate axis*
- real, dimension(:,:,:,:), allocatable allglobal::dradz

  *derivatives of coordinate axis*
- real, dimension(:,:,:,:), allocatable allglobal::dzadr

  *derivatives of coordinate axis*
- real, dimension(:,:,:,:), allocatable allglobal::dzadz

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable allglobal::drodr

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable allglobal::drodz

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable allglobal::dzodr

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable allglobal::dzodz

  *derivatives of coordinate axis*
- integer, dimension(:,:), allocatable allglobal::djkp

  *for calculating cylindrical volume*
- integer, dimension(:,:), allocatable allglobal::djkm

  *for calculating cylindrical volume*
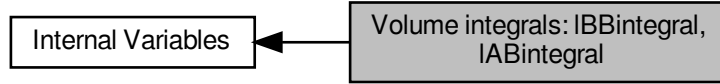
### 8.45.1 Detailed Description

- To facilitate construction of the metric integrals, various trigonometric identities are exploited.

- The following are used for volume integrals (see volume() ):

$$a_{i,j,k} = 4 \, m_k \oint\oint d\theta d\zeta \quad \cos(\alpha_i)\cos(\alpha_j)\cos(\alpha_k)/(2\pi)^2, \tag{285}$$

$$b_{i,j,k} = 4 \, m_j \oint\oint d\theta d\zeta \quad \cos(\alpha_i)\sin(\alpha_j)\sin(\alpha_k)/(2\pi)^2, \tag{286}$$

## 8.46 Volume integrals: IBBintegral, IABintegral

Collaboration diagram for Volume integrals: IBBintegral, IABintegral:

```
┌──────────────────┐        ┌──────────────────────────┐
│ Internal Variables │ ◄──── │ Volume integrals: IBBintegral, │
│                  │        │        IABintegral        │
└──────────────────┘        └──────────────────────────┘
```

**Variables**

- real, dimension(:), allocatable allglobal::lbbintegral

  *B.B integral.*
- real, dimension(:), allocatable allglobal::labintegral

  *A.B integral.*
- real, dimension(:), allocatable allglobal::vvolume

  *volume integral of $\sqrt{g}$; computed in volume*
- real allglobal::dvolume

  *derivative of volume w.r.t. interface geometry*

### 8.46.1 Detailed Description

- The energy functional, $F \equiv \sum_l F_l$, where

$$F_l \equiv \left( \int_{\mathcal{V}_l} \frac{p_l}{\gamma - 1} + \frac{B_l^2}{2} dv \right) = \frac{P_l}{\gamma - 1} V_l^{1-\gamma} + \int_{\mathcal{V}_l} \frac{B_l^2}{2} dv, \tag{287}$$

where the second expression is derived using $p_l V_l^{\gamma} = P_l$, where $P_l$ is the adiabatic-constant. In Eqn. (287), it is implicit that $\mathbf{B}$ satisfies (i) the toroidal and poloidal flux constraints; (ii) the interface constraint, $\mathbf{B} \cdot \nabla s = 0$; and (iii) the helicity constraint (or the transform constraint).

- The derivatives of $F_l$ with respect to the inner and outer adjacent interface geometry are stored in dFF(1↩ :Nvol,0:1,0:mn+mn-1), where

$F_l \equiv$ dFF(l,0, 0)

$\partial F_l / \partial R_{l-1,j} \equiv$ dFF(ll,0, j)

$\partial F_l / \partial Z_{l-1,j} \equiv$ dFF(ll,0,mn j)

$\partial F_l / \partial R_{l,j} \equiv$ dFF(ll,1, j)

$\partial F_l / \partial Z_{l,j} \equiv$ dFF(ll,1,mn j)

- The volume integrals $\int dv$, $\int B^2 \, dv$ and $\int \mathbf{A} \cdot \mathbf{B} \, dv$ in each volume are computed and saved in volume(0↩ :2,1:Nvol).

## 8.47 Internal global variables

internal global variables; internal logical variables; default values are provided here; these may be changed according to input values

Collaboration diagram for Internal global variables:



**Variables**

- integer allglobal::ivol

  *labels volume; some subroutines (called by NAG) are fixed argument list but require the volume label*

- real allglobal::gbzeta

  *toroidal (contravariant) field; calculated in bfield; required to convert $\dot{\theta}$ to $B^{\theta}$, $\dot{s}$ to $B^{s}$*

- integer, dimension(:), allocatable allglobal::iquad

  *internal copy of Nquad*

- real, dimension(:,:), allocatable allglobal::gaussianweight

  *weights for Gaussian quadrature*

- real, dimension(:,:), allocatable allglobal::gaussianabscissae

  *abscissae for Gaussian quadrature*

- logical allglobal::lblinear

  *controls selection of Beltrami field solver; depends on LBeltrami*

- logical allglobal::lbnewton

  *controls selection of Beltrami field solver; depends on LBeltrami*

- logical allglobal::lbsequad

  *controls selection of Beltrami field solver; depends on LBeltrami*

- real, dimension(1:3) allglobal::orzp

  *used in mg00aa() to determine $(s, \theta, \zeta)$ given $(R, Z, \varphi)$*
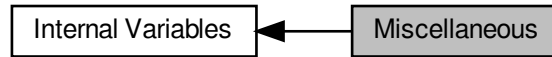
### 8.47.1 Detailed Description

internal global variables; internal logical variables; default values are provided here; these may be changed according to input values

## 8.48 Miscellaneous

The following are miscellaneous flags required for the virtual casing field, external (vacuum) field integration, ...

Collaboration diagram for Miscellaneous:



**Variables**

- integer allglobal::globaljk

  *labels position*
- real, dimension(:,:), allocatable allglobal::dxyz

  *computational boundary; position*

- real, dimension(:,:), allocatable allglobal::nxyz

  *computational boundary; normal*

- real, dimension(:,:), allocatable allglobal::jxyz

  *plasma boundary; surface current*
- real, dimension(1:2) allglobal::tetazeta

  *what is this?*
- real allglobal::virtualcasingfactor = -one / (four∗pi)

  *this agrees with diagno*
- integer allglobal::iberror

  *for computing error in magnetic field*
- integer allglobal::nfreeboundaryiterations

  *number of free-boundary iterations already performed*
- integer, parameter allglobal::node = 2

  *best to make this global for consistency between calling and called routines*
- logical allglobal::first_free_bound = .false.

  *flag to indicate that this is the first free-boundary iteration*

### 8.48.1 Detailed Description

The following are miscellaneous flags required for the virtual casing field, external (vacuum) field integration, ...

# 9 Module Documentation

## 9.1 allglobal Module Reference

global variable storage used as "workspace" throughout the code

**Data Types**

- type derivative

  $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ *(?) More...*

**Functions/Subroutines**

- subroutine **build_vector_potential** (lvol, iocons, aderiv, tderiv)
- subroutine readin

  *The master node reads the input namelist and sets various internal variables. The relevant quantities are then broadcast.*
- subroutine wrtend

  *The restart file is written.*
- subroutine ismyvolume (vvol)

  *Check if volume vvol is associated to the corresponding MPI node.*
- subroutine whichcpuid (vvol, cpu_id)

  *Returns which MPI node is associated to a given volume.*

**Variables**

- integer myid

  *MPI rank of current CPU.*
- integer ncpu

  *number of MPI tasks*
- integer ismyvolumevalue

  *flag to indicate if a CPU is operating on its assigned volume*
- real cpus

  *initial time*
- real pi2nfp

  *pi2/nfp ; assigned in readin()*
- real pi2pi2nfp

  $4\pi^2 Nfp$
- real pi2pi2nfphalf

  $2\pi^2 Nfp$
- real pi2pi2nfpquart

  $\pi^2 Nfp$
- real forceerr

  *total force-imbalance*
- real energy

  *MHD energy.*
- real, dimension(:), allocatable ipdt

  *Toroidal pressure-driven current.*

- real, dimension(:,:), allocatable ipdtdpf

    *Toroidal pressure-driven current.*
- integer mvol

    *number of total volumes; equal to Nvol for fixed-boundary; equal to Nvol+1 for free-boundary*
- logical yesstellsym

    *internal shorthand copies of Istellsym, which is an integer input;*
- logical notstellsym

    *internal shorthand copies of Istellsym, which is an integer input;*
- logical **yesmatrixfree**
- logical notmatrixfree

    *to use matrix-free method or not*
- real, dimension(:,:), allocatable cheby

    *local workspace for evaluation of Chebychev polynomials*
- real, dimension(:,:,:), allocatable zernike

    *local workspace for evaluation of Zernike polynomials*
- real, dimension(:,:,:), allocatable tt

    *derivatives of Chebyshev polynomials at the inner and outer interfaces;*
- real, dimension(:,:,:,:), allocatable rtt

    *derivatives of Zernike polynomials at the inner and outer interfaces;*
- real, dimension(:,:), allocatable rtm

    $r^m$ *term of Zernike polynomials at the origin*
- real, dimension(:), allocatable zernikedof

    *Zernike degree of freedom for each* $m$.
- logical, dimension(:), allocatable imagneticok

    *used to indicate if Beltrami fields have been correctly constructed;*
- logical iconstraintok

    *Used to break iteration loops of slaves in the global constraint minimization.*
- real, dimension(:,:), allocatable beltramierror

    *to store the integral of* $|curlB\text{-}mu{*}B|$ *computed by jo00aa;*
- integer mn

    *total number of Fourier harmonics for coordinates/fields; calculated from Mpol, Ntor in readin()*
- integer, dimension(:), allocatable im

    *poloidal mode numbers for Fourier representation*
- integer, dimension(:), allocatable in

    *toroidal mode numbers for Fourier representation*
- real, dimension(:), allocatable halfmm

    *I saw this already somewhere...*
- real, dimension(:), allocatable regumm

    *I saw this already somewhere...*
- real rscale

    *no idea*
- real, dimension(:,:), allocatable psifactor

    *no idea*
- real, dimension(:,:), allocatable inifactor

    *no idea*
- real, dimension(:), allocatable bbweight

    *weight on force-imbalance harmonics; used in dforce()*
- real, dimension(:), allocatable mmpp

    *spectral condensation factors*
- integer mne

    *enhanced resolution for metric elements*

---

- integer, dimension(:), allocatable ime

    *enhanced poloidal mode numbers for metric elements*
- integer, dimension(:), allocatable ine

    *enhanced toroidal mode numbers for metric elements*
- integer mns

    *enhanced resolution for straight field line transformation*
- integer, dimension(:), allocatable ims

    *enhanced poloidal mode numbers for straight field line transformation*
- integer, dimension(:), allocatable ins

    *enhanced toroidal mode numbers for straight field line transformation*
- integer lmpol

    *what is this?*
- integer lntor

    *what is this?*
- integer smpol

    *what is this?*
- integer sntor

    *what is this?*
- real xoffset = 1.0

    *used to normalize NAG routines (which ones exacly where?)*
- real, dimension(:,:), allocatable irbc

    *cosine R harmonics of interface surface geometry; stellarator symmetric*
- real, dimension(:,:), allocatable izbs

    *sine Z harmonics of interface surface geometry; stellarator symmetric*
- real, dimension(:,:), allocatable irbs

    *sine R harmonics of interface surface geometry; non-stellarator symmetric*
- real, dimension(:,:), allocatable izbc

    *cosine Z harmonics of interface surface geometry; non-stellarator symmetric*
- real, dimension(:,:), allocatable drbc

    *cosine R harmonics of interface surface geometry; stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable dzbs

    *sine Z harmonics of interface surface geometry; stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable drbs

    *sine R harmonics of interface surface geometry; non-stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable dzbc

    *cosine Z harmonics of interface surface geometry; non-stellarator symmetric; linear deformation*
- real, dimension(:,:), allocatable irij

    *interface surface geometry; real space*
- real, dimension(:,:), allocatable izij

    *interface surface geometry; real space*
- real, dimension(:,:), allocatable drij

    *interface surface geometry; real space*
- real, dimension(:,:), allocatable dzij

    *interface surface geometry; real space*
- real, dimension(:,:), allocatable trij

    *interface surface geometry; real space*
- real, dimension(:,:), allocatable tzij

    *interface surface geometry; real space*
- real, dimension(:), allocatable ivns

    *sine harmonics of vacuum normal magnetic field on interfaces; stellarator symmetric*
- real, dimension(:), allocatable ibns

*sine harmonics of plasma normal magnetic field on interfaces; stellarator symmetric*

- real, dimension(:), allocatable ivnc

    *cosine harmonics of vacuum normal magnetic field on interfaces; non-stellarator symmetric*

- real, dimension(:), allocatable ibnc

    *cosine harmonics of plasma normal magnetic field on interfaces; non-stellarator symmetric*

- real, dimension(:), allocatable lrbc

    *local workspace*

- real, dimension(:), allocatable lzbs

    *local workspace*

- real, dimension(:), allocatable lrbs

    *local workspace*

- real, dimension(:), allocatable lzbc

    *local workspace*

- integer nt

    *discrete resolution along $\theta$ of grid in real space*

- integer nz

    *discrete resolution along $\zeta$ of grid in real space*

- integer ntz

    *discrete resolution; Ntz=Nt∗Nz shorthand*

- integer hnt

    *discrete resolution; Ntz=Nt∗Nz shorthand*

- integer hnz

    *discrete resolution; Ntz=Nt∗Nz shorthand*

- real sontz

    *one / sqrt (one∗Ntz); shorthand*

- real, dimension(:,:,:), allocatable rij

    *real-space grid; R*

- real, dimension(:,:,:), allocatable zij

    *real-space grid; Z*

- real, dimension(:,:,:), allocatable xij

    *what is this?*

- real, dimension(:,:,:), allocatable yij

    *what is this?*

- real, dimension(:,:), allocatable sg

    *real-space grid; jacobian and its derivatives*

- real, dimension(:,:,:,:,:), allocatable guvij

    *real-space grid; metric elements*

- real, dimension(:,:,:), allocatable gvuij

    *real-space grid; metric elements (?); 10 Dec 15;*

- real, dimension(:,:,:,:,:), allocatable guvijsave

    *what is this?*

- integer, dimension(:,:), allocatable ki

    *identification of Fourier modes*

- integer, dimension(:,:,:), allocatable kijs

    *identification of Fourier modes*

- integer, dimension(:,:,:), allocatable kija

    *identification of Fourier modes*

- integer, dimension(:), allocatable iotakkii

    *identification of Fourier modes*

- integer, dimension(:,:), allocatable iotaksub

    *identification of Fourier modes*

- integer, dimension(:,:), allocatable iotakadd

    *identification of Fourier modes*
- integer, dimension(:,:), allocatable iotaksgn

    *identification of Fourier modes*
- real, dimension(:), allocatable efmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable ofmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable cfmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable sfmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable evmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable odmn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable comn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable simn

    *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable ijreal

    *what is this ?*
- real, dimension(:), allocatable ijimag

    *what is this ?*
- real, dimension(:), allocatable jireal

    *what is this ?*
- real, dimension(:), allocatable jiimag

    *what is this ?*
- real, dimension(:), allocatable jkreal

    *what is this ?*
- real, dimension(:), allocatable jkimag

    *what is this ?*
- real, dimension(:), allocatable kjreal

    *what is this ?*
- real, dimension(:), allocatable kjimag

    *what is this ?*
- real, dimension(:,:,:), allocatable bsupumn

    *tangential field on interfaces; $\theta$-component; required for virtual casing construction of field; 11 Oct 12*
- real, dimension(:,:,:), allocatable bsupvmn

    *tangential field on interfaces; $\zeta$-component; required for virtual casing construction of field; 11 Oct 12*
- real, dimension(:,:), allocatable goomne

    *described in preset()*
- real, dimension(:,:), allocatable goomno

    *described in preset()*
- real, dimension(:,:), allocatable gssmne

    *described in preset()*
- real, dimension(:,:), allocatable gssmno

    *described in preset()*
- real, dimension(:,:), allocatable gstmne

    *described in preset()*
- real, dimension(:,:), allocatable gstmno

*described in preset()*

- real, dimension(:,:), allocatable gszmne

    *described in preset()*
- real, dimension(:,:), allocatable gszmno

    *described in preset()*
- real, dimension(:,:), allocatable gttmne

    *described in preset()*
- real, dimension(:,:), allocatable gttmno

    *described in preset()*
- real, dimension(:,:), allocatable gtzmne

    *described in preset()*
- real, dimension(:,:), allocatable gtzmno

    *described in preset()*
- real, dimension(:,:), allocatable gzzmne

    *described in preset()*
- real, dimension(:,:), allocatable gzzmno

    *described in preset()*
- real, dimension(:,:,:,:), allocatable dtoocc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable dtoocs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable dtoosc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable dtooss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable ttsscc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable ttsscs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable ttsssc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable ttssss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdstcc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdstcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdstsc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdstss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdszcc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdszcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdszsc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable tdszss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable ddttcc

    *volume-integrated Chebychev-metrics; see matrix()*

- real, dimension(:,:,:,:,:), allocatable ddttcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddttsc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddttss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddtzcc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddtzcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddtzsc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddtzss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddzzcc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddzzcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddzzsc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:,:), allocatable ddzzss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:), allocatable tsc

    *what is this?*
- real, dimension(:,:), allocatable tss

    *what is this?*
- real, dimension(:,:), allocatable dtc

    *what is this?*
- real, dimension(:,:), allocatable dts

    *what is this?*
- real, dimension(:,:), allocatable dzc

    *what is this?*
- real, dimension(:,:), allocatable dzs

    *what is this?*
- real, dimension(:,:), allocatable ttc

    *what is this?*
- real, dimension(:,:), allocatable tzc

    *what is this?*
- real, dimension(:,:), allocatable tts

    *what is this?*
- real, dimension(:,:), allocatable tzs

    *what is this?*
- real, dimension(:), allocatable dtflux

    $\delta\psi_{toroidal}$ *in each annulus*
- real, dimension(:), allocatable dpflux

    $\delta\psi_{poloidal}$ *in each annulus*
- real, dimension(:), allocatable sweight

    *minimum poloidal length constraint weight*
- integer, dimension(:), allocatable nadof

    *degrees of freedom in Beltrami fields in each annulus*
- integer, dimension(:), allocatable nfielddof

*degrees of freedom in Beltrami fields in each annulus, field only, no Lagrange multipliers*

- type(subgrid), dimension(:,:,:,:), allocatable ate

  *magnetic vector potential cosine Fourier harmonics; stellarator-symmetric*

- type(subgrid), dimension(:,:,:,:), allocatable aze

  *magnetic vector potential cosine Fourier harmonics; stellarator-symmetric*

- type(subgrid), dimension(:,:,:,:), allocatable ato

  *magnetic vector potential sine Fourier harmonics; non-stellarator-symmetric*

- type(subgrid), dimension(:,:,:,:), allocatable azo

  *magnetic vector potential sine Fourier harmonics; non-stellarator-symmetric*

- integer, dimension(:,:), allocatable lma

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lmb

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lmc

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lmd

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lme

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lmf

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lmg

  *Lagrange multipliers (?)*

- integer, dimension(:,:), allocatable lmh

  *Lagrange multipliers (?)*

- real, dimension(:,:), allocatable lmavalue

  *what is this?*

- real, dimension(:,:), allocatable lmbvalue

  *what is this?*

- real, dimension(:,:), allocatable lmcvalue

  *what is this?*

- real, dimension(:,:), allocatable lmdvalue

  *what is this?*

- real, dimension(:,:), allocatable lmevalue

  *what is this?*

- real, dimension(:,:), allocatable lmfvalue

  *what is this?*

- real, dimension(:,:), allocatable lmgvalue

  *what is this?*

- real, dimension(:,:), allocatable lmhvalue

  *what is this?*

- integer, dimension(:,:), allocatable fso

  *what is this?*

- integer, dimension(:,:), allocatable fse

  *what is this?*

- logical lcoordinatesingularity

  *set by* `LREGION` *macro; true if inside the innermost volume*

- logical lplasmaregion

  *set by* `LREGION` *macro; true if inside the plasma region*

- logical lvacuumregion

  *set by* `LREGION` *macro; true if inside the vacuum region*

- logical lsavedguvij

  *flag used in matrix free*
- logical localconstraint

  *what is this?*
- real, dimension(:,:), allocatable dma

  *energy and helicity matrices; quadratic forms*
- real, dimension(:,:), allocatable dmb

  *energy and helicity matrices; quadratic forms*
- real, dimension(:,:), allocatable dmd

  *energy and helicity matrices; quadratic forms*
- real, dimension(:), allocatable dmas

  *sparse version of dMA, data*
- real, dimension(:), allocatable dmds

  *sparse version of dMD, data*
- integer, dimension(:), allocatable idmas

  *sparse version of dMA and dMD, indices*
- integer, dimension(:), allocatable jdmas

  *sparse version of dMA and dMD, indices*
- integer, dimension(:), allocatable ndmasmax

  *number of elements for sparse matrices*
- integer, dimension(:), allocatable ndmas

  *number of elements for sparse matrices*
- real, dimension(:), allocatable dmg

  *what is this?*
- real, dimension(:,:), allocatable solution

  *this is allocated in dforce; used in mp00ac and ma02aa; and is passed to packab*
- real, dimension(:,:,:), allocatable gmreslastsolution

  *used to store the last solution for restarting GMRES*
- real, dimension(:), allocatable mbpsi

  *matrix vector products*
- logical liluprecond

  *whether to use ILU preconditioner for GMRES*
- real, dimension(:,:), allocatable beltramiinverse

  *what is this?*
- real, dimension(:,:,:), allocatable diotadxup

  *measured rotational transform on inner/outer interfaces for each volume; d(transform)/dx; (see dforce)*
- real, dimension(:,:,:), allocatable ditgpdxtp

  *measured toroidal and poloidal current on inner/outer interfaces for each volume; d(Itor,Gpol)/dx; (see dforce)*
- real, dimension(:,:,:,:), allocatable glambda

  *save initial guesses for iterative calculation of rotational-transform*
- integer lmns

  *what is this?*
- real, dimension(:,:,:), allocatable bemn

  *force vector; stellarator-symmetric (?)*
- real, dimension(:,:), allocatable iomn

  *force vector; stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable somn

  *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable pomn

  *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable bomn

*force vector; stellarator-symmetric (?)*

- real, dimension(:,:), allocatable iemn

    *force vector; stellarator-symmetric (?)*

- real, dimension(:,:,:), allocatable semn

    *force vector; non-stellarator-symmetric (?)*

- real, dimension(:,:,:), allocatable pemn

    *force vector; non-stellarator-symmetric (?)*

- real, dimension(:), allocatable bbe

    *force vector (?); stellarator-symmetric (?)*

- real, dimension(:), allocatable iio

    *force vector (?); stellarator-symmetric (?)*

- real, dimension(:), allocatable bbo

    *force vector (?); non-stellarator-symmetric (?)*

- real, dimension(:), allocatable iie

    *force vector (?); non-stellarator-symmetric (?)*

- real, dimension(:,:,:), allocatable btemn

    *covariant $\theta$ cosine component of the tangential field on interfaces; stellarator-symmetric*

- real, dimension(:,:,:), allocatable bzemn

    *covariant $\zeta$ cosine component of the tangential field on interfaces; stellarator-symmetric*

- real, dimension(:,:,:), allocatable btomn

    *covariant $\theta$ sine component of the tangential field on interfaces; non-stellarator-symmetric*

- real, dimension(:,:,:), allocatable bzomn

    *covariant $\zeta$ sine component of the tangential field on interfaces; non-stellarator-symmetric*

- real, dimension(:,:), allocatable bloweremn

    *covariant field for Hessian computation*

- real, dimension(:,:), allocatable bloweromn

    *covariant field for Hessian computation*

- integer lgdof

    *geometrical degrees of freedom associated with each interface*

- integer ngdof

    *total geometrical degrees of freedom*

- real, dimension(:,:,:), allocatable dbbdrz

    *derivative of magnetic field w.r.t. geometry (?)*

- real, dimension(:,:), allocatable diidrz

    *derivative of spectral constraints w.r.t. geometry (?)*

- real, dimension(:,:,:,:,:), allocatable dffdrz

    *derivatives of $B^2$ at the interfaces wrt geometry*

- real, dimension(:,:,:,:), allocatable dbbdmp

    *derivatives of $B^2$ at the interfaces wrt mu and dpflux*

- real, dimension(:,:,:,:,:), allocatable dmupfdx

    *derivatives of mu and dpflux wrt geometry at constant interface transform*

- logical lhessianallocated

    *flag to indicate that force gradient matrix is allocated (?)*

- real, dimension(:,:), allocatable hessian

    *force gradient matrix (?)*

- real, dimension(:,:), allocatable dessian

    *derivative of force gradient matrix (?)*

- real, dimension(:,:), allocatable cosi

    *some precomputed cosines*

- real, dimension(:,:), allocatable sini

    *some precomputed sines*

- real, dimension(:), allocatable gteta

  *something related to $\sqrt{g}$ and $\theta$ ?*
- real, dimension(:), allocatable gzeta

  *something related to $\sqrt{g}$ and $\zeta$ ?*
- real, dimension(:), allocatable ajk

  *definition of coordinate axis*
- real, dimension(:,:,:,:), allocatable dradr

  *derivatives of coordinate axis*
- real, dimension(:,:,:,:), allocatable dradz

  *derivatives of coordinate axis*
- real, dimension(:,:,:,:), allocatable dzadr

  *derivatives of coordinate axis*
- real, dimension(:,:,:,:), allocatable dzadz

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable drodr

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable drodz

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable dzodr

  *derivatives of coordinate axis*
- real, dimension(:,:,:), allocatable dzodz

  *derivatives of coordinate axis*
- integer, dimension(:,:), allocatable djkp

  *for calculating cylindrical volume*
- integer, dimension(:,:), allocatable djkm

  *for calculating cylindrical volume*
- real, dimension(:), allocatable lbbintegral

  *B.B integral.*
- real, dimension(:), allocatable labintegral

  *A.B integral.*
- real, dimension(:), allocatable vvolume

  *volume integral of $\sqrt{g}$; computed in volume*
- real dvolume

  *derivative of volume w.r.t. interface geometry*
- integer ivol

  *labels volume; some subroutines (called by NAG) are fixed argument list but require the volume label*
- real gbzeta

  *toroidal (contravariant) field; calculated in bfield; required to convert $\dot{\theta}$ to $B^{\theta}$, $\dot{s}$ to $B^{s}$*
- integer, dimension(:), allocatable iquad

  *internal copy of Nquad*
- real, dimension(:,:), allocatable gaussianweight

  *weights for Gaussian quadrature*
- real, dimension(:,:), allocatable gaussianabscissae

  *abscissae for Gaussian quadrature*
- logical lblinear

  *controls selection of Beltrami field solver; depends on LBeltrami*
- logical lbnewton

  *controls selection of Beltrami field solver; depends on LBeltrami*
- logical lbsequad

  *controls selection of Beltrami field solver; depends on LBeltrami*
- real, dimension(1:3) orzp

*used in mg00aa() to determine $(s, \theta, \zeta)$ given $(R, Z, \varphi)$*

- type(derivative) dbdx

  $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ *(?)*

- integer globaljk

  *labels position*

- real, dimension(:,:), allocatable dxyz

  *computational boundary; position*


- real, dimension(:,:), allocatable nxyz

  *computational boundary; normal*


- real, dimension(:,:), allocatable jxyz

  *plasma boundary; surface current*

- real, dimension(1:2) tetazeta

  *what is this?*

- real virtualcasingfactor = -one / (four∗pi)

  *this agrees with diagno*

- integer iberror

  *for computing error in magnetic field*

- integer nfreeboundaryiterations

  *number of free-boundary iterations already performed*

- integer, parameter node = 2

  *best to make this global for consistency between calling and called routines*

- logical first_free_bound = .false.

  *flag to indicate that this is the first free-boundary iteration*


### 9.1.1 Detailed Description

global variable storage used as "workspace" throughout the code

### 9.1.2 Function/Subroutine Documentation

#### 9.1.2.1 readin() `subroutine allglobal::readin`

The master node reads the input namelist and sets various internal variables. The relevant quantities are then broadcast.

**machine precision**

- The machine precision `machprec` is determined using myprec(), which is similar to corresponding the NAG routine.

- The variables `vsmall`, `small` and `sqrtmachprec` are set.

**input file extension (command line argument)**

- The input file name, `ext`, is given as the first command line input, and the input file itself is `ext.sp`

- Additional command line inputs recognized are:
  - `-help`, `-h` : will give help information to user; under construction
  - `-readin` will immediately set `Wreadin=T` ; this may be over-ruled when the `screenlist` is read

**reading of physicslist**

- The internal variable, `Mvol=Nvol+Lfreebound` , gives the number of computational domains.

- The input value for the fluxes enclosed within each interface, `tflux(1:Mvol)` and `tflux(1:Mvol)`, are immediately normalized:

  `tflux(1:Mvol)` $\rightarrow$ `tflux(1:Mvol)/tflux(Nvol)`.

  `pflux(1:Mvol)` $\rightarrow$ `pflux(1:Mvol)/tflux(Nvol)`.

  The input $\Phi_{edge} \equiv$ `phiedge` will provide the total toroidal flux; see [preset()](preset()).

- The input value for the toroidal current constraint (`Isurf(1:Mvol)` and `Ivolume(1:Mvol)` ) are also immediately normalized, using `curtor` . $Ivolume \rightarrow Ivolume \cdot \frac{curtor}{\sum_i Isurf_i+Ivolume_i}$ $Isurf \rightarrow Isurf \cdot \frac{curtor}{\sum_i Isurf_i+Ivolume_i}$

**Current profiles normalization**

In case of a free boundary calculation (`Lfreebound=1`) and using a current constraint (`Lconstraint=3`), the current profiles are renormalized in order to match the linking current `curtor`. More specifically,

$$Isurf_i \quad \rightarrow \quad Isurf_i \cdot \frac{curtor}{\sum_{i=1}^{Mvol-1} Isurf_i+Ivol_i} Ivol_i \quad \rightarrow \quad Ivol_i \cdot \frac{curtor}{\sum_{i=1}^{Mvol-1} Isurf_i + Ivol_i} \tag{288}$$

Finally, the volume current in the vacuum region is set to $0$.

**reading of numericlist**

**reading of locallist**

**reading of globallist**

**reading of diagnosticslist**

**reading of screenlist**

**broadcast command line input**

**broadcast physicslist**

**broadcast numericlist**

**broadcast globallist**

**broadcast locallist**

**broadcast diagnosticslist**

**broadcast screenlist**

**set internal parameters that depend on physicslist**

**total number of volumes: Mvol**

- The number of plasma volumes is `Mvol = Nvol + Lfreebound`.

**Fourier mode identification: mn, im(1:mn) and in(1:mn)**

- The Fourier description of even periodic functions is

$$f(\theta,\zeta) = \sum_{n=0}^{N} f_{0,n}\cos(-n\zeta) + \sum_{m=1}^{M}\sum_{n=-N}^{N} f_{m,n}\cos(m\theta - n\zeta), \tag{289}$$

where the resolution is given on input, $M \equiv \texttt{Mpol}$ and $N \equiv \texttt{Ntor}$.

- For convenience, the Fourier summations are written as

$$f(s,\theta,\zeta) \quad = \quad \sum_{j} f_j(s)\cos(m_j\theta - n_j\zeta), \tag{290}$$

for $j = 1, \texttt{mn}$, where $\texttt{mn} = N + 1 + M(2N+1)$.

- The integer arrays `im(1:mn)` and `in(1:mn)` contain the $m_j$ and $n_j$.

- The array `in` includes the `Nfp` factor.

**regularization factor: halfmm(1:mn), regumm(1:mn)**

- The "regularization" factor, `halfmm(1:mn) = im(1:mn) * half`, is real.

- This is used in [lforce()](), [bfield()](), [stzxyz()](), [coords()](), [jo00aa()](), [ma00aa()](), sc00aa() and [tr00ab()]().

**extended resolution Fourier mode identification: mne, ime and ine**

- The "extended" Fourier resolution is defined by $\texttt{lMpol} = 4\,\texttt{Mpol}$, $\texttt{lNtor} = 4\texttt{Ntor}$.

**Fourier mode identification for straight-fieldline angle: mns, ims and ins**

**set internal parameters that depend on numericlist**

**set internal parameters that depend on locallist**

**set internal parameters that depend on globallist**

**set internal parameters that depend on diagnosticslist**

**geometry: iRbc(1:mn,0:Mvol, iZbs(1:mn,0:Mvol), iRbs(1:mn,0:Mvol) and iZbc(1:mn,0:Mvol)**
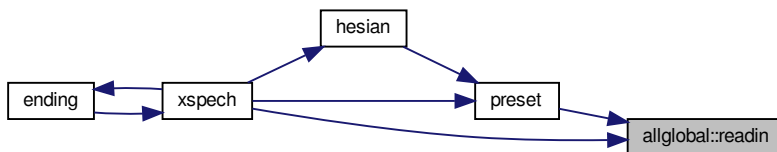
- `iRbc`, `iZbs`, `iRbs` and `iZbc` : Fourier harmonics of interface geometry

- `iVns`, `iVnc`, `iBns` and `iBns` : Fourier harmonics of normal field at computational boundary

**construction of coordinate axis: ajk**

- This is only used in [rzaxis()]() to perform the poloidal integration and is defined quite simply:

  `ajk[i]` $\equiv 2\pi$ if $m_i = 0$, and

  `ajk[i]` $\equiv 0$ if $m_i \neq 0$.

References inputlist::adiabatic, ajk, beltramierror, inputlist::bnc, inputlist::bns, inputlist::c05factor, inputlist↩
::c05xmax, inputlist::c05xtol, cpus, inputlist::curpol, inputlist::curtor, dforce(), inputlist::dpp, inputlist::dqq, drbc,
drbs, inputlist::drz, dzbc, dzbs, inputlist::epsgmres, inputlist::epsilon, inputlist::epsilu, inputlist::escale, inputlist::ext,
inputlist::forcetol, inputlist::gamma, inputlist::gbnbld, inputlist::gbntol, gi00ab(), constants::half, halfmm, inputlist↩
::helicity, ibnc, ibns, inputlist::igeometry, im, ime, inputlist::imethod, inputlist::impol, ims, in, ine, ins, inputlist↩
::intor, inputlist::iorder, inputlist::iota, inputlist::iotatol, inputlist::iprecon, irbc, irbs, inputlist::istellsym, inputlist::isurf,
fileunits::iunit, ivnc, ivns, inputlist::ivolume, izbc, izbs, inputlist::ladiabatic, inputlist::lautoinitbn, inputlist::lbeltrami,
inputlist::lcheck, inputlist::lconstraint, inputlist::lerrortype, inputlist::lextrap, inputlist::lfindzero, inputlist::lfreebound,
inputlist::lgmresprec, inputlist::lhevalues, inputlist::lhevectors, inputlist::lhmatrix, inputlist::linitgues, inputlist↩
::linitialize, inputlist::lmatsolver, lmpol, lntor, inputlist::lp, inputlist::lperturbed, inputlist::lq, inputlist::lrad, inputlist↩
::lreadgf, inputlist::lreflect, inputlist::lrzaxis, inputlist::lsparse, inputlist::lsvdiota, inputlist::ltiming, inputlist::lzerovac,
numerical::machprec, inputlist::maxrndgues, inputlist::mfreeits, inputlist::mmpol, mn, mne, mns, inputlist::mntor,
inputlist::mnvol, inputlist::mpol, inputlist::mregular, inputlist::mu, inputlist::mupfits, inputlist::mupftol, mvol, myid,
numerical::myprec(), inputlist::ndiscrete, inputlist::nfp, inputlist::ngrid, inputlist::nitergmres, notstellsym, inputlist↩
::nppts, inputlist::nptrj, inputlist::nquad, inputlist::ntor, inputlist::ntoraxis, inputlist::nvol, inputlist::odetol, inputlist↩
::oita, constants::one, inputlist::opsilon, fileunits::ounit, inputlist::pcondense, inputlist::pflux, inputlist::phiedge,
constants::pi2, inputlist::pl, inputlist::ppts, inputlist::pr, inputlist::pressure, inputlist::pscale, inputlist::ql, inputlist↩
::qr, inputlist::rac, inputlist::ras, inputlist::rbc, inputlist::rbs, regumm, inputlist::rp, inputlist::rpol, inputlist::rq, rscale,
inputlist::rtor, inputlist::rwc, inputlist::rws, numerical::small, smpol, sntor, numerical::sqrtmachprec, inputlist::tflux,
inputlist::upsilon, inputlist::vcasingeps, inputlist::vcasingits, inputlist::vcasingper, inputlist::vcasingtol, inputlist↩
::vnc, inputlist::vns, numerical::vsmall, inputlist::wmacros, inputlist::wpoloidal, inputlist::wreadin, inputlist::wwrtend,
yesstellsym, inputlist::zac, inputlist::zas, inputlist::zbc, inputlist::zbs, constants::zero, inputlist::zwc, and inputlist↩
::zws.

Referenced by preset(), and xspech().

Here is the call graph for this function:

Here is the caller graph for this function:



## 9.2   constants Module Reference

some constants used throughout the code

**Variables**

- real, parameter zero = 0.0

  *0*
- real, parameter one = 1.0

  *1*
- real, parameter two = 2.0

  *2*
- real, parameter three = 3.0

  *3*
- real, parameter four = 4.0

  *4*
- real, parameter five = 5.0

  *5*
- real, parameter six = 6.0

  *6*
- real, parameter seven = 7.0

  *7*
- real, parameter eight = 8.0

  *8*
- real, parameter nine = 9.0

  *9*
- real, parameter ten = 10.0

  *10*
- real, parameter eleven = 11.0

  *11*
- real, parameter twelve = 12.0

  *12*
- real, parameter hundred = 100.0

  *100*
- real, parameter thousand = 1000.0

  *1000*
- real, parameter half = one / two

*1/2*

- real, parameter third = one / three

    *1/3*

- real, parameter quart = one / four

    *1/4*

- real, parameter fifth = one / five

    *1/5*

- real, parameter sixth = one / six

    *1/6*

- real, parameter pi2 = 6.28318530717958623

    $2\pi$

- real, parameter pi = pi2 / two

    $\pi$

- real, parameter mu0 = 2.0E-07 ∗ pi2

    $4\pi \cdot 10^{-7}$

- real, parameter goldenmean = 1.618033988749895

    *golden mean* = $(1 + \sqrt{5})/2$ ;

- real, parameter version = 2.00

    *version of SPEC*

### 9.2.1 Detailed Description

some constants used throughout the code

## 9.3 cputiming Module Reference

timing variables

**Variables**

- real treadin = 0.0

    *timing of readin()*

- real twrtend = 0.0

    *timing of wrtend()*

### 9.3.1 Detailed Description

timing variables

## 9.4 fftw_interface Module Reference

Interface to FFTW library.

**Variables**

- type(c_ptr) planf

    *FFTW-related (?)*
- type(c_ptr) planb

    *FFTW-related (?)*
- complex(c_double_complex), dimension(:,:,:), allocatable cplxin

    *FFTW-related (?)*
- complex(c_double_complex), dimension(:,:,:), allocatable cplxout

    *FFTW-related (?)*

### 9.4.1   Detailed Description

Interface to FFTW library.

## 9.5   fileunits Module Reference

central definition of file units to avoid conflicts

**Variables**

- integer iunit = 10

    *input; used in global/readin:ext.sp, global/wrtend:ext.sp.end*
- integer ounit = 6

    *screen output;*
- integer gunit = 13

    *wall geometry; used in wa00aa*
- integer aunit = 11

    *vector potential; used in ra00aa:.ext.AtAzmn;*
- integer dunit = 12

    *derivative matrix; used in newton:.ext.GF;*
- integer hunit = 14

    *eigenvalues of Hessian; under re-construction;*
- integer munit = 14

    *matrix elements of Hessian;*
- integer lunit = 20

    *local unit; used in lunit+myid: pp00aa:.ext.poincare,.ext.transform;*
- integer vunit = 15

    *for examination of adaptive quadrature; used in casing:.ext.vcint;*

### 9.5.1   Detailed Description

central definition of file units to avoid conflicts

## 9.6   newtontime Module Reference

timing of Newton iterations

**Variables**

- integer nfcalls

    *number of calls to get function values (?)*
- integer ndcalls

    *number of calls to get derivative values (?)*
- real lastcpu

    *last CPU that called this (?)*

### 9.6.1 Detailed Description

timing of Newton iterations

## 9.7 numerical Module Reference

platform-dependant numerical resolution

**Functions/Subroutines**

- real function myprec ()

    *Duplicates NAG routine* `X02AJF` *(machine precision)*

**Variables**

- real machprec

    *machine precision according to NAG-like routine myprec()*
- real vsmall

    *very small number*
- real small

    *small number*
- real sqrtmachprec

    *square root of machine precision*
- real, parameter logtolerance = 1.0e-32

    *this is used to avoid taking alog10(zero); see e.g. dforce()*

### 9.7.1 Detailed Description

platform-dependant numerical resolution

### 9.7.2 Function/Subroutine Documentation

**9.7.2.1   myprec()**   `real function numerical::myprec`

Duplicates NAG routine `X02AJF` (machine precision)

JAB; 27 Jul 17 I suggest that this be removed; SRH: 27 Feb 18;

**Returns**

   machine precision

References small.

Referenced by allglobal::readin().

Here is the caller graph for this function:
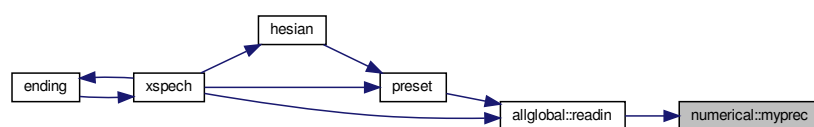


## 9.8   typedefns Module Reference

type definitions for custom datatypes

**Data Types**

- type matrixlu
- type subgrid

   *used for quantities which have different resolutions in different volumes, e.g. the vector potential More...*

### 9.8.1   Detailed Description

type definitions for custom datatypes

### 9.8.2   Data Type Documentation

**Class Members**

| real, dimension(:,:), allocatable | mat | |
|---|---|---|
| integer, dimension(:), allocatable | ipivot | |

#### 9.8.2.1   type typedefns::matrixlu

**9.8.2.2 type typedefns::subgrid** used for quantities which have different resolutions in different volumes, e.g. the vector potential

**Class Members**

| real, dimension(:), allocatable | s | coefficients |
|---|---|---|
| integer, dimension(:), allocatable | i | indices |

# 10  Data Type Documentation

## 10.1  intghs_module::intghs_workspace Type Reference

**Public Attributes**

- real, dimension(:,:), allocatable **efmn**
- real, dimension(:,:), allocatable **ofmn**
- real, dimension(:,:), allocatable **cfmn**
- real, dimension(:,:), allocatable **sfmn**
- real, dimension(:,:), allocatable **evmn**
- real, dimension(:,:), allocatable **odmn**
- real, dimension(:,:), allocatable **ijreal**
- real, dimension(:,:), allocatable **jireal**
- real, dimension(:,:), allocatable **jkreal**
- real, dimension(:,:), allocatable **kjreal**
- real, dimension(:,:,:), allocatable **bloweremn**
- real, dimension(:,:,:), allocatable **bloweromn**
- real, dimension(:,:,:), allocatable **gbupper**
- real, dimension(:,:,:), allocatable **blower**
- real, dimension(:,:,:,:), allocatable **basis**

### 10.1.1  Member Data Documentation

**10.1.1.1  efmn** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::efmn`

**10.1.1.2  ofmn** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::ofmn`

**10.1.1.3  cfmn** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::cfmn`

**10.1.1.4 sfmn** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::sfmn`

**10.1.1.5 evmn** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::evmn`

**10.1.1.6 odmn** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::odmn`

**10.1.1.7 ijreal** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::ijreal`

**10.1.1.8 jireal** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::jireal`

**10.1.1.9 jkreal** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::jkreal`

**10.1.1.10 kjreal** `real, dimension(:,:), allocatable intghs_module::intghs_workspace::kjreal`

**10.1.1.11 bloweremn** `real, dimension(:,:,:), allocatable intghs_module::intghs_workspace↩ ::bloweremn`

**10.1.1.12 bloweromn** `real, dimension(:,:,:), allocatable intghs_module::intghs_workspace↩ ::bloweromn`

**10.1.1.13 gbupper** `real, dimension(:,:,:), allocatable intghs_module::intghs_workspace::gbupper`

**10.1.1.14 blower** `real, dimension(:,:,:), allocatable intghs_module::intghs_workspace::blower`

**10.1.1.15  basis** `real, dimension(:,:,:,:), allocatable intghs_module::intghs_workspace::basis`

The documentation for this type was generated from the following file:

- [intghs.f90](intghs.f90)

# 11  File Documentation

## 11.1  basefn.f90 File Reference

Polynomials evaluation.

**Functions/Subroutines**

- subroutine [get_cheby](get_cheby) (lss, lrad, cheby)

    *Get the Chebyshev polynomials with zeroth, first derivatives.*
- subroutine [get_cheby_d2](get_cheby_d2) (lss, lrad, cheby)

    *Get the Chebyshev polynomials with zeroth, first and second derivatives.*
- subroutine [get_zernike](get_zernike) (r, lrad, mpol, zernike)

    *Get the Zernike polynomials with zeroth, first derivatives.*
- subroutine [get_zernike_d2](get_zernike_d2) (r, lrad, mpol, zernike)

    *Get the Zernike polynomials with zeroth, first, second derivatives.*
- subroutine [get_zernike_rm](get_zernike_rm) (r, lrad, mpol, zernike)

    *Get the Zernike polynomials $Z(n,m,r)/r^{\wedge}m$.*

### 11.1.1  Detailed Description

Polynomials evaluation.

### 11.1.2  Function/Subroutine Documentation

**11.1.2.1  get_cheby()** `subroutine get_cheby (`
         `real, intent(in) lss,`
         `integer, intent(in) lrad,`
         `real, dimension(0:lrad,0:1), intent(inout) cheby )`

Get the Chebyshev polynomials with zeroth, first derivatives.

**Parameters**

| | | |
|---|---|---|
| `in` | *lss* | coordinate input lss |
| `in` | *lrad* | radial resolution |
| `out` | *cheby* | the value, first derivative of Chebyshev polynomial |

References constants::one, constants::two, and constants::zero.

Referenced by bfield(), intghs(), ma00aa(), preset(), and spsint().

Here is the caller graph for this function:
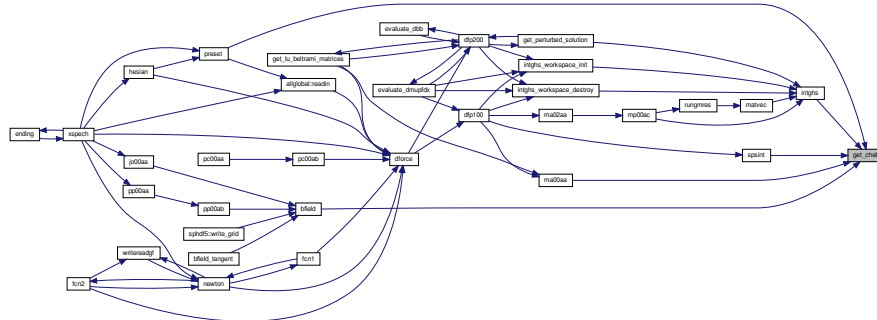


**11.1.2.2 get_cheby_d2()** `subroutine get_cheby_d2 (`
    `real, intent(in) lss,`
    `integer, intent(in) lrad,`
    `real, dimension(0:lrad,0:2), intent(inout) cheby )`

Get the Chebyshev polynomials with zeroth, first and second derivatives.

**Parameters**

| in | *lss* | coordinate input lss |
|---|---|---|
| in | *lrad* | radial resolution |
| out | *cheby* | the value, first and second derivative of Chebyshev polynomial |

References constants::one, constants::two, and constants::zero.

Referenced by bfield_tangent(), and jo00aa().

Here is the caller graph for this function:
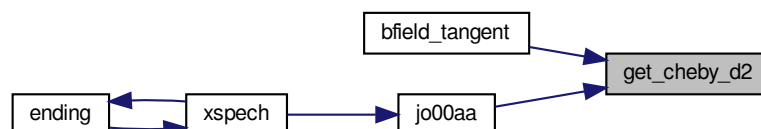
### 11.1.2.3 get_zernike()

```
subroutine get_zernike (
        real, intent(in) r,
        integer, intent(in) lrad,
        integer, intent(in) mpol,
        real, dimension(0:lrad,0:mpol,0:1), intent(inout) zernike )
```

Get the Zernike polynomials with zeroth, first derivatives.

**Parameters**

| in  | *r*      | coordinate input                                    |
|-----|----------|-----------------------------------------------------|
| in  | *lrad*   | radial resolution                                   |
| in  | *mpol*   | poloidal resolution                                 |
| out | *zernike* | the value, first derivative of Zernike polynomial  |

References constants::one, constants::two, and constants::zero.

Referenced by bfield(), intghs(), ma00aa(), preset(), and spsint().

Here is the caller graph for this function:



### 11.1.2.4 get_zernike_d2()

```
subroutine get_zernike_d2 (
        real, intent(in) r,
        integer, intent(in) lrad,
        integer, intent(in) mpol,
        real, dimension(0:lrad,0:mpol,0:2), intent(inout) zernike )
```

Get the Zernike polynomials with zeroth, first, second derivatives.

**Parameters**

| in  | *r*      | coordinate input                                           |
|-----|----------|------------------------------------------------------------|
| in  | *lrad*   | radial resolution                                          |
| in  | *mpol*   | poloidal resolution                                        |
| out | *zernike* | the value, first/second derivative of Zernike polynomial  |

References constants::one, constants::two, and constants::zero.

Referenced by bfield_tangent(), and jo00aa().

Here is the caller graph for this function:
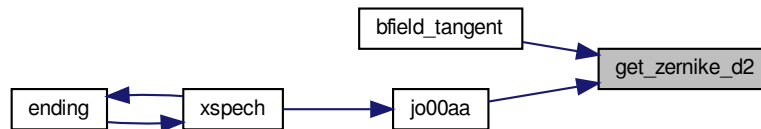


### 11.1.2.5 get_zernike_rm() subroutine get_zernike_rm (
        real, intent(in) *r,*
        integer, intent(in) *lrad,*
        integer, intent(in) *mpol,*
        real, dimension(0:lrad,0:mpol), intent(inout) *zernike* )

Get the Zernike polynomials $Z(n,m,r)/r^{\wedge}m$.

**Parameters**

| in | *r* | coordinate input |
|---|---|---|
| in | *lrad* | radial resolution |
| in | *mpol* | poloidal resolution |
| out | *zernike* | the value |

References constants::one, constants::two, and constants::zero.

Referenced by preset().

Here is the caller graph for this function:



## 11.2 bfield.f90 File Reference

Returns $\dot{s} \equiv B^s/B^\zeta$ and $\dot{\theta} \equiv B^\theta/B^\zeta$.

**Functions/Subroutines**

- subroutine bfield (zeta, st, Bst)

  *Compute the magnetic field.*

- subroutine bfield_tangent (zeta, st, Bst)

  *compute the tangential magnetic field*

### 11.2.1 Detailed Description

Returns $\dot{s} \equiv B^s/B^\zeta$ and $\dot{\theta} \equiv B^\theta/B^\zeta$.

### 11.2.2 Function/Subroutine Documentation

#### 11.2.2.1 bfield_tangent()

```
subroutine bfield_tangent (
        real, intent(in) zeta,
        real, dimension(1:6), intent(in) st,
        real, dimension(1:6), intent(out) Bst )
```

compute the tangential magnetic field

**Parameters**

| in  | *zeta* | toroidal angle |
|-----|--------|----------------|
| in  | *st*   | radial(s) and poloidal(theta) positions |
| out | *Bst*  | tangential magnetic field |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, bfield(), allglobal::cpus, allglobal::gbzeta, get_cheby_d2(), get_zernike_d2(), constants::half, allglobal::halfmm, allglobal::im, allglobal::in, allglobal↩
::ivol, allglobal::lcoordinatesingularity, inputlist::lrad, allglobal::mn, inputlist::mpol, allglobal::mvol, allglobal↩
::myid, allglobal::ncpu, allglobal::node, allglobal::notstellsym, constants::one, fileunits::ounit, allglobal::regumm,
numerical::small, constants::two, numerical::vsmall, inputlist::wmacros, and constants::zero.

Here is the call graph for this function:
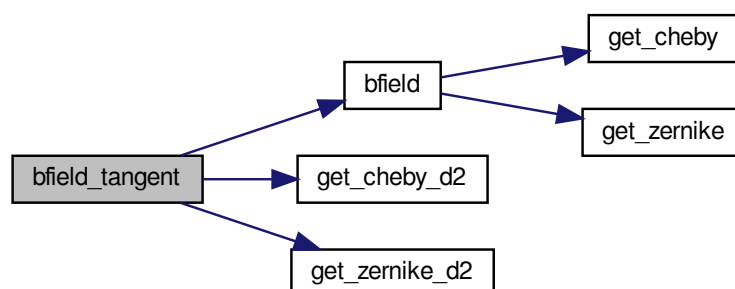
## 11.3 bnorml.f90 File Reference

Computes $\mathbf{B}_{Plasma} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta$ on the computational boundary, $\partial \mathcal{D}$.

**Functions/Subroutines**

- subroutine bnorml (mn, Ntz, efmn, ofmn)

    *Computes $\mathbf{B}_{Plasma} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta$ on the computational boundary, $\partial \mathcal{D}$.*

### 11.3.1 Detailed Description

Computes $\mathbf{B}_{Plasma} \cdot \mathbf{e}_\theta \times \mathbf{e}_\zeta$ on the computational boundary, $\partial \mathcal{D}$.

## 11.4 brcast.f90 File Reference

Broadcasts Beltrami fields, profiles, . . .

**Functions/Subroutines**

- subroutine brcast (lvol)

    *Broadcasts Beltrami fields, profiles, . . .*

### 11.4.1 Detailed Description

Broadcasts Beltrami fields, profiles, . . .

## 11.5 casing.f90 File Reference

Constructs the field created by the plasma currents, at an arbitrary, external location using virtual casing.

**Functions/Subroutines**

- subroutine casing (teta, zeta, gBn, icasing)

    *Constructs the field created by the plasma currents, at an arbitrary, external location using virtual casing.*
- subroutine dvcfield (Ndim, tz, Nfun, vcintegrand)

    *Differential virtual casing integrand.*

### 11.5.1 Detailed Description

Constructs the field created by the plasma currents, at an arbitrary, external location using virtual casing.

## 11.6 coords.f90 File Reference

Calculates coordinates, $\mathbf{x}(s,\theta,\zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, using FFTs.

**Functions/Subroutines**

- subroutine coords (lvol, lss, Lcurvature, Ntz, mn)

  *Calculates coordinates, $\mathbf{x}(s,\theta,\zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, using FFTs.*

### 11.6.1 Detailed Description

Calculates coordinates, $\mathbf{x}(s,\theta,\zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, using FFTs.

## 11.7 curent.f90 File Reference

Computes the plasma current, $I \equiv \int B_\theta\,d\theta$, and the "linking" current, $G \equiv \int B_\zeta\,d\zeta$.

**Functions/Subroutines**

- subroutine curent (lvol, mn, Nt, Nz, iflag, ldItGp)

  *Computes the plasma current, $I \equiv \int B_\theta\,d\theta$, and the "linking" current, $G \equiv \int B_\zeta\,d\zeta$.*

### 11.7.1 Detailed Description

Computes the plasma current, $I \equiv \int B_\theta\,d\theta$, and the "linking" current, $G \equiv \int B_\zeta\,d\zeta$.

## 11.8 df00ab.f90 File Reference

Evaluates volume integrals, and their derivatives w.r.t. interface geometry, using "packed" format.

**Functions/Subroutines**

- subroutine df00ab (pNN, xi, Fxi, DFxi, Ldfjac, iflag)

  *Evaluates volume integrals, and their derivatives w.r.t. interface geometry, using "packed" format.*

### 11.8.1 Detailed Description

Evaluates volume integrals, and their derivatives w.r.t. interface geometry, using "packed" format.

## 11.9 dforce.f90 File Reference

Calculates $\mathbf{F}(\mathbf{x})$, where $\mathbf{x} \equiv \{\text{geometry}\} \equiv \{R_{i,v}, Z_{i,v}\}$ and $\mathbf{F} \equiv [[p+B^2/2]] + \{\text{spectral constraints}\}$, and $\nabla\mathbf{F}$.

**Functions/Subroutines**

- subroutine dforce (NGdof, position, force, LComputeDerivatives, LComputeAxis)

  *Calculates* $\mathbf{F}(\mathbf{x})$*, where* $\mathbf{x} \equiv \{geometry\} \equiv \{R_{i,v}, Z_{i,v}\}$ *and* $\mathbf{F} \equiv [[p + B^2/2]] + \{spectral\ constraints\}$*, and* $\nabla\mathbf{F}$*.*

### 11.9.1 Detailed Description

Calculates $\mathbf{F}(\mathbf{x})$, where $\mathbf{x} \equiv \{\text{geometry}\} \equiv \{R_{i,v}, Z_{i,v}\}$ and $\mathbf{F} \equiv [[p + B^2/2]] + \{\text{spectral constraints}\}$, and $\nabla\mathbf{F}$.

## 11.10 dfp100.f90 File Reference

Split the work between MPI nodes and evaluate the global constraint.

**Functions/Subroutines**

- subroutine dfp100 (Ndofgl, x, Fvec, LComputeDerivatives)

  *Split the work between MPI nodes and evaluate the global constraint.*

### 11.10.1 Detailed Description

Split the work between MPI nodes and evaluate the global constraint.

### 11.10.2 Function/Subroutine Documentation

#### 11.10.2.1 dfp100()

```
subroutine dfp100 (
        integer Ndofgl,
        real, dimension(1:mvol-1) x,
        real, dimension(1:ndofgl) Fvec,
        logical LComputeDerivatives )
```

Split the work between MPI nodes and evaluate the global constraint.

**Parameters**

| Ndofgl | |
|---|---|
| x | |
| Fvec | |
| LComputeDerivatives | |

References allocate_beltrami_matrices(), allocate_geometry_matrices(), compute_guvijsave(), allglobal←
::cpus, curent(), inputlist::curpol, allglobal::dbdx, allglobal::ddttcc, allglobal::ddttcs, allglobal::ddttsc, allglobal←
::ddttss, allglobal::ddtzcc, allglobal::ddtzcs, allglobal::ddtzsc, allglobal::ddtzss, allglobal::ddzzcc, allglobal←
::ddzzcs, allglobal::ddzzsc, allglobal::ddzzss, deallocate_beltrami_matrices(), deallocate_geometry_matrices(),

allglobal::dma, allglobal::dmb, allglobal::dmd, allglobal::dmg, allglobal::dpflux, allglobal::dtoocc, allglobal::dtoocs, allglobal::dtoosc, allglobal::dtooss, allglobal::guvijsave, constants::half, allglobal::iconstraintok, inputlist::igeometry, allglobal::imagneticok, intghs_workspace_destroy(), intghs_workspace_init(), allglobal::ipdt, allglobal::ipdtdpf, allglobal::iquad, allglobal::ismyvolume(), allglobal::ismyvolumevalue, inputlist::isurf, allglobal::izbs, lbpol(), inputlist↩ ::lconstraint, allglobal::lcoordinatesingularity, inputlist::lfreebound, allglobal::liluprecond, allglobal::localconstraint, allglobal::lplasmaregion, inputlist::lrad, allglobal::lsavedguvij, allglobal::lvacuumregion, ma00aa(), ma02aa(), ma-trix(), allglobal::mbpsi, allglobal::mn, constants::mu0, allglobal::mvol, allglobal::myid, allglobal::nadof, allglobal↩ ::ncpu, allglobal::notmatrixfree, allglobal::nt, inputlist::nvol, allglobal::nz, constants::one, fileunits::ounit, constants↩ ::pi, constants::pi2, allglobal::solution, spsint(), spsmat(), allglobal::tdstcc, allglobal::tdstcs, allglobal::tdstsc, allglobal::tdstss, allglobal::tdszcc, allglobal::tdszcs, allglobal::tdszsc, allglobal::tdszss, allglobal::ttsscc, allglobal↩ ::ttsscs, allglobal::ttsssc, allglobal::ttssss, constants::two, volume(), allglobal::whichcpuid(), inputlist::wmacros, allglobal::xoffset, and constants::zero.

Referenced by dforce(), and evaluate_dmupfdx().

Here is the call graph for this function:
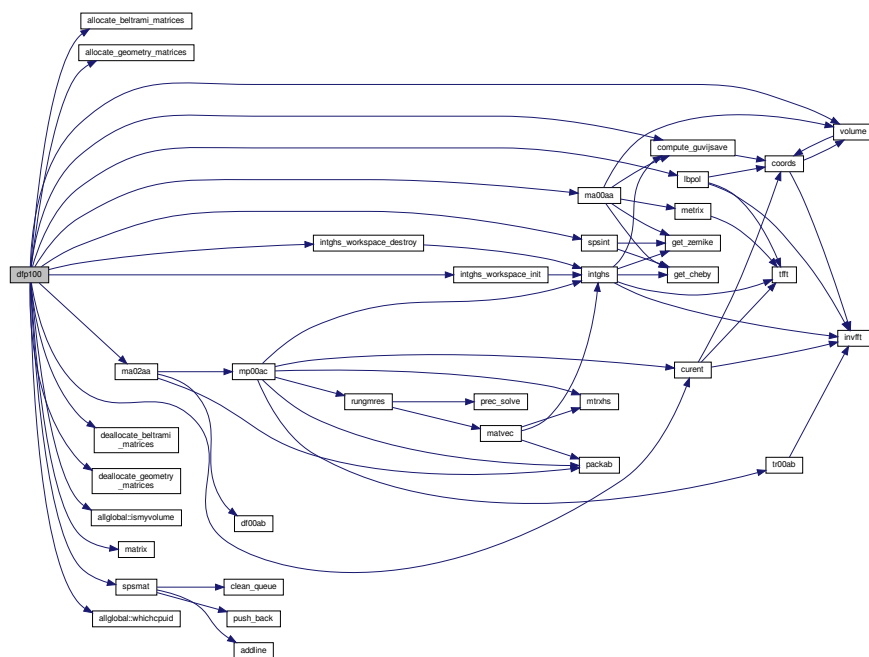
Here is the caller graph for this function:
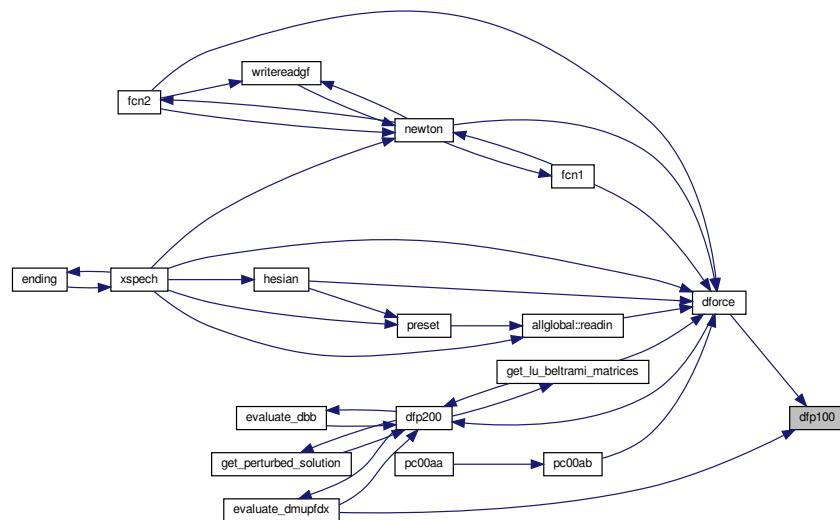


## 11.11 dfp200.f90 File Reference

Given the field consistent with the constraints and the geometry, computes local quantites related to the force evaluation.

### Functions/Subroutines

- subroutine dfp200 (LcomputeDerivatives, vvol)

  *Given the field consistent with the constraints and the geometry, computes local quantites related to the force evaluation.*

- subroutine get_lu_beltrami_matrices (vvol, oBI, NN)

  *get LU Beltrami matrices*

- subroutine get_perturbed_solution (vvol, oBI, NN)

  *This routine evaluates the value of the magnetic field once the interface is perturbed using matrix perturbation theory.*

- subroutine evaluate_dmupfdx (innout, idof, ii, issym, irz)

  *Evaluate mu and psip derivatives and store them in dmupfdx.*

- subroutine evaluate_dbb (lvol, idof, innout, issym, irz, ii, dBB, XX, YY, length, dRR, dZZ, dII, dLL, dPP, Ntz)

  *Evaluate the derivative of the square of the magnetic field modulus. Add spectral constraint derivatives if required.*

### 11.11.1 Detailed Description

Given the field consistent with the constraints and the geometry, computes local quantites related to the force evaluation.

### 11.11.2 Function/Subroutine Documentation

**11.11.2.1 dfp200()** `subroutine dfp200 (`
`        logical, intent(in) LcomputeDerivatives,`
`        integer vvol )`

Given the field consistent with the constraints and the geometry, computes local quantites related to the force evaluation.

**Parameters**

| LcomputeDerivatives | |
|---|---|
| vvol | |

References inputlist::adiabatic, allocate_beltrami_matrices(), allocate_geometry_matrices(), allglobal::cpus, deallocate_beltrami_matrices(), deallocate_geometry_matrices(), inputlist::epsilon, evaluate_dbb(), evaluate↩
_dmupfdx(), inputlist::ext, inputlist::gamma, get_lu_beltrami_matrices(), get_perturbed_solution(), constants↩
::half, inputlist::igeometry, intghs_workspace_destroy(), intghs_workspace_init(), allglobal::iquad, inputlist↩
::lcheck, inputlist::lconstraint, allglobal::lcoordinatesingularity, inputlist::lextrap, inputlist::lfindzero, lforce(), inputlist↩
::lfreebound, allglobal::lplasmaregion, inputlist::lrad, allglobal::lvacuumregion, inputlist::mpol, inputlist::mu, allglobal::mvol, allglobal::myid, allglobal::ncpu, inputlist::ntor, inputlist::nvol, constants::one, fileunits::ounit, packab(), inputlist::pscale, numerical::small, inputlist::tflux, constants::two, volume(), inputlist::wmacros, and constants::zero.

Referenced by dforce(), evaluate_dbb(), evaluate_dmupfdx(), get_lu_beltrami_matrices(), and get_perturbed_↩
solution().

Here is the call graph for this function:

Here is the caller graph for this function:
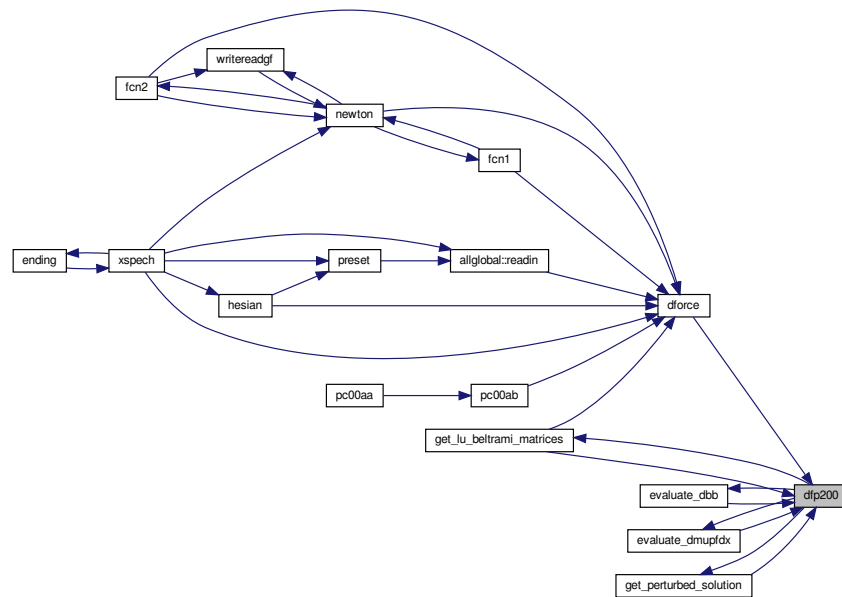


**11.11.2.2   get_lu_beltrami_matrices()**   `subroutine get_lu_beltrami_matrices (`
            `integer, intent(in) ` *vvol,*
            `type(`[matrixlu]`), intent(inout) ` *oBI,*
            `integer, intent(in) ` *NN* ` )`

get LU Beltrami matrices

**Parameters**

| vvol | |
|------|---|
| oBI  | |
| NN   | |

References allglobal::cpus, allglobal::dbdx, dforce(), dfp200(), allglobal::dma, allglobal::dmb, allglobal::dmd, allglobal::dmg, constants::half, allglobal::iquad, inputlist::lconstraint, allglobal::lcoordinatesingularity, allglobal↩
::lplasmaregion, inputlist::lrad, allglobal::lsavedguvij, allglobal::lvacuumregion, ma00aa(), matrix(), allglobal::mn, allglobal::mne, inputlist::mu, allglobal::myid, allglobal::ncpu, allglobal::nt, allglobal::nz, constants::one, fileunits↩
::ounit, allglobal::solution, constants::two, inputlist::wmacros, and constants::zero.

Referenced by dfp200().

Here is the call graph for this function:



Here is the caller graph for this function:



**11.11.2.3 get_perturbed_solution()** `subroutine get_perturbed_solution (`
        `integer, intent(in) vvol,`
        `type(matrixlu), intent(inout) oBI,`
        `integer, intent(in) NN )`

This routine evaluates the value of the magnetic field once the interface is perturbed using matrix perturbation theory.

**Parameters**

| vvol | |
|------|--|
| oBI | |
| NN | |

References allglobal::cpus, allglobal::dbdx, dfp200(), allglobal::dma, allglobal::dmb, allglobal::dmd, allglobal←
::dmg, allglobal::dpflux, allglobal::dtflux, constants::half, intghs(), allglobal::iquad, inputlist::lconstraint, inputlist::lrad,
allglobal::mn, mtrxhs(), inputlist::mu, allglobal::myid, allglobal::nadof, allglobal::ncpu, constants::one, fileunits::ounit,
packab(), allglobal::solution, constants::two, inputlist::wmacros, and constants::zero.

Referenced by dfp200().

Here is the call graph for this function:
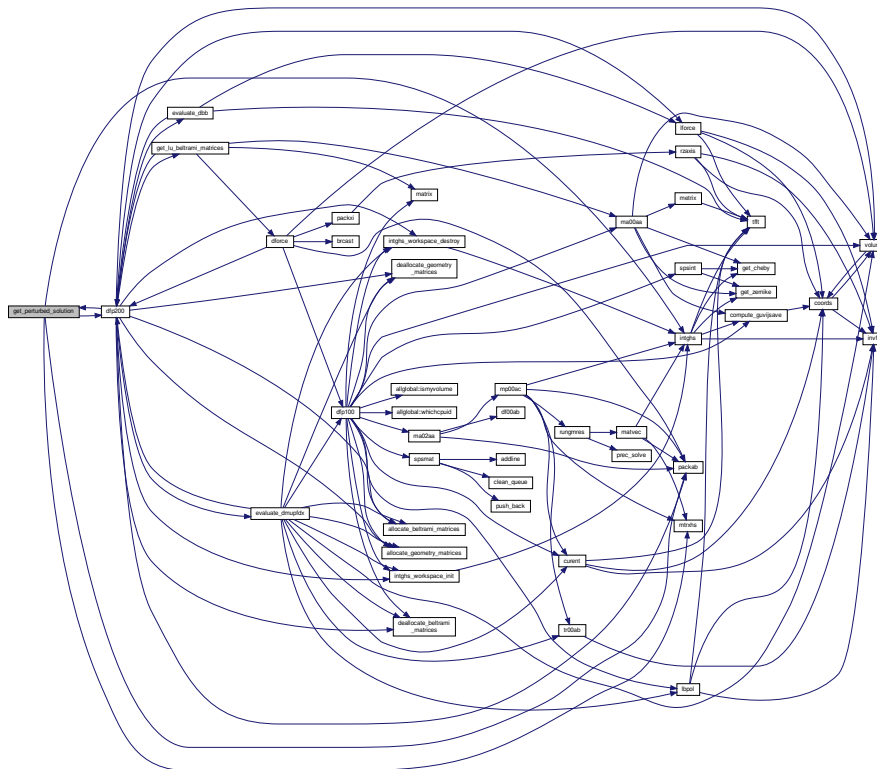
Here is the caller graph for this function:
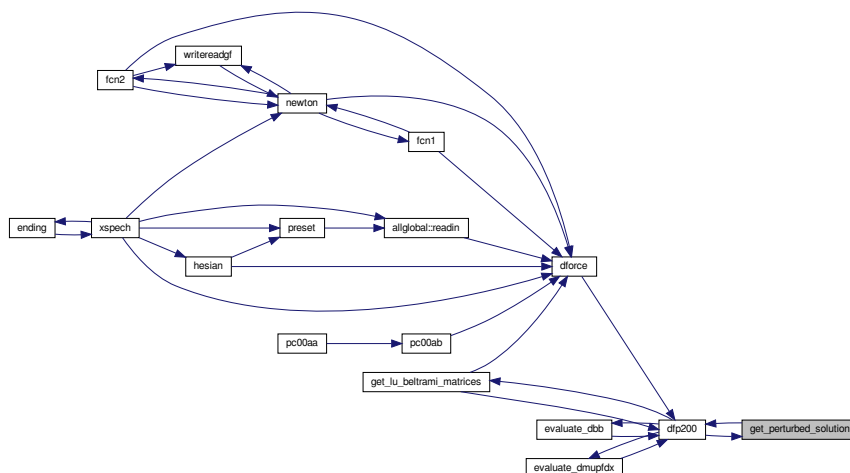


### 11.11.2.4  evaluate_dmupfdx()  `subroutine evaluate_dmupfdx (`

```
        integer innout,
        integer idof,
        integer ii,
        integer issym,
        integer irz )
```

Evaluate mu and psip derivatives and store them in dmupfdx.

**Parameters**

| | |
|---|---|
| *innout* | |
| *idof* | |
| *ii* | |
| *issym* | |
| *irz* | |

References allocate_beltrami_matrices(), allocate_geometry_matrices(), allglobal::cpus, curent(), deallocate_←
beltrami_matrices(), deallocate_geometry_matrices(), dfp100(), dfp200(), inputlist::drz, constants::half, inputlist←
::igeometry, intghs_workspace_destroy(), intghs_workspace_init(), allglobal::iquad, allglobal::irbc, allglobal←
::irbs, allglobal::izbc, allglobal::izbs, lbpol(), inputlist::lcheck, inputlist::lconstraint, allglobal::lcoordinatesingularity,
inputlist::lfreebound, allglobal::lplasmaregion, inputlist::lrad, allglobal::lvacuumregion, inputlist::mu, inputlist←
::mupftol, allglobal::mvol, allglobal::myid, allglobal::ncpu, allglobal::ngdof, inputlist::nvol, constants::one, fileunits←
::ounit, numerical::small, tr00ab(), constants::two, volume(), inputlist::wmacros, and constants::zero.

Referenced by dfp200().

Here is the call graph for this function:



Here is the caller graph for this function:



**11.11.2.5   evaluate_dbb()**   subroutine evaluate_dbb (
        integer *lvol,*

```
            integer idof,
            integer innout,
            integer issym,
            integer irz,
            integer ii,
            real, dimension(1:ntz,-1:2) dBB,
            real, dimension(1:ntz) XX,
            real, dimension(1:ntz) YY,
            real, dimension(1:ntz) length,
            real, dimension(1:ntz,-1:2) dRR,
            real, dimension(1:ntz,-1:2) dZZ,
            real, dimension(1:ntz) dII,
            real, dimension(1:ntz) dLL,
            real, dimension(1:ntz) dPP,
            integer Ntz )
```

Evaluate the derivative of the square of the magnetic field modulus. Add spectral constraint derivatives if required.

**Parameters**

| | |
|---|---|
| *lvol* | |
| *idof* | |
| *innout* | |
| *issym* | |
| *irz* | |
| *ii* | |
| *dBB* | |
| *XX* | |
| *YY* | |
| *length* | |
| *dRR* | |
| *dZZ* | |
| *dII* | |
| *dLL* | |
| *dPP* | |
| *Ntz* | |

References inputlist::adiabatic, allglobal::cpus, dfp200(), allglobal::dpflux, inputlist::drz, inputlist::epsilon, inputlist↩
::ext, inputlist::gamma, constants::half, inputlist::igeometry, allglobal::irbc, allglobal::irbs, allglobal::izbc, allglobal↩
::izbs, inputlist::lcheck, inputlist::lconstraint, allglobal::lcoordinatesingularity, lforce(), allglobal::localconstraint,
allglobal::lplasmaregion, inputlist::lrad, allglobal::lvacuumregion, allglobal::mvol, allglobal::myid, allglobal::ncpu,
inputlist::ntor, inputlist::nvol, constants::one, fileunits::ounit, inputlist::pscale, numerical::small, tfft(), constants::two,
inputlist::wmacros, and constants::zero.

Referenced by dfp200().

Here is the call graph for this function:
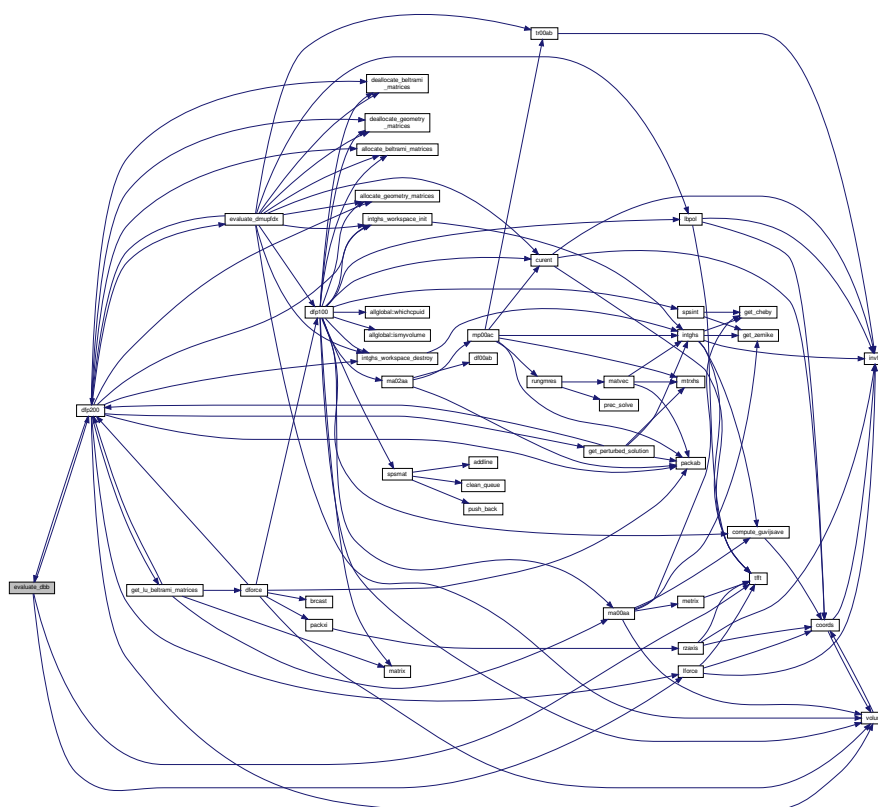


Here is the caller graph for this function:
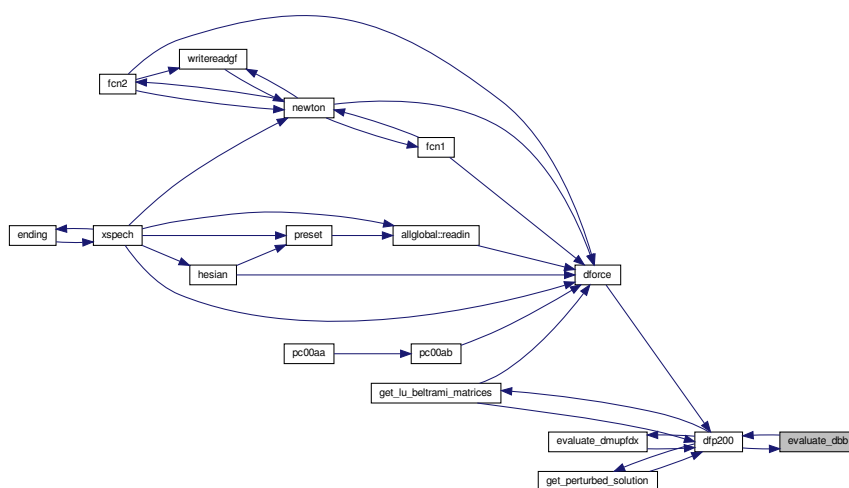


## 11.12 global.f90 File Reference

Defines input namelists and global variables, and opens some output files.

**Data Types**

- type typedefns::subgrid

  *used for quantities which have different resolutions in different volumes, e.g. the vector potential More...*
- type typedefns::matrixlu
- type allglobal::derivative

  $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ *(?) More...*

**Modules**

- module constants

  *some constants used throughout the code*
- module numerical

  *platform-dependant numerical resolution*
- module fileunits

  *central definition of file units to avoid conflicts*
- module cputiming

  *timing variables*
- module typedefns

  *type definitions for custom datatypes*
- module allglobal

  *global variable storage used as "workspace" throughout the code*
- module fftw_interface

  *Interface to FFTW library.*

**Functions/Subroutines**

- real function numerical::myprec ()

  *Duplicates NAG routine $X02AJF$ (machine precision)*
- subroutine **allglobal::build_vector_potential** (lvol, iocons, aderiv, tderiv)
- subroutine allglobal::readin

  *The master node reads the input namelist and sets various internal variables. The relevant quantities are then broadcast.*
- subroutine allglobal::wrtend

  *The restart file is written.*
- subroutine allglobal::ismyvolume (vvol)

  *Check if volume vvol is associated to the corresponding MPI node.*
- subroutine allglobal::whichcpuid (vvol, cpu_id)

  *Returns which MPI node is associated to a given volume.*

**Variables**

- real, parameter constants::zero = 0.0

  *0*
- real, parameter constants::one = 1.0

  *1*
- real, parameter constants::two = 2.0

  *2*
- real, parameter constants::three = 3.0

  *3*

- real, parameter [constants::four](#) = 4.0

  *4*
- real, parameter [constants::five](#) = 5.0

  *5*
- real, parameter [constants::six](#) = 6.0

  *6*
- real, parameter [constants::seven](#) = 7.0

  *7*
- real, parameter [constants::eight](#) = 8.0

  *8*
- real, parameter [constants::nine](#) = 9.0

  *9*
- real, parameter [constants::ten](#) = 10.0

  *10*
- real, parameter [constants::eleven](#) = 11.0

  *11*
- real, parameter [constants::twelve](#) = 12.0

  *12*
- real, parameter [constants::hundred](#) = 100.0

  *100*
- real, parameter [constants::thousand](#) = 1000.0

  *1000*
- real, parameter [constants::half](#) = one / two

  *1/2*
- real, parameter [constants::third](#) = one / three

  *1/3*
- real, parameter [constants::quart](#) = one / four

  *1/4*
- real, parameter [constants::fifth](#) = one / five

  *1/5*
- real, parameter [constants::sixth](#) = one / six

  *1/6*
- real, parameter [constants::pi2](#) = 6.28318530717958623

  $2\pi$
- real, parameter [constants::pi](#) = pi2 / two

  $\pi$
- real, parameter [constants::mu0](#) = 2.0E-07 $*$ pi2

  $4\pi \cdot 10^{-7}$
- real, parameter [constants::goldenmean](#) = 1.618033988749895

  *golden mean* $= (1 + \sqrt{5})/2$ *;*
- real, parameter [constants::version](#) = 2.00

  *version of SPEC*
- real [numerical::machprec](#)

  *machine precision according to NAG-like routine [myprec()](#)*
- real [numerical::vsmall](#)

  *very small number*
- real [numerical::small](#)

  *small number*
- real [numerical::sqrtmachprec](#)

  *square root of machine precision*
- real, parameter [numerical::logtolerance](#) = 1.0e-32

*this is used to avoid taking alog10(zero); see e.g. dforce()*
- integer fileunits::iunit = 10

  *input; used in global/readin:ext.sp, global/wrtend:ext.sp.end*
- integer fileunits::ounit = 6

  *screen output;*
- integer fileunits::gunit = 13

  *wall geometry; used in wa00aa*
- integer fileunits::aunit = 11

  *vector potential; used in ra00aa:.ext.AtAzmn;*
- integer fileunits::dunit = 12

  *derivative matrix; used in newton:.ext.GF;*
- integer fileunits::hunit = 14

  *eigenvalues of Hessian; under re-construction;*
- integer fileunits::munit = 14

  *matrix elements of Hessian;*
- integer fileunits::lunit = 20

  *local unit; used in lunit+myid: pp00aa:.ext.poincare,.ext.transform;*
- integer fileunits::vunit = 15

  *for examination of adaptive quadrature; used in casing:.ext.vcint;*
- real cputiming::treadin = 0.0

  *timing of readin()*
- real cputiming::twrtend = 0.0

  *timing of wrtend()*
- character inputlist::ext

  *The input file is , ext.sp , where* `ext*100` *or* `ext.sp*100` *is given as command line input.*
- integer, parameter inputlist::mnvol = 256

  *The maximum value of* `Nvol` *is* `MNvol=256.`
- integer, parameter inputlist::mmpol = 64

  *The maximum value of* `Mpol` *is* `MNpol=64.`
- integer, parameter inputlist::mntor = 64

  *The maximum value of* `Ntor` *is* `MNtor=64.`
- integer inputlist::igeometry = 3

  *selects Cartesian, cylindrical or toroidal geometry;*
- integer inputlist::istellsym = 1

  *stellarator symmetry is enforced if* `Istellsym==1`
- integer inputlist::lfreebound = 0

  *compute vacuum field surrounding plasma*
- real inputlist::phiedge = 1.0

  *total enclosed toroidal magnetic flux;*
- real inputlist::curtor = 0.0

  *total enclosed (toroidal) plasma current;*
- real inputlist::curpol = 0.0

  *total enclosed (poloidal) linking current;*
- real inputlist::gamma = 0.0

  *adiabatic index; cannot set* $|\gamma| = 1$
- integer inputlist::nfp = 1

  *field periodicity*
- integer inputlist::nvol = 1

  *number of volumes*
- integer inputlist::mpol = 0

  *number of poloidal Fourier harmonics*

- integer inputlist::ntor = 0

  *number of toroidal Fourier harmonics*
- integer, dimension(1:mnvol+1) inputlist::lrad = 4

  *Chebyshev resolution in each volume.*
- integer inputlist::lconstraint = -1

  *selects constraints; primarily used in ma02aa() and mp00ac().*
- real, dimension(1:mnvol+1) inputlist::tflux = 0.0

  *toroidal flux, $\psi_t$, enclosed by each interface*
- real, dimension(1:mnvol+1) inputlist::pflux = 0.0

  *poloidal flux, $\psi_p$, enclosed by each interface*
- real, dimension(1:mnvol) inputlist::helicity = 0.0

  *helicity, $\mathcal{K}$, in each volume, $\mathcal{V}_i$*
- real inputlist::pscale = 0.0

  *pressure scale factor*
- real, dimension(1:mnvol+1) inputlist::pressure = 0.0

  *pressure in each volume*
- integer inputlist::ladiabatic = 0

  *logical flag*
- real, dimension(1:mnvol+1) inputlist::adiabatic = 0.0

  *adiabatic constants in each volume*
- real, dimension(1:mnvol+1) inputlist::mu = 0.0

  *helicity-multiplier, $\mu$, in each volume*
- real, dimension(1:mnvol+1) inputlist::ivolume = 0.0

  *Toroidal current constraint normalized by $\mu_0$ ( $I_{volume} = \mu_0 \cdot [A]$), in each volume. This is a cumulative quantity: $I_{\mathcal{V},i} = \int_0^{\psi_{t,i}} \mathbf{J} \cdot \mathbf{dS}$. Physically, it represents the sum of all non-pressure driven currents.*
- real, dimension(1:mnvol) inputlist::isurf = 0.0

  *Toroidal current normalized by $\mu_0$ at each interface (cumulative). This is the sum of all pressure driven currents.*
- integer, dimension(0:mnvol) inputlist::pl = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::ql = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::pr = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::qr = 0

  *"inside" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- real, dimension(0:mnvol) inputlist::iota = 0.0

  *rotational-transform, $\iota$, on inner side of each interface*
- integer, dimension(0:mnvol) inputlist::lp = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::lq = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::rp = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- integer, dimension(0:mnvol) inputlist::rq = 0

  *"outer" interface rotational-transform is $\iota = (p_l + \gamma p_r)/(q_l + \gamma q_r)$, where $\gamma$ is the golden mean, $\gamma = (1 + \sqrt{5})/2$.*
- real, dimension(0:mnvol) inputlist::oita = 0.0

  *rotational-transform, $\iota$, on outer side of each interface*
- real inputlist::mupftol = 1.0e-14

  *accuracy to which $\mu$ and $\Delta\psi_p$ are required*
- integer inputlist::mupfits = 8

  *an upper limit on the transform/helicity constraint iterations;*

- real [inputlist::rpol](#) = 1.0

    *poloidal extent of slab (effective radius)*
- real [inputlist::rtor](#) = 1.0

    *toroidal extent of slab (effective radius)*
- integer [inputlist::lreflect](#) = 0

    *=1 reflect the upper and lower bound in slab, =0 do not reflect*
- real, dimension( 0:mntor) [inputlist::rac](#) = 0.0

    *stellarator symmetric coordinate axis;*
- real, dimension( 0:mntor) [inputlist::zas](#) = 0.0

    *stellarator symmetric coordinate axis;*
- real, dimension( 0:mntor) [inputlist::ras](#) = 0.0

    *non-stellarator symmetric coordinate axis;*
- real, dimension( 0:mntor) [inputlist::zac](#) = 0.0

    *non-stellarator symmetric coordinate axis;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::rbc](#) = 0.0

    *stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::zbs](#) = 0.0

    *stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::rbs](#) = 0.0

    *non-stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::zbc](#) = 0.0

    *non-stellarator symmetric boundary components;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::rwc](#) = 0.0

    *stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::zws](#) = 0.0

    *stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::rws](#) = 0.0

    *non-stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::zwc](#) = 0.0

    *non-stellarator symmetric boundary components of wall;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::vns](#) = 0.0

    *stellarator symmetric normal field at boundary; vacuum component;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::bns](#) = 0.0

    *stellarator symmetric normal field at boundary; plasma component;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::vnc](#) = 0.0

    *non-stellarator symmetric normal field at boundary; vacuum component;*
- real, dimension(-mntor:mntor,-mmpol:mmpol) [inputlist::bnc](#) = 0.0

    *non-stellarator symmetric normal field at boundary; plasma component;*
- integer [inputlist::linitialize](#) = 0

    *Used to initialize geometry using a regularization / extrapolation method.*
- integer [inputlist::lautoinitbn](#) = 1

    *Used to initialize $B_{ns}$ using an initial fixed-boundary calculation.*
- integer [inputlist::lzerovac](#) = 0

    *Used to adjust vacuum field to cancel plasma field on computational boundary.*
- integer [inputlist::ndiscrete](#) = 2

    *resolution of the real space grid on which fast Fourier transforms are performed is given by `Ndiscrete*Mpol*4`*
- integer [inputlist::nquad](#) = -1

    *Resolution of the Gaussian quadrature.*
- integer [inputlist::impol](#) = -4

    *Fourier resolution of straight-fieldline angle on interfaces.*
- integer [inputlist::intor](#) = -4

   *Fourier resolution of straight-fieldline angle on interfaces;.*

- integer inputlist::lsparse = 0

   *controls method used to solve for rotational-transform on interfaces*

- integer inputlist::lsvdiota = 0

   *controls method used to solve for rotational-transform on interfaces; only relevant if `Lsparse = 0`*

- integer inputlist::imethod = 3

   *controls iterative solution to sparse matrix arising in real-space transformation to the straight-fieldline angle; only relevant if `Lsparse.eq.2`;*

- integer inputlist::iorder = 2

   *controls real-space grid resolution for constructing the straight-fieldline angle; only relevant if `Lsparse>0`*

- integer inputlist::iprecon = 0

   *controls iterative solution to sparse matrix arising in real-space transformation to the straight-fieldline angle; only relevant if `Lsparse.eq.2`;*

- real inputlist::iotatol = -1.0

   *tolerance required for iterative construction of straight-fieldline angle; only relevant if `Lsparse.ge.2`*

- integer inputlist::lextrap = 0

   *geometry of innermost interface is defined by extrapolation*

- integer inputlist::mregular = -1

   *maximum regularization factor*

- integer inputlist::lrzaxis = 1

   *controls the guess of geometry axis in the innermost volume or initialization of interfaces*

- integer inputlist::ntoraxis = 3

   *the number of $n$ harmonics used in the Jacobian $m = 1$ harmonic elimination method; only relevant if `Lrzaxis.`$\leftrightarrow$`ge.1`.*

- integer inputlist::lbeltrami = 4

   *Control flag for solution of Beltrami equation.*

- integer inputlist::linitgues = 1

   *controls how initial guess for Beltrami field is constructed*

- integer inputlist::lposdef = 0

   *redundant;*

- real inputlist::maxrndgues = 1.0

   *the maximum random number of the Beltrami field if `Linitgues = 3`*

- integer inputlist::lmatsolver = 3

   *1 for LU factorization, 2 for GMRES, 3 for GMRES matrix-free*

- integer inputlist::nitergmres = 200

   *number of max iteration for GMRES*

- real inputlist::epsgmres = 1e-14

   *the precision of GMRES*

- integer inputlist::lgmresprec = 1

   *type of preconditioner for GMRES, 1 for ILU sparse matrix*

- real inputlist::epsilu = 1e-12

   *the precision of incomplete LU factorization for preconditioning*

- integer inputlist::lfindzero = 0

   *use Newton methods to find zero of force-balance, which is computed by dforce()*

- real inputlist::escale = 0.0

   *controls the weight factor, `BBweight`, in the force-imbalance harmonics*

- real inputlist::opsilon = 1.0

   *weighting of force-imbalance*

- real inputlist::pcondense = 2.0

   *spectral condensation parameter*

- real inputlist::epsilon = 0.0

*weighting of spectral-width constraint*

- real inputlist::wpoloidal = 1.0

  *"star-like" poloidal angle constraint radial exponential factor used in preset() to construct* `sweight`

- real inputlist::upsilon = 1.0

  *weighting of "star-like" poloidal angle constraint used in preset() to construct* `sweight`

- real inputlist::forcetol = 1.0e-10

  *required tolerance in force-balance error; only used as an initial check*

- real inputlist::c05xmax = 1.0e-06

  *required tolerance in position,* $\mathbf{x} \equiv \{R_{i,v}, Z_{i,v}\}$

- real inputlist::c05xtol = 1.0e-12

  *required tolerance in position,* $\mathbf{x} \equiv \{R_{i,v}, Z_{i,v}\}$

- real inputlist::c05factor = 1.0e-02

  *used to control initial step size in* `C05NDF` *and* `C05PDF`

- logical inputlist::lreadgf = .true.

  *read* $\nabla_{\mathbf{x}}\mathbf{F}$ *from file* `ext.GF`

- integer inputlist::mfreeits = 0

  *maximum allowed free-boundary iterations*

- real inputlist::bnstol = 1.0e-06

  *redundant;*

- real inputlist::bnsblend = 0.666

  *redundant;*

- real inputlist::gbntol = 1.0e-06

  *required tolerance in free-boundary iterations*

- real inputlist::gbnbld = 0.666

  *normal blend*

- real inputlist::vcasingeps = 1.e-12

  *regularization of Biot-Savart; see bnorml(), casing()*

- real inputlist::vcasingtol = 1.e-08

  *accuracy on virtual casing integral; see bnorml(), casing()*

- integer inputlist::vcasingits = 8

  *minimum number of calls to adaptive virtual casing routine; see casing()*

- integer inputlist::vcasingper = 1

  *periods of integragion in adaptive virtual casing routine; see casing()*

- integer inputlist::mcasingcal = 8

  *minimum number of calls to adaptive virtual casing routine; see casing(); redundant;*

- real inputlist::odetol = 1.0e-07

  *o.d.e. integration tolerance for all field line tracing routines*

- real inputlist::absreq = 1.0e-08

  *redundant*

- real inputlist::relreq = 1.0e-08

  *redundant*

- real inputlist::absacc = 1.0e-04

  *redundant*

- real inputlist::epsr = 1.0e-08

  *redundant*

- integer inputlist::nppts = 0

  *number of toroidal transits used (per trajectory) in following field lines for constructing Poincaré plots; if* `nPpts<1`, *no Poincaré plot is constructed;*

- real inputlist::ppts = 0.0

  *stands for Poincare plot theta start. Chose at which angle (normalized over* $\pi$ *) the Poincare field-line tracing start.*

- integer, dimension(1:mnvol+1) inputlist::nptrj = -1

   *number of trajectories in each annulus to be followed in constructing Poincaré plot*

- logical [inputlist::lhevalues](#) = .false.

   *to compute eigenvalues of* $\nabla \mathbf{F}$

- logical [inputlist::lhevectors](#) = .false.

   *to compute eigenvectors (and also eigenvalues) of* $\nabla \mathbf{F}$

- logical [inputlist::lhmatrix](#) = .false.

   *to compute and write to file the elements of* $\nabla \mathbf{F}$

- integer [inputlist::lperturbed](#) = 0

   *to compute linear, perturbed equilibrium*

- integer [inputlist::dpp](#) = -1

   *perturbed harmonic*

- integer [inputlist::dqq](#) = -1

   *perturbed harmonic*

- integer [inputlist::lerrortype](#) = 0

   *the type of error output for Lcheck=1*

- integer [inputlist::ngrid](#) = -1

   *the number of points to output in the grid, -1 for Lrad(vvol)*

- real [inputlist::drz](#) = 1E-5

   *difference in geometry for finite difference estimate (debug only)*

- integer [inputlist::lcheck](#) = 0

   *implement various checks*

- logical [inputlist::ltiming](#) = .false.

   *to check timing*

- real [inputlist::fudge](#) = 1.0e-00

   *redundant*

- real [inputlist::scaling](#) = 1.0e-00

   *redundant*

- logical [inputlist::wbuild_vector_potential](#) = .false.
- logical [inputlist::wreadin](#) = .false.

   *write screen output of [readin()](#)*

- logical [inputlist::wwrtend](#) = .false.

   *write screen output of [wrtend()](#)*

- logical [inputlist::wmacros](#) = .false.

   *write screen output from expanded macros*

- integer [allglobal::myid](#)

   *MPI rank of current CPU.*

- integer [allglobal::ncpu](#)

   *number of MPI tasks*

- integer [allglobal::ismyvolumevalue](#)

   *flag to indicate if a CPU is operating on its assigned volume*

- real [allglobal::cpus](#)

   *initial time*

- real [allglobal::pi2nfp](#)

   *pi2/nfp ; assigned in [readin()](#)*

- real [allglobal::pi2pi2nfp](#)

   $4\pi^2 Nfp$

- real [allglobal::pi2pi2nfphalf](#)

   $2\pi^2 Nfp$

- real [allglobal::pi2pi2nfpquart](#)

   $\pi^2 Nfp$

- real [allglobal::forceerr](#)

*total force-imbalance*

- real allglobal::energy

    *MHD energy.*

- real, dimension(:), allocatable allglobal::ipdt

    *Toroidal pressure-driven current.*

- real, dimension(:,:), allocatable allglobal::ipdtdpf

    *Toroidal pressure-driven current.*

- integer allglobal::mvol

    *number of total volumes; equal to Nvol for fixed-boundary; equal to Nvol+1 for free-boundary*

- logical allglobal::yesstellsym

    *internal shorthand copies of Istellsym, which is an integer input;*

- logical allglobal::notstellsym

    *internal shorthand copies of Istellsym, which is an integer input;*

- logical **allglobal::yesmatrixfree**
- logical allglobal::notmatrixfree

    *to use matrix-free method or not*

- real, dimension(:,:), allocatable allglobal::cheby

    *local workspace for evaluation of Chebychev polynomials*

- real, dimension(:,:,:), allocatable allglobal::zernike

    *local workspace for evaluation of Zernike polynomials*

- real, dimension(:,:,:), allocatable allglobal::tt

    *derivatives of Chebyshev polynomials at the inner and outer interfaces;*

- real, dimension(:,:,:,:), allocatable allglobal::rtt

    *derivatives of Zernike polynomials at the inner and outer interfaces;*

- real, dimension(:,:), allocatable allglobal::rtm

    $r^m$ *term of Zernike polynomials at the origin*

- real, dimension(:), allocatable allglobal::zernikedof

    *Zernike degree of freedom for each* $m$.

- logical, dimension(:), allocatable allglobal::imagneticok

    *used to indicate if Beltrami fields have been correctly constructed;*

- logical allglobal::iconstraintok

    *Used to break iteration loops of slaves in the global constraint minimization.*

- real, dimension(:,:), allocatable allglobal::beltramierror

    *to store the integral of* |*curlB-mu∗B*| *computed by jo00aa;*

- integer allglobal::mn

    *total number of Fourier harmonics for coordinates/fields; calculated from Mpol, Ntor in readin()*

- integer, dimension(:), allocatable allglobal::im

    *poloidal mode numbers for Fourier representation*

- integer, dimension(:), allocatable allglobal::in

    *toroidal mode numbers for Fourier representation*

- real, dimension(:), allocatable allglobal::halfmm

    *I saw this already somewhere...*

- real, dimension(:), allocatable allglobal::regumm

    *I saw this already somewhere...*

- real allglobal::rscale

    *no idea*

- real, dimension(:,:), allocatable allglobal::psifactor

    *no idea*

- real, dimension(:,:), allocatable allglobal::inifactor

    *no idea*

- real, dimension(:), allocatable allglobal::bbweight

*weight on force-imbalance harmonics; used in [dforce()](#)*

- real, dimension(:), allocatable [allglobal::mmpp](#)

    *spectral condensation factors*

- integer [allglobal::mne](#)

    *enhanced resolution for metric elements*

- integer, dimension(:), allocatable [allglobal::ime](#)

    *enhanced poloidal mode numbers for metric elements*

- integer, dimension(:), allocatable [allglobal::ine](#)

    *enhanced toroidal mode numbers for metric elements*

- integer [allglobal::mns](#)

    *enhanced resolution for straight field line transformation*

- integer, dimension(:), allocatable [allglobal::ims](#)

    *enhanced poloidal mode numbers for straight field line transformation*

- integer, dimension(:), allocatable [allglobal::ins](#)

    *enhanced toroidal mode numbers for straight field line transformation*

- integer [allglobal::lmpol](#)

    *what is this?*

- integer [allglobal::lntor](#)

    *what is this?*

- integer [allglobal::smpol](#)

    *what is this?*

- integer [allglobal::sntor](#)

    *what is this?*

- real [allglobal::xoffset](#) = 1.0

    *used to normalize NAG routines (which ones exacly where?)*

- real, dimension(:,:), allocatable [allglobal::irbc](#)

    *cosine R harmonics of interface surface geometry; stellarator symmetric*

- real, dimension(:,:), allocatable [allglobal::izbs](#)

    *sine Z harmonics of interface surface geometry; stellarator symmetric*

- real, dimension(:,:), allocatable [allglobal::irbs](#)

    *sine R harmonics of interface surface geometry; non-stellarator symmetric*

- real, dimension(:,:), allocatable [allglobal::izbc](#)

    *cosine Z harmonics of interface surface geometry; non-stellarator symmetric*

- real, dimension(:,:), allocatable [allglobal::drbc](#)

    *cosine R harmonics of interface surface geometry; stellarator symmetric; linear deformation*

- real, dimension(:,:), allocatable [allglobal::dzbs](#)

    *sine Z harmonics of interface surface geometry; stellarator symmetric; linear deformation*

- real, dimension(:,:), allocatable [allglobal::drbs](#)

    *sine R harmonics of interface surface geometry; non-stellarator symmetric; linear deformation*

- real, dimension(:,:), allocatable [allglobal::dzbc](#)

    *cosine Z harmonics of interface surface geometry; non-stellarator symmetric; linear deformation*

- real, dimension(:,:), allocatable [allglobal::irij](#)

    *interface surface geometry; real space*

- real, dimension(:,:), allocatable [allglobal::izij](#)

    *interface surface geometry; real space*

- real, dimension(:,:), allocatable [allglobal::drij](#)

    *interface surface geometry; real space*

- real, dimension(:,:), allocatable [allglobal::dzij](#)

    *interface surface geometry; real space*

- real, dimension(:,:), allocatable [allglobal::trij](#)

    *interface surface geometry; real space*

- real, dimension(:,:), allocatable [allglobal::tzij](#)

    *interface surface geometry; real space*

- real, dimension(:), allocatable [allglobal::ivns](#)

    *sine harmonics of vacuum normal magnetic field on interfaces; stellarator symmetric*

- real, dimension(:), allocatable [allglobal::ibns](#)

    *sine harmonics of plasma normal magnetic field on interfaces; stellarator symmetric*

- real, dimension(:), allocatable [allglobal::ivnc](#)

    *cosine harmonics of vacuum normal magnetic field on interfaces; non-stellarator symmetric*

- real, dimension(:), allocatable [allglobal::ibnc](#)

    *cosine harmonics of plasma normal magnetic field on interfaces; non-stellarator symmetric*

- real, dimension(:), allocatable [allglobal::lrbc](#)

    *local workspace*

- real, dimension(:), allocatable [allglobal::lzbs](#)

    *local workspace*

- real, dimension(:), allocatable [allglobal::lrbs](#)

    *local workspace*

- real, dimension(:), allocatable [allglobal::lzbc](#)

    *local workspace*

- integer [allglobal::nt](#)

    *discrete resolution along $\theta$ of grid in real space*

- integer [allglobal::nz](#)

    *discrete resolution along $\zeta$ of grid in real space*

- integer [allglobal::ntz](#)

    *discrete resolution; Ntz=Nt*Nz shorthand*

- integer [allglobal::hnt](#)

    *discrete resolution; Ntz=Nt*Nz shorthand*

- integer [allglobal::hnz](#)

    *discrete resolution; Ntz=Nt*Nz shorthand*

- real [allglobal::sontz](#)

    *one / sqrt (one*Ntz); shorthand*

- real, dimension(:,:,:), allocatable [allglobal::rij](#)

    *real-space grid; R*

- real, dimension(:,:,:), allocatable [allglobal::zij](#)

    *real-space grid; Z*

- real, dimension(:,:,:), allocatable [allglobal::xij](#)

    *what is this?*

- real, dimension(:,:,:), allocatable [allglobal::yij](#)

    *what is this?*

- real, dimension(:,:), allocatable [allglobal::sg](#)

    *real-space grid; jacobian and its derivatives*

- real, dimension(:,:,:,:), allocatable [allglobal::guvij](#)

    *real-space grid; metric elements*

- real, dimension(:,:,:), allocatable [allglobal::gvuij](#)

    *real-space grid; metric elements (?); 10 Dec 15;*

- real, dimension(:,:,:,:), allocatable [allglobal::guvijsave](#)

    *what is this?*

- integer, dimension(:,:), allocatable [allglobal::ki](#)

    *identification of Fourier modes*

- integer, dimension(:,:,:), allocatable [allglobal::kijs](#)

    *identification of Fourier modes*

- integer, dimension(:,:,:), allocatable [allglobal::kija](#)

*identification of Fourier modes*
- integer, dimension(:), allocatable [allglobal::iotakkii](allglobal::iotakkii)

   *identification of Fourier modes*
- integer, dimension(:,:), allocatable [allglobal::iotaksub](allglobal::iotaksub)

   *identification of Fourier modes*
- integer, dimension(:,:), allocatable [allglobal::iotakadd](allglobal::iotakadd)

   *identification of Fourier modes*
- integer, dimension(:,:), allocatable [allglobal::iotaksgn](allglobal::iotaksgn)

   *identification of Fourier modes*
- real, dimension(:), allocatable [allglobal::efmn](allglobal::efmn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::ofmn](allglobal::ofmn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::cfmn](allglobal::cfmn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::sfmn](allglobal::sfmn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::evmn](allglobal::evmn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::odmn](allglobal::odmn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::comn](allglobal::comn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::simn](allglobal::simn)

   *Fourier harmonics; dummy workspace.*
- real, dimension(:), allocatable [allglobal::ijreal](allglobal::ijreal)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::ijimag](allglobal::ijimag)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::jireal](allglobal::jireal)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::jiimag](allglobal::jiimag)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::jkreal](allglobal::jkreal)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::jkimag](allglobal::jkimag)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::kjreal](allglobal::kjreal)

   *what is this ?*
- real, dimension(:), allocatable [allglobal::kjimag](allglobal::kjimag)

   *what is this ?*
- real, dimension(:,:,:), allocatable [allglobal::bsupumn](allglobal::bsupumn)

   *tangential field on interfaces; $\theta$-component; required for virtual casing construction of field; 11 Oct 12*
- real, dimension(:,:,:), allocatable [allglobal::bsupvmn](allglobal::bsupvmn)

   *tangential field on interfaces; $\zeta$ -component; required for virtual casing construction of field; 11 Oct 12*
- real, dimension(:,:), allocatable [allglobal::goomne](allglobal::goomne)

   *described in [preset()](preset())*
- real, dimension(:,:), allocatable [allglobal::goomno](allglobal::goomno)

   *described in [preset()](preset())*
- real, dimension(:,:), allocatable [allglobal::gssmne](allglobal::gssmne)

   *described in [preset()](preset())*

- real, dimension(:,:), allocatable allglobal::gssmno

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gstmne

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gstmno

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gszmne

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gszmno

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gttmne

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gttmno

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gtzmne

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gtzmno

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gzzmne

    *described in preset()*
- real, dimension(:,:), allocatable allglobal::gzzmno

    *described in preset()*
- real, dimension(:,:,:,:), allocatable allglobal::dtoocc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::dtoocs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::dtoosc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::dtooss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::ttsscc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::ttsscs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::ttsssc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::ttssss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdstcc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdstcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdstsc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdstss

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdszcc

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdszcs

    *volume-integrated Chebychev-metrics; see matrix()*
- real, dimension(:,:,:,:), allocatable allglobal::tdszsc

*volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::tdszss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttcc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttcs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttsc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddttss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzcc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzcs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzsc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddtzss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzcc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzcs](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzsc](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:,:,:,:), allocatable [allglobal::ddzzss](#)

  *volume-integrated Chebychev-metrics; see [matrix()](#)*

- real, dimension(:,:), allocatable [allglobal::tsc](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::tss](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::dtc](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::dts](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::dzc](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::dzs](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::ttc](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::tzc](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::tts](#)

  *what is this?*

- real, dimension(:,:), allocatable [allglobal::tzs](#)

  *what is this?*

- real, dimension(:), allocatable [allglobal::dtflux](#)

  $\delta\psi_{toroidal}$ *in each annulus*

- real, dimension(:), allocatable [allglobal::dpflux](#)

  $\delta\psi_{poloidal}$ *in each annulus*

- real, dimension(:), allocatable allglobal::sweight

    *minimum poloidal length constraint weight*
- integer, dimension(:), allocatable allglobal::nadof

    *degrees of freedom in Beltrami fields in each annulus*
- integer, dimension(:), allocatable allglobal::nfielddof

    *degrees of freedom in Beltrami fields in each annulus, field only, no Lagrange multipliers*
- type(subgrid), dimension(:,:,:), allocatable allglobal::ate

    *magnetic vector potential cosine Fourier harmonics; stellarator-symmetric*
- type(subgrid), dimension(:,:,:), allocatable allglobal::aze

    *magnetic vector potential cosine Fourier harmonics; stellarator-symmetric*
- type(subgrid), dimension(:,:,:), allocatable allglobal::ato

    *magnetic vector potential sine Fourier harmonics; non-stellarator-symmetric*
- type(subgrid), dimension(:,:,:), allocatable allglobal::azo

    *magnetic vector potential sine Fourier harmonics; non-stellarator-symmetric*
- integer, dimension(:,:), allocatable allglobal::lma

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmb

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmc

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmd

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lme

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmf

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmg

    *Lagrange multipliers (?)*
- integer, dimension(:,:), allocatable allglobal::lmh

    *Lagrange multipliers (?)*
- real, dimension(:,:), allocatable allglobal::lmavalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmbvalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmcvalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmdvalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmevalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmfvalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmgvalue

    *what is this?*
- real, dimension(:,:), allocatable allglobal::lmhvalue

    *what is this?*
- integer, dimension(:,:), allocatable allglobal::fso

    *what is this?*
- integer, dimension(:,:), allocatable allglobal::fse

    *what is this?*
- logical allglobal::lcoordinatesingularity

*set by* `LREGION` *macro; true if inside the innermost volume*

- logical [allglobal::lplasmaregion](#)

    *set by* `LREGION` *macro; true if inside the plasma region*

- logical [allglobal::lvacuumregion](#)

    *set by* `LREGION` *macro; true if inside the vacuum region*

- logical [allglobal::lsavedguvij](#)

    *flag used in matrix free*

- logical [allglobal::localconstraint](#)

    *what is this?*

- real, dimension(:,:), allocatable [allglobal::dma](#)

    *energy and helicity matrices; quadratic forms*

- real, dimension(:,:), allocatable [allglobal::dmb](#)

    *energy and helicity matrices; quadratic forms*

- real, dimension(:,:), allocatable [allglobal::dmd](#)

    *energy and helicity matrices; quadratic forms*

- real, dimension(:), allocatable [allglobal::dmas](#)

    *sparse version of dMA, data*

- real, dimension(:), allocatable [allglobal::dmds](#)

    *sparse version of dMD, data*

- integer, dimension(:), allocatable [allglobal::idmas](#)

    *sparse version of dMA and dMD, indices*

- integer, dimension(:), allocatable [allglobal::jdmas](#)

    *sparse version of dMA and dMD, indices*

- integer, dimension(:), allocatable [allglobal::ndmasmax](#)

    *number of elements for sparse matrices*

- integer, dimension(:), allocatable [allglobal::ndmas](#)

    *number of elements for sparse matrices*

- real, dimension(:), allocatable [allglobal::dmg](#)

    *what is this?*

- real, dimension(:,:), allocatable [allglobal::solution](#)

    *this is allocated in dforce; used in mp00ac and ma02aa; and is passed to packab*

- real, dimension(:,:,:), allocatable [allglobal::gmreslastsolution](#)

    *used to store the last solution for restarting GMRES*

- real, dimension(:), allocatable [allglobal::mbpsi](#)

    *matrix vector products*

- logical [allglobal::liluprecond](#)

    *whether to use ILU preconditioner for GMRES*

- real, dimension(:,:), allocatable [allglobal::beltramiinverse](#)

    *what is this?*

- real, dimension(:,:,:), allocatable [allglobal::diotadxup](#)

    *measured rotational transform on inner/outer interfaces for each volume; d(transform)/dx; (see dforce)*

- real, dimension(:,:,:), allocatable [allglobal::ditgpdxtp](#)

    *measured toroidal and poloidal current on inner/outer interfaces for each volume; d(Itor,Gpol)/dx; (see dforce)*

- real, dimension(:,:,:,:), allocatable [allglobal::glambda](#)

    *save initial guesses for iterative calculation of rotational-transform*

- integer [allglobal::lmns](#)

    *what is this?*

- real, dimension(:,:,:), allocatable [allglobal::bemn](#)

    *force vector; stellarator-symmetric (?)*

- real, dimension(:,:), allocatable [allglobal::iomn](#)

    *force vector; stellarator-symmetric (?)*

- real, dimension(:,:,:), allocatable allglobal::somn

    *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::pomn

    *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::bomn

    *force vector; stellarator-symmetric (?)*
- real, dimension(:,:), allocatable allglobal::iemn

    *force vector; stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::semn

    *force vector; non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::pemn

    *force vector; non-stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::bbe

    *force vector (?); stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::iio

    *force vector (?); stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::bbo

    *force vector (?); non-stellarator-symmetric (?)*
- real, dimension(:), allocatable allglobal::iie

    *force vector (?); non-stellarator-symmetric (?)*
- real, dimension(:,:,:), allocatable allglobal::btemn

    *covariant $\theta$ cosine component of the tangential field on interfaces; stellarator-symmetric*
- real, dimension(:,:,:), allocatable allglobal::bzemn

    *covariant $\zeta$ cosine component of the tangential field on interfaces; stellarator-symmetric*
- real, dimension(:,:,:), allocatable allglobal::btomn

    *covariant $\theta$ sine component of the tangential field on interfaces; non-stellarator-symmetric*
- real, dimension(:,:,:), allocatable allglobal::bzomn

    *covariant $\zeta$ sine component of the tangential field on interfaces; non-stellarator-symmetric*
- real, dimension(:,:), allocatable allglobal::bloweremn

    *covariant field for Hessian computation*
- real, dimension(:,:), allocatable allglobal::bloweromn

    *covariant field for Hessian computation*
- integer allglobal::lgdof

    *geometrical degrees of freedom associated with each interface*
- integer allglobal::ngdof

    *total geometrical degrees of freedom*
- real, dimension(:,:,:), allocatable allglobal::dbbdrz

    *derivative of magnetic field w.r.t. geometry (?)*
- real, dimension(:,:), allocatable allglobal::diidrz

    *derivative of spectral constraints w.r.t. geometry (?)*
- real, dimension(:,:,:,:,:), allocatable allglobal::dffdrz

    *derivatives of B$^\wedge$2 at the interfaces wrt geometry*
- real, dimension(:,:,:,:,:), allocatable allglobal::dbbdmp

    *derivatives of B$^\wedge$2 at the interfaces wrt mu and dpflux*
- real, dimension(:,:,:,:,:,:), allocatable allglobal::dmupfdx

    *derivatives of mu and dpflux wrt geometry at constant interface transform*
- logical allglobal::lhessianallocated

    *flag to indicate that force gradient matrix is allocated (?)*
- real, dimension(:,:), allocatable allglobal::hessian

    *force gradient matrix (?)*
- real, dimension(:,:), allocatable allglobal::dessian

*derivative of force gradient matrix (?)*

- real, dimension(:,:), allocatable allglobal::cosi

  *some precomputed cosines*

- real, dimension(:,:), allocatable allglobal::sini

  *some precomputed sines*

- real, dimension(:), allocatable allglobal::gteta

  *something related to $\sqrt{g}$ and $\theta$ ?*

- real, dimension(:), allocatable allglobal::gzeta

  *something related to $\sqrt{g}$ and $\zeta$ ?*

- real, dimension(:), allocatable allglobal::ajk

  *definition of coordinate axis*

- real, dimension(:,:,:,:), allocatable allglobal::dradr

  *derivatives of coordinate axis*

- real, dimension(:,:,:,:), allocatable allglobal::dradz

  *derivatives of coordinate axis*

- real, dimension(:,:,:,:), allocatable allglobal::dzadr

  *derivatives of coordinate axis*

- real, dimension(:,:,:,:), allocatable allglobal::dzadz

  *derivatives of coordinate axis*

- real, dimension(:,:,:), allocatable allglobal::drodr

  *derivatives of coordinate axis*

- real, dimension(:,:,:), allocatable allglobal::drodz

  *derivatives of coordinate axis*

- real, dimension(:,:,:), allocatable allglobal::dzodr

  *derivatives of coordinate axis*

- real, dimension(:,:,:), allocatable allglobal::dzodz

  *derivatives of coordinate axis*

- integer, dimension(:,:), allocatable allglobal::djkp

  *for calculating cylindrical volume*

- integer, dimension(:,:), allocatable allglobal::djkm

  *for calculating cylindrical volume*

- real, dimension(:), allocatable allglobal::lbbintegral

  *B.B integral.*

- real, dimension(:), allocatable allglobal::labintegral

  *A.B integral.*

- real, dimension(:), allocatable allglobal::vvolume

  *volume integral of $\sqrt{g}$; computed in volume*

- real allglobal::dvolume

  *derivative of volume w.r.t. interface geometry*

- integer allglobal::ivol

  *labels volume; some subroutines (called by NAG) are fixed argument list but require the volume label*

- real allglobal::gbzeta

  *toroidal (contravariant) field; calculated in bfield; required to convert $\dot{\theta}$ to $B^{\theta}$, $\dot{s}$ to $B^{s}$*

- integer, dimension(:), allocatable allglobal::iquad

  *internal copy of Nquad*

- real, dimension(:,:), allocatable allglobal::gaussianweight

  *weights for Gaussian quadrature*

- real, dimension(:,:), allocatable allglobal::gaussianabscissae

  *abscissae for Gaussian quadrature*

- logical allglobal::lblinear

  *controls selection of Beltrami field solver; depends on LBeltrami*

---

- logical [allglobal::lbnewton](#)

    *controls selection of Beltrami field solver; depends on LBeltrami*

- logical [allglobal::lbsequad](#)

    *controls selection of Beltrami field solver; depends on LBeltrami*

- real, dimension(1:3) [allglobal::orzp](#)

    *used in mg00aa() to determine $(s, \theta, \zeta)$ given $(R, Z, \varphi)$*

- type(derivative) [allglobal::dbdx](#)

    $\mathrm{d}\mathbf{B}/\mathrm{d}\mathbf{X}$ *(?)*

- integer [allglobal::globaljk](#)

    *labels position*

- real, dimension(:,:), allocatable [allglobal::dxyz](#)

    *computational boundary; position*


- real, dimension(:,:), allocatable [allglobal::nxyz](#)

    *computational boundary; normal*


- real, dimension(:,:), allocatable [allglobal::jxyz](#)

    *plasma boundary; surface current*

- real, dimension(1:2) [allglobal::tetazeta](#)

    *what is this?*

- real [allglobal::virtualcasingfactor](#) = -one / (four$*$pi)

    *this agrees with diagno*

- integer [allglobal::iberror](#)

    *for computing error in magnetic field*

- integer [allglobal::nfreeboundaryiterations](#)

    *number of free-boundary iterations already performed*

- integer, parameter [allglobal::node](#) = 2

    *best to make this global for consistency between calling and called routines*

- logical [allglobal::first_free_bound](#) = .false.

    *flag to indicate that this is the first free-boundary iteration*

- type(c_ptr) [fftw_interface::planf](#)

    *FFTW-related (?)*

- type(c_ptr) [fftw_interface::planb](#)

    *FFTW-related (?)*

- complex(c_double_complex), dimension(:,:,:), allocatable [fftw_interface::cplxin](#)

    *FFTW-related (?)*

- complex(c_double_complex), dimension(:,:,:), allocatable [fftw_interface::cplxout](#)

    *FFTW-related (?)*


### 11.12.1  Detailed Description

Defines input namelists and global variables, and opens some output files.

Note that all variables in namelist need to be broadcasted in readin.

**Input geometry**

- The geometry of the $l$-th interface, for $l = 0, N$ where $N \equiv$ Nvol, is described by a set of Fourier harmonics, using an arbitrary poloidal angle,

$$R_l(\theta, \zeta) \quad = \quad \sum_j R_{j,l} \cos(m_j \theta - n_j \zeta), \tag{291}$$

$$Z_l(\theta, \zeta) \quad = \quad \sum_j Z_{j,l} \sin(m_j \theta - n_j \zeta). \tag{292}$$

- These harmonics are read from the `ext.sp` file and come directly after the namelists described above. The required format is as follows:

$$
\begin{array}{cccccccccc}
m_1 & n_1 & R_{1,0} & Z_{1,0} & R_{1,1} & Z_{1,1} & ... & R_{1,N} & Z_{1,N} \\
m_2 & n_2 & R_{2,0} & Z_{2,0} & R_{2,1} & Z_{2,1} & ... & R_{2,N} & Z_{2,N} \\
... & & & & & & & & \\
m_j & n_j & R_{j,0} & Z_{j,0} & R_{j,1} & Z_{j,1} & ... & R_{j,N} & Z_{j,N} \\
... & & & & & & & &
\end{array} \tag{293}
$$

- The coordinate axis corresponds to $j = 0$ and the outermost boundary corresponds to $j =$ Nvol.

- An arbitrary selection of harmonics may be inluded in any order, but only those within the range specified by Mpol and Ntor will be used.

- The geometry of *all* the interfaces, i.e. $l = 0, N$, including the degenerate "coordinate-axis" interface, must be given.

### 11.12.2 Data Type Documentation

#### 11.12.2.1 type typedefns::subgrid
used for quantities which have different resolutions in different volumes, e.g. the vector potential

**Class Members**

| real, dimension(:), allocatable | s | coefficients |
|---|---|---|
| integer, dimension(:), allocatable | i | indices |

**Class Members**

| real, dimension(:,:), allocatable | mat | |
|---|---|---|
| integer, dimension(:), allocatable | ipivot | |

#### 11.12.2.2 type typedefns::matrixlu

## 11.13 hesian.f90 File Reference

Computes eigenvalues and eigenvectors of derivative matrix, $\nabla_\xi \mathbf{F}$.

**Functions/Subroutines**

- subroutine hesian (NGdof, position, Mvol, mn, LGdof)

  *Computes eigenvalues and eigenvectors of derivative matrix, $\nabla_\xi \mathbf{F}$.*

**11.13.1 Detailed Description**

Computes eigenvalues and eigenvectors of derivative matrix, $\nabla_\xi \mathbf{F}$.

## 11.14 intghs.f90 File Reference

(integrals) ! Calculates volume integrals of Chebyshev-polynomials and covariant field for Hessian computation.

**Data Types**

- type intghs_module::intghs_workspace

**Functions/Subroutines**

- subroutine intghs (lquad, mn, lvol, lrad, idx)

    *Calculates volume integrals of Chebyshev polynomials and covariant field products.*
- subroutine intghs_workspace_init (lvol)

    *init workspace*
- subroutine intghs_workspace_destroy ()

    *free workspace*

**Variables**

- type(intghs_workspace) **intghs_module::wk**

**11.14.1 Detailed Description**

(integrals) ! Calculates volume integrals of Chebyshev-polynomials and covariant field for Hessian computation.

**11.14.2 Function/Subroutine Documentation**

**11.14.2.1 intghs()** `subroutine intghs (`
            `integer, intent(in) lquad,`
            `integer, intent(in) mn,`
            `integer, intent(in) lvol,`
            `integer, intent(in) lrad,`
            `integer, intent(in) idx )`

Calculates volume integrals of Chebyshev polynomials and covariant field products.

**Parameters**

| | |
|---|---|
| *lquad* | |
| *mn* | |
| *lvol* | |
| *lrad* | |
| *idx* | |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, compute_guvijsave(), allglobal::cpus, allglobal::dbdx, allglobal::dtc, allglobal::dts, allglobal::dzc, allglobal::dzs, allglobal::gaussianabscissae, allglobal← ::gaussianweight, get_cheby(), get_zernike(), allglobal::guvij, allglobal::guvijsave, constants::half, allglobal::im, allglobal::in, invfft(), allglobal::lcoordinatesingularity, allglobal::lsavedguvij, allglobal::mne, inputlist::mpol, allglobal← ::mvol, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, allglobal::nt, allglobal::ntz, allglobal::nz, constants← ::one, fileunits::ounit, constants::pi, constants::pi2, allglobal::pi2pi2nfp, allglobal::pi2pi2nfphalf, allglobal::sg, numerical::small, numerical::sqrtmachprec, tfft(), allglobal::tsc, allglobal::tss, allglobal::ttc, allglobal::tts, constants← ::two, allglobal::tzc, allglobal::tzs, numerical::vsmall, inputlist::wmacros, allglobal::yesstellsym, and constants::zero.

Referenced by get_perturbed_solution(), intghs_workspace_destroy(), intghs_workspace_init(), matvec(), and mp00ac().

Here is the call graph for this function:
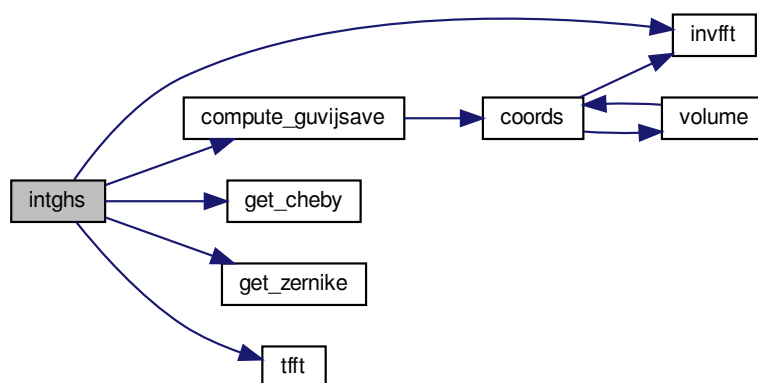


Here is the caller graph for this function:
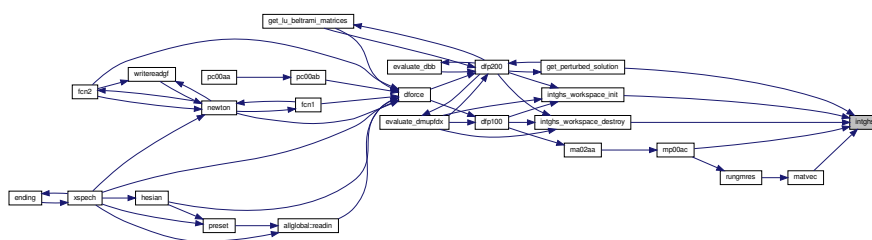


**11.14.2.2   intghs_workspace_init()**    `subroutine intghs_workspace_init (`
            `integer, intent(in)` *lvol* `)`

init workspace

---

**Parameters**

| *lvol* | |
|--------|--|

References allglobal::cpus, intghs(), allglobal::iquad, inputlist::lrad, allglobal::mn, inputlist::mpol, allglobal::myid, allglobal::ncpu, allglobal::ntz, fileunits::ounit, inputlist::wmacros, and constants::zero.

Referenced by dfp100(), dfp200(), and evaluate_dmupfdx().

Here is the call graph for this function:
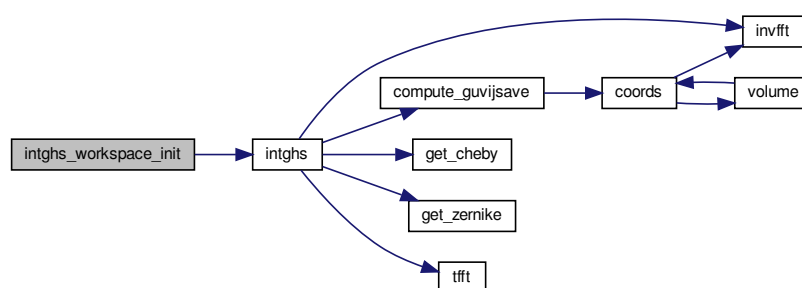


Here is the caller graph for this function:
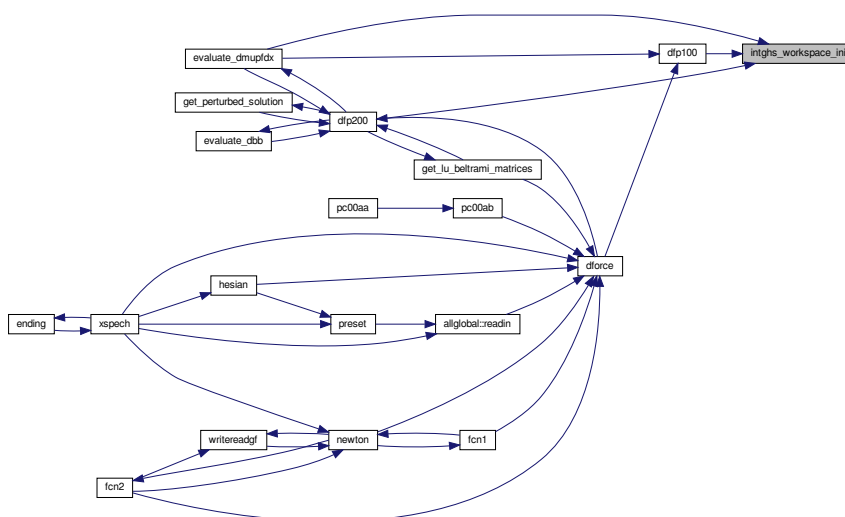


**11.14.2.3 intghs_workspace_destroy()**  subroutine intghs_workspace_destroy

free workspace

**Parameters**

| *lvol* | |
|--------|---|

References allglobal::cpus, intghs(), allglobal::myid, allglobal::ncpu, fileunits::ounit, and inputlist::wmacros.

Referenced by dfp100(), dfp200(), and evaluate_dmupfdx().

Here is the call graph for this function:
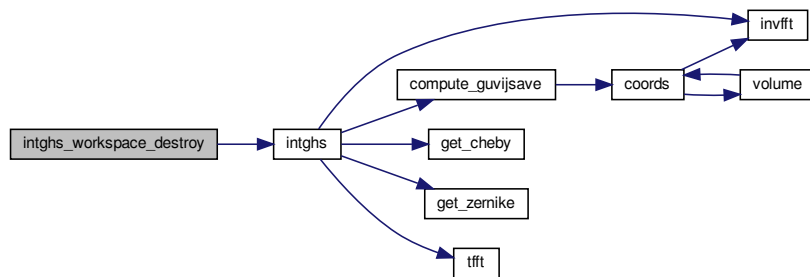


Here is the caller graph for this function:
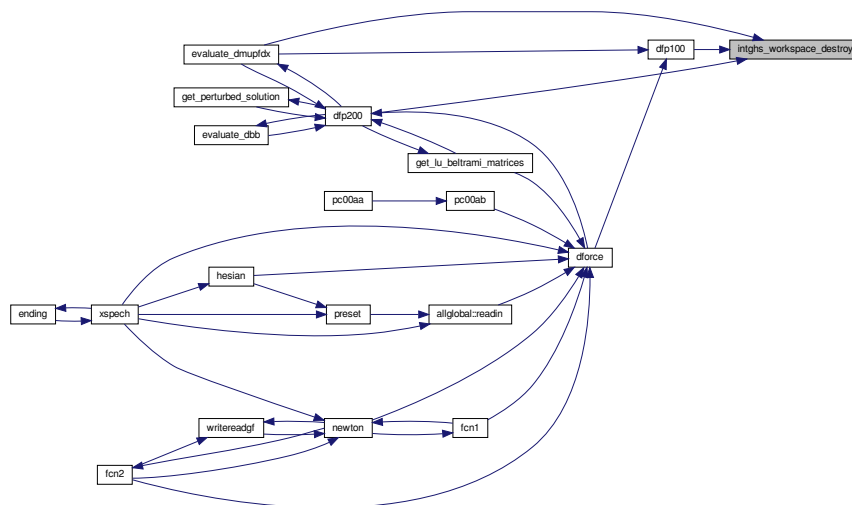


## 11.15 jo00aa.f90 File Reference

Measures error in Beltrami equation, $\nabla \times \mathbf{B} - \mu \mathbf{B}$.

**Functions/Subroutines**

- subroutine jo00aa (lvol, Ntz, lquad, mn)

    *Measures error in Beltrami equation, $\nabla \times \mathbf{B} - \mu \mathbf{B}$.*

### 11.15.1 Detailed Description

Measures error in Beltrami equation, $\nabla \times \mathbf{B} - \mu \mathbf{B}$.

## 11.16 lbpol.f90 File Reference

Computes Btheta at the interface.

### Functions/Subroutines

- subroutine lbpol (lvol, Bt00, ideriv, iocons)

  *Computes Btheta at the interface.*

### 11.16.1 Detailed Description

Computes Btheta at the interface.

### 11.16.2 Function/Subroutine Documentation

#### 11.16.2.1 lbpol()
```
subroutine lbpol (
        integer lvol,
        real, dimension(1:mvol, 0:1, -1:2) Bt00,
        integer ideriv,
        integer iocons )
```

Computes Btheta at the interface.

**Parameters**

| lvol | |
| --- | --- |
| Bt00 | |
| ideriv | |
| iocons | |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::cfmn, allglobal::comn, coords(), allglobal::cpus, allglobal::dbdx, allglobal::efmn, allglobal::evmn, allglobal::guvij, constants::half, inputlist::igeometry, allglobal::im, allglobal::ime, allglobal::in, allglobal::ine, invfft(), inputlist::lcheck, allglobal::lcoordinatesingularity, inputlist::lrad, allglobal::mne, constants::mu0, allglobal::mvol, allglobal::myid, allglobal::notstellsym, allglobal←↩::nt, allglobal::nz, allglobal::odmn, allglobal::ofmn, constants::one, fileunits::ounit, constants::pi, constants::pi2, allglobal::regumm, allglobal::sfmn, allglobal::sg, allglobal::simn, tfft(), allglobal::tt, constants::two, allglobal←↩::yesstellsym, and constants::zero.

Referenced by dfp100(), evaluate_dmupfdx(), and xspech().

Here is the call graph for this function:
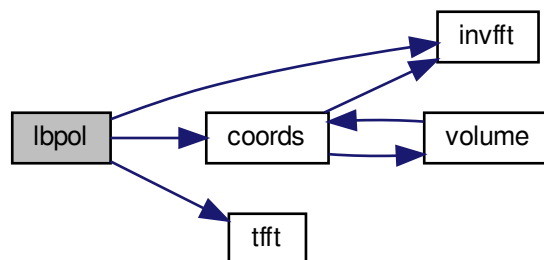


Here is the caller graph for this function:
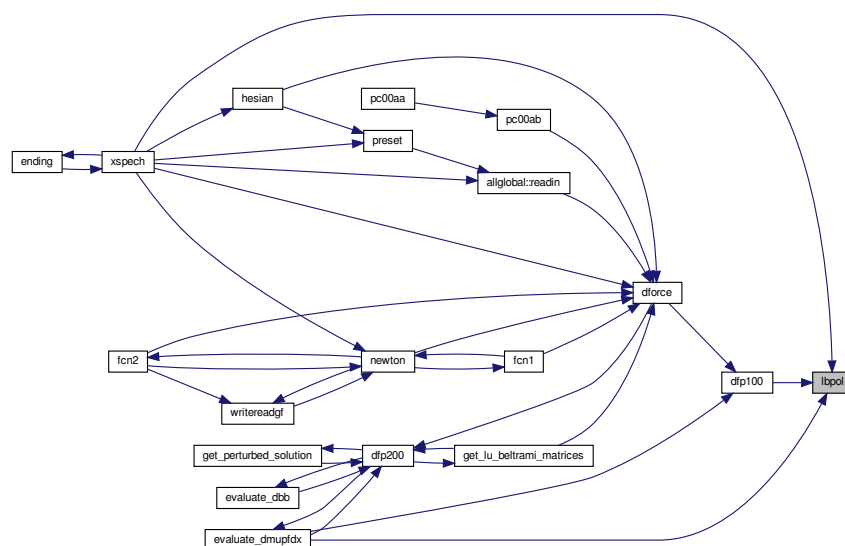


## 11.17 Iforce.f90 File Reference

Computes $B^2$, and the spectral condensation constraints if required, on the interfaces, $\mathcal{I}_i$.

**Functions/Subroutines**

- subroutine Iforce (lvol, iocons, ideriv, Ntz, dBB, XX, YY, length, DDl, MMl, iflag)

  *Computes $B^2$, and the spectral condensation constraints if required, on the interfaces, $\mathcal{I}_i$.*

### 11.17.1 Detailed Description

Computes $B^2$, and the spectral condensation constraints if required, on the interfaces, $\mathcal{I}_i$.

## 11.18   ma00aa.f90 File Reference

Calculates volume integrals of Chebyshev polynomials and metric element products.

**Functions/Subroutines**

- subroutine [ma00aa](#) (lquad, mn, lvol, lrad)

    *Calculates volume integrals of Chebyshev polynomials and metric element products.*

### 11.18.1   Detailed Description

Calculates volume integrals of Chebyshev polynomials and metric element products.

## 11.19   ma02aa.f90 File Reference

Constructs Beltrami field in given volume consistent with flux, helicity, rotational-transform and/or parallel-current constraints.

**Functions/Subroutines**

- subroutine [ma02aa](#) (lvol, NN)

    *Constructs Beltrami field in given volume consistent with flux, helicity, rotational-transform and/or parallel-current constraints.*

### 11.19.1   Detailed Description

Constructs Beltrami field in given volume consistent with flux, helicity, rotational-transform and/or parallel-current constraints.

## 11.20   manual.f90 File Reference

Code development issues and future physics applications.

### 11.20.1   Detailed Description

Code development issues and future physics applications.

**See also**

   [Manual / Documentation](#)

## 11.21   matrix.f90 File Reference

Constructs energy and helicity matrices that represent the Beltrami linear system.

**Functions/Subroutines**

- subroutine [matrix](lvol, mn, lrad)

    *Constructs energy and helicity matrices that represent the Beltrami linear system.*
    ***gauge conditions***
- subroutine **matrixbg** (lvol, mn, lrad)

### 11.21.1 Detailed Description

Constructs energy and helicity matrices that represent the Beltrami linear system.

## 11.22 memory.f90 File Reference

memory management module

**Functions/Subroutines**

- subroutine [allocate_beltrami_matrices](vvol, LcomputeDerivatives)

    *allocate Beltrami matrices*
- subroutine [deallocate_beltrami_matrices](LcomputeDerivatives)

    *deallocate Beltrami matrices*
- subroutine [allocate_geometry_matrices](vvol, LcomputeDerivatives)

    *allocate geometry matrices*
- subroutine [deallocate_geometry_matrices](LcomputeDerivatives)

    *deallocate geometry matrices*

### 11.22.1 Detailed Description

memory management module

### 11.22.2 Function/Subroutine Documentation

#### 11.22.2.1 allocate_beltrami_matrices() subroutine allocate_beltrami_matrices (

```
        integer, intent(in) vvol,
        logical, intent(in) LcomputeDerivatives )
```

allocate Beltrami matrices

**Parameters**

| | |
|---|---|
| *vvol* | |
| *LcomputeDerivatives* | |

---

References allglobal::dma, allglobal::dmas, allglobal::dmb, allglobal::dmd, allglobal::dmds, allglobal::dmg, allglobal::idmas, allglobal::jdmas, allglobal::liluprecond, allglobal::mbpsi, allglobal::nadof, allglobal::ndmasmax, allglobal::notmatrixfree, allglobal::solution, inputlist::wmacros, and constants::zero.

Referenced by dfp100(), dfp200(), and evaluate_dmupfdx().

Here is the caller graph for this function:
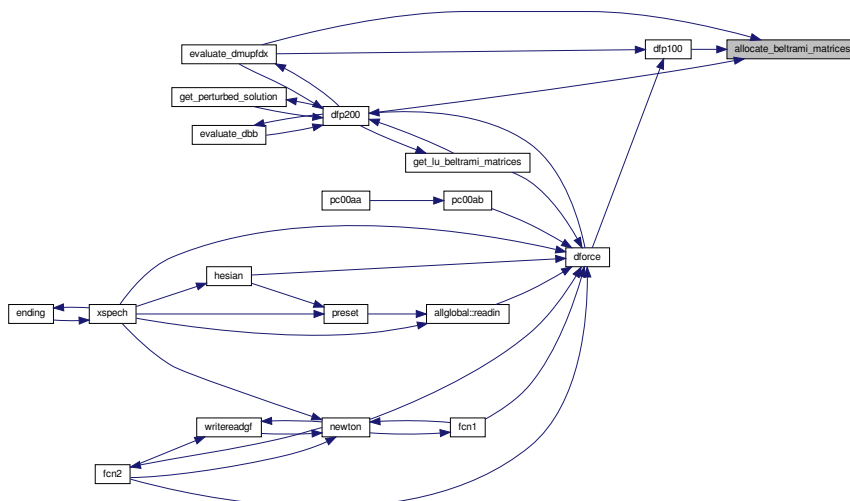


### 11.22.2.2 deallocate_beltrami_matrices() subroutine deallocate_beltrami_matrices ( logical, intent(in) *LcomputeDerivatives* )

deallocate Beltrami matrices

**Parameters**

| LcomputeDerivatives | |
|---|---|

References allglobal::dma, allglobal::dmas, allglobal::dmb, allglobal::dmd, allglobal::dmds, allglobal::dmg, allglobal::idmas, allglobal::jdmas, allglobal::liluprecond, allglobal::mbpsi, allglobal::notmatrixfree, allglobal::solution, and inputlist::wmacros.

Referenced by dfp100(), dfp200(), and evaluate_dmupfdx().

Here is the caller graph for this function:
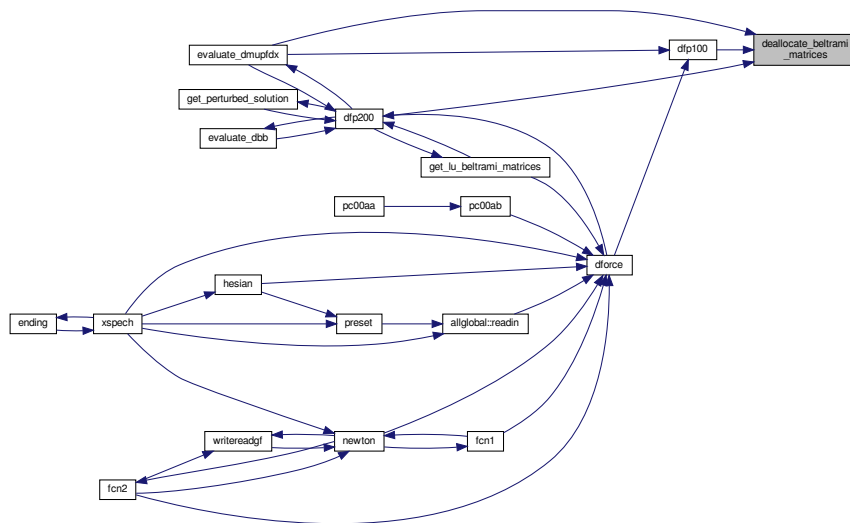


### 11.22.2.3 allocate_geometry_matrices()

```
subroutine allocate_geometry_matrices (
         integer vvol,
         logical, intent(in) LcomputeDerivatives )
```

allocate geometry matrices

**Parameters**

| | |
|---|---|
| *vvol* | |
| *LcomputeDerivatives* | |

References allglobal::ddttcc, allglobal::ddttcs, allglobal::ddttsc, allglobal::ddttss, allglobal::ddtzcc, allglobal↩
::ddtzcs, allglobal::ddtzsc, allglobal::ddtzss, allglobal::ddzzcc, allglobal::ddzzcs, allglobal::ddzzsc, allglobal↩
::ddzzss, allglobal::dtc, allglobal::dtoocc, allglobal::dtoocs, allglobal::dtoosc, allglobal::dtooss, allglobal::dts,
allglobal::dzc, allglobal::dzs, allglobal::guvijsave, allglobal::iquad, allglobal::lcoordinatesingularity, inputlist::lrad,
inputlist::mpol, allglobal::notstellsym, allglobal::tdstcc, allglobal::tdstcs, allglobal::tdstsc, allglobal::tdstss, allglobal↩
::tdszcc, allglobal::tdszcs, allglobal::tdszsc, allglobal::tdszss, allglobal::tsc, allglobal::tss, allglobal::ttc, allglobal::tts,
allglobal::ttsscc, allglobal::ttsscs, allglobal::ttsssc, allglobal::ttssss, allglobal::tzc, allglobal::tzs, inputlist::wmacros,
and constants::zero.

Referenced by dfp100(), dfp200(), and evaluate_dmupfdx().

Here is the caller graph for this function:
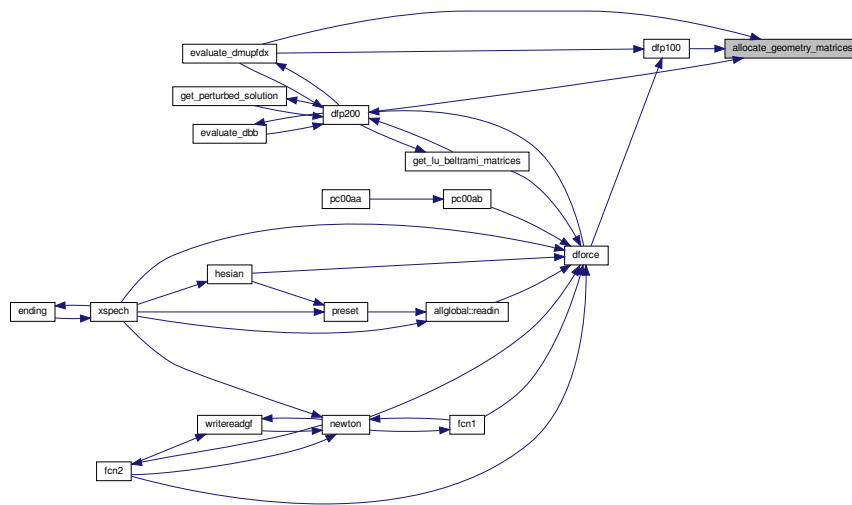


**11.22.2.4 deallocate_geometry_matrices()** subroutine deallocate_geometry_matrices (
     logical, intent(in) *LcomputeDerivatives* )

deallocate geometry matrices

**Parameters**

| *LcomputeDerivatives* | |
| --- | --- |

References allglobal::ddttcc, allglobal::ddttcs, allglobal::ddttsc, allglobal::ddttss, allglobal::ddtzcc, allglobal↩
::ddtzcs, allglobal::ddtzsc, allglobal::ddtzss, allglobal::ddzzcc, allglobal::ddzzcs, allglobal::ddzzsc, allglobal::ddzzss,
allglobal::dtc, allglobal::dtoocc, allglobal::dtoocs, allglobal::dtoosc, allglobal::dtooss, allglobal::dts, allglobal::dzc,
allglobal::dzs, allglobal::guvijsave, allglobal::lsavedguvij, allglobal::notstellsym, allglobal::tdstcc, allglobal::tdstcs,
allglobal::tdstsc, allglobal::tdstss, allglobal::tdszcc, allglobal::tdszcs, allglobal::tdszsc, allglobal::tdszss, allglobal↩
::tsc, allglobal::tss, allglobal::ttc, allglobal::tts, allglobal::ttsscc, allglobal::ttsscs, allglobal::ttsssc, allglobal::ttssss,
allglobal::tzc, allglobal::tzs, inputlist::wmacros, and constants::zero.

Referenced by dfp100(), dfp200(), and evaluate_dmupfdx().

Here is the caller graph for this function:
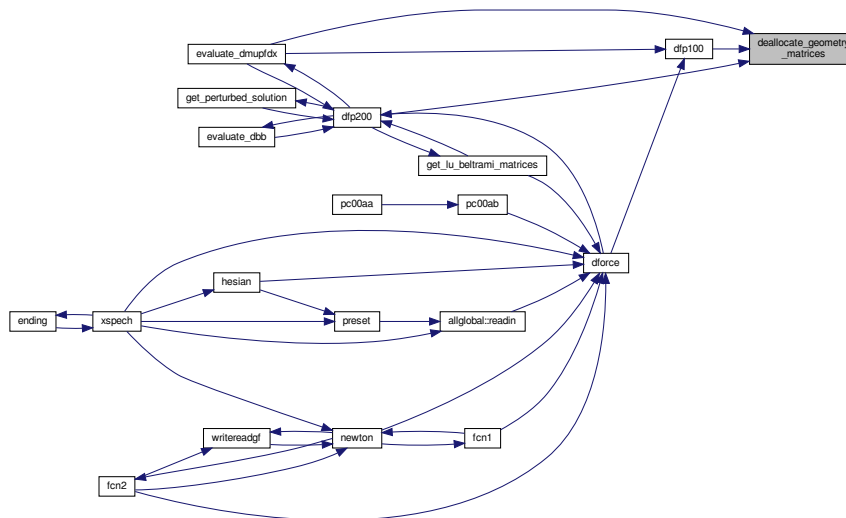


## 11.23   metrix.f90 File Reference

Calculates the metric quantities, $\sqrt{g}\, g^{\mu\nu}$, which are required for the energy and helicity integrals.

### Functions/Subroutines

- subroutine [metrix](#) (lquad, lvol)

  *Calculates the metric quantities, $\sqrt{g}\, g^{\mu\nu}$, which are required for the energy and helicity integrals.*
- subroutine [compute_guvijsave](#) (lquad, vvol, ideriv, Lcurvature)

  *compute guvijsave*

### 11.23.1   Detailed Description

Calculates the metric quantities, $\sqrt{g}\, g^{\mu\nu}$, which are required for the energy and helicity integrals.

### 11.23.2   Function/Subroutine Documentation

**11.23.2.1   compute_guvijsave()**    `subroutine compute_guvijsave (`
            `integer, intent(in) `*`lquad,`*
            `integer, intent(in) `*`vvol,`*
            `integer, intent(in) `*`ideriv,`*
            `integer, intent(in) `*`Lcurvature `*`)`

compute guvijsave

---

**Parameters**

| | |
|---|---|
| *lquad* | |
| *vvol* | |
| *ideriv* | |
| *Lcurvature* | |

References coords(), allglobal::gaussianabscissae, allglobal::guvij, allglobal::guvijsave, and allglobal::sg.

Referenced by dfp100(), intghs(), and ma00aa().

Here is the call graph for this function:
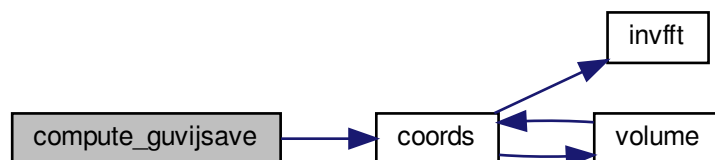


Here is the caller graph for this function:



## 11.24 mp00ac.f90 File Reference

Solves Beltrami/vacuum (linear) system, given matrices.

**Functions/Subroutines**

- subroutine [mp00ac](Ndof, Xdof, Fdof, Ddof, Ldfjac, iflag)

    *Solves Beltrami/vacuum (linear) system, given matrices.*
    **unpacking fluxes, helicity multiplier**
- subroutine [rungmres](n, nrestart, mu, vvol, rhs, sol, ipar, fpar, wk, nw, guess, a, au, jau, ju, iperm, ierr)

    *run GMRES*
- subroutine [matvec](n, x, ax, a, mu, vvol)

    *compute a.x by either by coumputing it directly, or using a matrix free method*
- subroutine [prec_solve](n, vecin, vecout, au, jau, ju, iperm)

    *apply the preconditioner*

### 11.24.1    Detailed Description

Solves Beltrami/vacuum (linear) system, given matrices.

### 11.24.2    Function/Subroutine Documentation

**11.24.2.1    rungmres()**    `subroutine rungmres (`
　　　　　`integer n,`
　　　　　`integer nrestart,`
　　　　　`real mu,`
　　　　　`integer vvol,`
　　　　　`real, dimension(1:n) rhs,`
　　　　　`real, dimension(1:n) sol,`
　　　　　`integer, dimension(16) ipar,`
　　　　　`real, dimension(16) fpar,`
　　　　　`real, dimension(1:nw) wk,`
　　　　　`integer nw,`
　　　　　`real, dimension(n) guess,`
　　　　　`real, dimension(*) a,`
　　　　　`real, dimension(*) au,`
　　　　　`integer, dimension(*) jau,`
　　　　　`integer, dimension(*) ju,`
　　　　　`integer, dimension(*) iperm,`
　　　　　`integer ierr )`

run GMRES

**Parameters**

| | |
|---|---|
| *n* | |
| *nrestart* | |
| *mu* | |
| *vvol* | |
| *rhs* | |
| *sol* | |
| *ipar* | |
| *fpar* | |
| *wk* | |

**Parameters**

| nw | |
|---|---|
| guess | |
| a | |
| au | |
| jau | |
| ju | |
| iperm | |
| ierr | |

References inputlist::epsgmres, allglobal::liluprecond, matvec(), inputlist::nitergmres, constants::one, prec_solve(), and constants::zero.

Referenced by mp00ac().

Here is the call graph for this function:
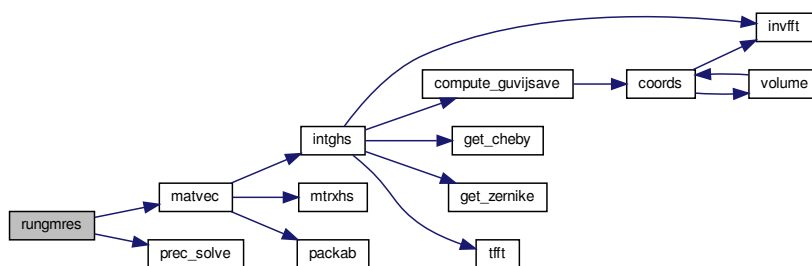


Here is the caller graph for this function:
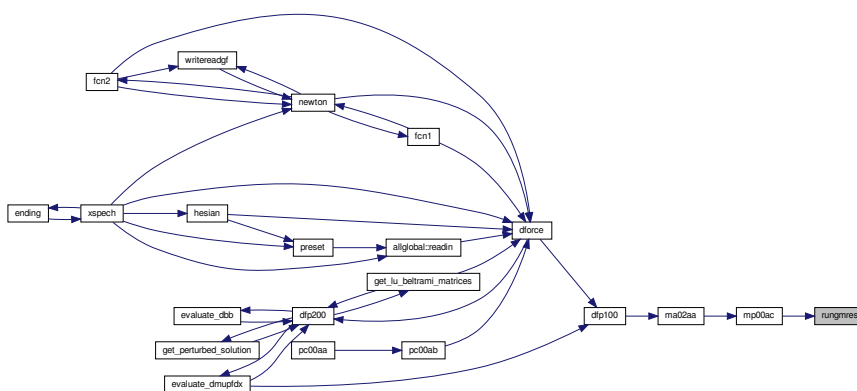
**11.24.2.2   matvec()**   subroutine matvec (
            integer, intent(in) *n,*
            real, dimension(1:n) *x,*
            real, dimension(1:n) *ax,*
            real, dimension(*) *a,*
            real *mu,*
            integer, intent(in) *vvol* )

compute a.x by either by coumputing it directly, or using a matrix free method

**Parameters**

| | |
|------|--|
| *n* | |
| *x* | |
| *ax* | |
| *a* | |
| *mu* | |
| *vvol* | |

References allglobal::dmd, intghs(), allglobal::iquad, inputlist::lrad, mtrxhs(), allglobal::notmatrixfree, constants↩
::one, packab(), and constants::zero.

Referenced by rungmres().

Here is the call graph for this function:
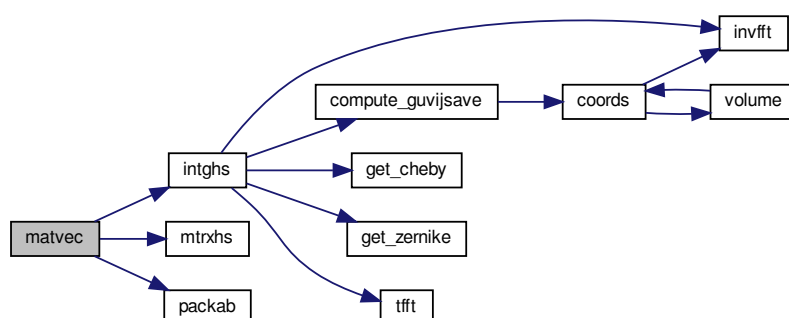


Here is the caller graph for this function:
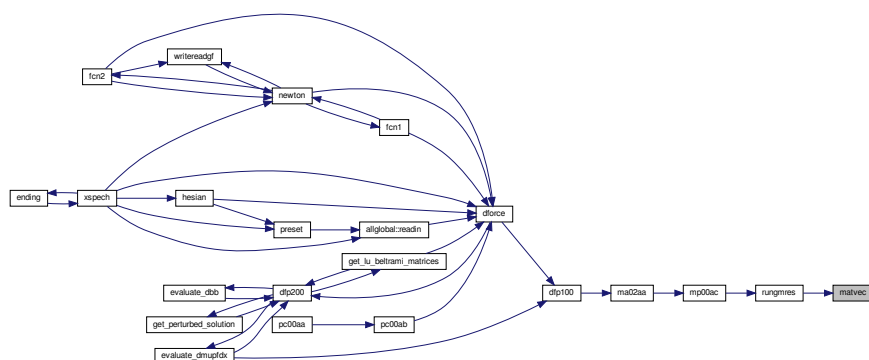
**11.24.2.3 prec_solve()** `subroutine prec_solve (`
        `integer` *n,*
        `real, dimension(*)` *vecin,*
        `real, dimension(*)` *vecout,*
        `real, dimension(*)` *au,*
        `integer, dimension(*)` *jau,*
        `integer, dimension(*)` *ju,*
        `integer, dimension(*)` *iperm* `)`

apply the preconditioner

**Parameters**

| | |
|---|---|
| *n* | |
| *vecin* | |
| *vecout* | |
| *au* | |
| *jau* | |
| *ju* | |
| *iperm* | |

Referenced by rungmres().

Here is the caller graph for this function:
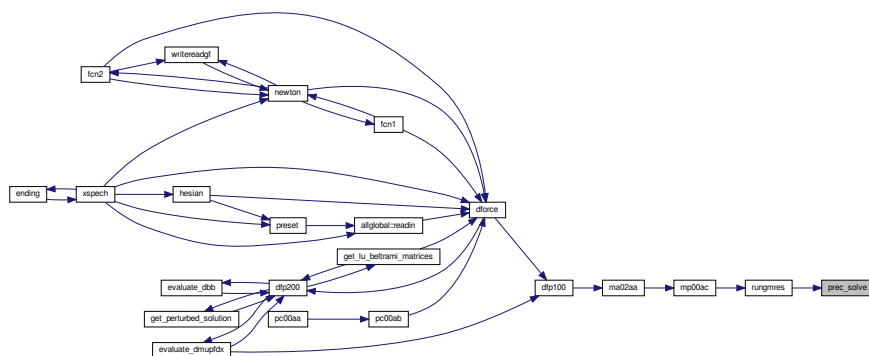


## 11.25 mtrxhs.f90 File Reference

(build matrices) ! Constructs matrices that represent the Beltrami linear system, matrix-free.

**Functions/Subroutines**

- subroutine [mtrxhs](lvol, mn, lrad, resultA, resultD, idx)

  *Constructs matrices that represent the Beltrami linear system, matrix-free.*

### 11.25.1  Detailed Description

(build matrices) ! Constructs matrices that represent the Beltrami linear system, matrix-free.

### 11.25.2  Function/Subroutine Documentation

#### 11.25.2.1  mtrxhs()

```
subroutine mtrxhs (
        integer, intent(in) lvol,
        integer, intent(in) mn,
        integer, intent(in) lrad,
        real, dimension(0:nadof(lvol)), intent(out) resultA,
        real, dimension(0:nadof(lvol)), intent(out) resultD,
        integer, intent(in) idx )
```

Constructs matrices that represent the Beltrami linear system, matrix-free.

**Parameters**

| | |
|---|---|
| *lvol* | |
| *mn* | |
| *lrad* | |
| *resultA* | |
| *resultD* | |
| *idx* | |

References allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::cpus, allglobal::dbdx, allglobal::dtc, allglobal::dts, allglobal::dzc, allglobal::dzs, constants::half, allglobal::im, allglobal::in, allglobal::lcoordinatesingularity, allglobal::lma, allglobal::lmavalue, allglobal::lmb, allglobal::lmbvalue, allglobal::lmc, allglobal::lmcvalue, allglobal↩
::lmd, allglobal::lmdvalue, allglobal::lme, allglobal::lmevalue, allglobal::lmf, allglobal::lmfvalue, allglobal::lmg, allglobal::lmgvalue, allglobal::lmh, allglobal::lmhvalue, inputlist::mpol, allglobal::myid, allglobal::nadof, allglobal↩
::ncpu, allglobal::notstellsym, constants::one, fileunits::ounit, allglobal::rtm, allglobal::rtt, numerical::small, allglobal::tsc, allglobal::tss, allglobal::tt, allglobal::ttc, allglobal::tts, constants::two, allglobal::tzc, allglobal::tzs, inputlist::wmacros, allglobal::yesstellsym, and constants::zero.

Referenced by get_perturbed_solution(), matvec(), and mp00ac().

Here is the caller graph for this function:
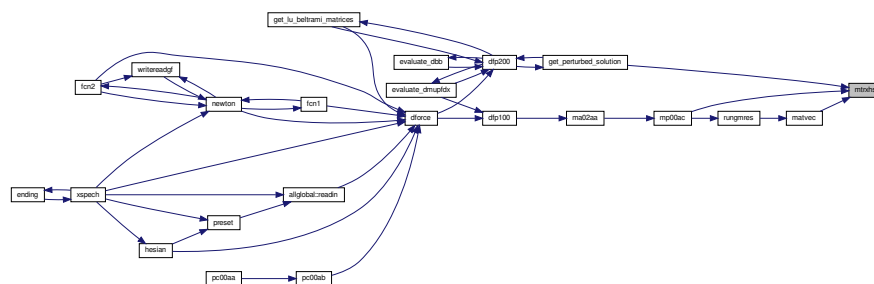
## 11.26 newton.f90 File Reference

Employs Newton method to find $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{x} \equiv \{\text{geometry}\}$ and $\mathbf{F}$ is defined in dforce() .

**Modules**

- module newtontime

  *timing of Newton iterations*

**Functions/Subroutines**

- subroutine newton (NGdof, position, ihybrd)

  *Employs Newton method to find $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{x} \equiv \{\text{geometry}\}$ and $\mathbf{F}$ is defined in dforce() .*
- subroutine writereadgf (readorwrite, NGdof, ireadhessian)

  *read or write force-derivative matrix*
- subroutine fcn1 (NGdof, xx, fvec, irevcm)

  *fcn1*
- subroutine fcn2 (NGdof, xx, fvec, fjac, Ldfjac, irevcm)

  *fcn2*

**Variables**

- integer newtontime::nfcalls

  *number of calls to get function values (?)*
- integer newtontime::ndcalls

  *number of calls to get derivative values (?)*
- real newtontime::lastcpu

  *last CPU that called this (?)*

### 11.26.1 Detailed Description

Employs Newton method to find $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{x} \equiv \{\text{geometry}\}$ and $\mathbf{F}$ is defined in dforce() .

## 11.27 numrec.f90 File Reference

Various miscellaneous "numerical" routines.

**Functions/Subroutines**

- subroutine gi00ab (Mpol, Ntor, Nfp, mn, im, in)

  *Assign Fourier mode labels.*
- subroutine getimn (Mpol, Ntor, Nfp, mi, ni, idx)

  *convert m and n to index*
- subroutine tfft (Nt, Nz, ijreal, ijimag, mn, im, in, efmn, ofmn, cfmn, sfmn, ifail)

  *Forward Fourier transform (fftw wrapper)*
- subroutine invfft (mn, im, in, efmn, ofmn, cfmn, sfmn, Nt, Nz, ijreal, ijimag)

  *Inverse Fourier transform (fftw wrapper)*
- subroutine gauleg (n, weight, abscis, ifail)

  *Gauss-Legendre weights and abscissae.*

### 11.27.1   Detailed Description

Various miscellaneous "numerical" routines.

### 11.27.2   Function/Subroutine Documentation

#### 11.27.2.1   getimn()
```
subroutine getimn (
        integer, intent(in) Mpol,
        integer, intent(in) Ntor,
        integer, intent(in) Nfp,
        integer, intent(in) mi,
        integer, intent(in) ni,
        integer, intent(out) idx )
```

convert m and n to index

**Parameters**

| | |
|------|--|
| *Mpol* | |
| *Ntor* | |
| *Nfp* | |
| *mi* | |
| *ni* | |
| *idx* | |

Referenced by preset().

Here is the caller graph for this function:



## 11.28   packab.f90 File Reference

Packs, and unpacks, Beltrami field solution vector; $\mathbf{a} \equiv \{A_{\theta,e,i,l}, A_{\zeta,e,i,l}, \text{etc.}\}$.

### Functions/Subroutines

- subroutine packab (packorunpack, lvol, NN, solution, ideriv)

  *Packs and unpacks Beltrami field solution vector.*

### 11.28.1 Detailed Description

Packs, and unpacks, Beltrami field solution vector; $\mathbf{a} \equiv \{A_{\theta,e,i,l}, A_{\zeta,e,i,l}, \text{etc.}\}$.

## 11.29 packxi.f90 File Reference

Packs, and unpacks, geometrical degrees of freedom; and sets coordinate axis.

**Functions/Subroutines**

- subroutine packxi (NGdof, position, Mvol, mn, iRbc, iZbs, iRbs, iZbc, packorunpack, LComputeDerivatives, LComputeAxis)

  *Packs, and unpacks, geometrical degrees of freedom; and sets coordinate axis.*

### 11.29.1 Detailed Description

Packs, and unpacks, geometrical degrees of freedom; and sets coordinate axis.

## 11.30 pc00aa.f90 File Reference

Use preconditioned conjugate gradient method to find minimum of energy functional.

**Functions/Subroutines**

- subroutine pc00aa (NGdof, position, Nvol, mn, ie04dgf)

  *Use preconditioned conjugate gradient method to find minimum of energy functional.*

### 11.30.1 Detailed Description

Use preconditioned conjugate gradient method to find minimum of energy functional.

## 11.31 pc00ab.f90 File Reference

Returns the energy functional and it's derivatives with respect to geometry.

**Functions/Subroutines**

- subroutine pc00ab (mode, NGdof, Position, Energy, Gradient, nstate, iuser, ruser)

  *Returns the energy functional and it's derivatives with respect to geometry.*

### 11.31.1 Detailed Description

Returns the energy functional and it's derivatives with respect to geometry.

## 11.32 pp00aa.f90 File Reference

Constructs Poincaré plot and "approximate" rotational-transform (driver).

**Functions/Subroutines**

- subroutine pp00aa

  *Constructs Poincaré plot and "approximate" rotational-transform (driver).*

### 11.32.1 Detailed Description

Constructs Poincaré plot and "approximate" rotational-transform (driver).

## 11.33 pp00ab.f90 File Reference

Follows magnetic fieldline using ode-integration routine from rksuite.f .

**Functions/Subroutines**

- subroutine pp00ab (lvol, sti, Nz, nPpts, poincaredata, fittedtransform, utflag)

  *Constructs Poincaré plot and "approximate" rotational-transform (for single field line).*

### 11.33.1 Detailed Description

Follows magnetic fieldline using ode-integration routine from rksuite.f .

## 11.34 preset.f90 File Reference

Allocates and initializes internal arrays.

**Functions/Subroutines**

- subroutine preset

  *Allocates and initializes internal arrays.*

### 11.34.1 Detailed Description

Allocates and initializes internal arrays.

## 11.35 ra00aa.f90 File Reference

Writes vector potential to .ext.sp.A .

**Functions/Subroutines**

- subroutine ra00aa (writeorread)

    *Writes vector potential to .ext.sp.A .*

### 11.35.1 Detailed Description

Writes vector potential to .ext.sp.A .

## 11.36 rzaxis.f90 File Reference

The coordinate axis is assigned via a poloidal average over an arbitrary surface.

**Functions/Subroutines**

- subroutine rzaxis (Mvol, mn, inRbc, inZbs, inRbs, inZbc, ivol, LcomputeDerivatives)

    *The coordinate axis is assigned via a poloidal average over an arbitrary surface.*

### 11.36.1 Detailed Description

The coordinate axis is assigned via a poloidal average over an arbitrary surface.

## 11.37 sphdf5.f90 File Reference

(output) ! Writes all the output information to ext.h5.

**Functions/Subroutines**

- subroutine sphdf5::init_outfile

    *initialize the interface to the HDF5 library and open the output file*
- subroutine sphdf5::mirror_input_to_outfile

    *mirror input variables into output file*
- subroutine sphdf5::init_convergence_output

    *prepare convergence evolution output*
- subroutine sphdf5::write_convergence_output (nDcalls, ForceErr)

    *write convergence output (evolution of interface geometry, force, etc); was in global.f90/wrtend for wflag.eq.-1 previously*
- subroutine sphdf5::write_grid

    *write the magnetic field on a grid; previously the (wflag.eq.1) part of globals.f90/wrtend to write .ext.sp.grid;*
- subroutine sphdf5::init_flt_output (numTrajTotal)

    *init field line tracing output group and create array datasets*
- subroutine sphdf5::write_poincare (offset, data, success)

    *write a hyperslab of Poincare data*
- subroutine sphdf5::write_transform (offset, length, lvol, diotadxup, fiota)

    *write rotational transform output from field line following*
- subroutine sphdf5::finalize_flt_output

    *finalize Poincare output*
- subroutine sphdf5::write_vector_potential (sumLrad, allAte, allAze, allAto, allAzo)

    *write the magnetic vector potential Fourier harmonics to the output file group /vector_potential*
- subroutine sphdf5::hdfint

    *final output*
- subroutine sphdf5::finish_outfile

    *Close all open HDF5 objects (we know of) and list any remaining still-open objects.*

**Variables**

- logical, parameter **sphdf5::hdfdebug** = .false.
- integer, parameter **sphdf5::internalhdf5msg** = 0
- integer **sphdf5::hdfier**
- integer **sphdf5::rank**
- integer(hid_t) **sphdf5::file_id**
- integer(hid_t) **sphdf5::space_id**
- integer(hid_t) **sphdf5::dset_id**
- integer(hsize_t), dimension(1:1) **sphdf5::onedims**
- integer(hsize_t), dimension(1:2) **sphdf5::twodims**
- integer(hsize_t), dimension(1:3) **sphdf5::threedims**
- logical **sphdf5::grp_exists**
- logical **sphdf5::var_exists**
- integer(hid_t) **sphdf5::iteration_dset_id**
- integer(hid_t) **sphdf5::dataspace**
- integer(hid_t) **sphdf5::memspace**
- integer(hsize_t), dimension(1) **sphdf5::old_data_dims**
- integer(hsize_t), dimension(1) **sphdf5::data_dims**
- integer(hsize_t), dimension(1) **sphdf5::max_dims**
- integer(hid_t) **sphdf5::plist_id**
- integer(hid_t) **sphdf5::dt_ndcalls_id**
- integer(hid_t) **sphdf5::dt_energy_id**
- integer(hid_t) **sphdf5::dt_forceerr_id**
- integer(hid_t) **sphdf5::dt_irbc_id**
- integer(hid_t) **sphdf5::dt_izbs_id**
- integer(hid_t) **sphdf5::dt_irbs_id**
- integer(hid_t) **sphdf5::dt_izbc_id**
- integer, parameter **sphdf5::rankp** =3
- integer, parameter **sphdf5::rankt** =2
- integer(hid_t) **sphdf5::grppoincare**
- integer(hid_t) **sphdf5::dset_id_t**
- integer(hid_t) **sphdf5::dset_id_s**
- integer(hid_t) **sphdf5::dset_id_r**
- integer(hid_t) **sphdf5::dset_id_z**
- integer(hid_t) **sphdf5::dset_id_success**
- integer(hid_t) **sphdf5::filespace_t**
- integer(hid_t) **sphdf5::filespace_s**
- integer(hid_t) **sphdf5::filespace_r**
- integer(hid_t) **sphdf5::filespace_z**
- integer(hid_t) **sphdf5::filespace_success**
- integer(hid_t) **sphdf5::memspace_t**
- integer(hid_t) **sphdf5::memspace_s**
- integer(hid_t) **sphdf5::memspace_r**
- integer(hid_t) **sphdf5::memspace_z**
- integer(hid_t) **sphdf5::memspace_success**
- integer(hid_t) **sphdf5::grptransform**
- integer(hid_t) **sphdf5::dset_id_diotadxup**
- integer(hid_t) **sphdf5::dset_id_fiota**
- integer(hid_t) **sphdf5::filespace_diotadxup**
- integer(hid_t) **sphdf5::filespace_fiota**
- integer(hid_t) **sphdf5::memspace_diotadxup**
- integer(hid_t) **sphdf5::memspace_fiota**
- character(len=15), parameter **sphdf5::aname** = "description"
- integer(hid_t) **sphdf5::attr_id**

- integer(hid_t) **sphdf5::aspace_id**
- integer(hid_t) **sphdf5::atype_id**
- integer, parameter **sphdf5::arank** = 1
- integer(hsize_t), dimension(arank) **sphdf5::adims** = (/1/)
- integer(size_t) **sphdf5::attrlen**
- character(len=:), allocatable **sphdf5::attr_data**

### 11.37.1  Detailed Description

(output) ! Writes all the output information to ext.h5.

### 11.37.2  Function/Subroutine Documentation

#### 11.37.2.1  init_flt_output()   subroutine sphdf5::init_flt_output (
            integer, intent(in) *numTrajTotal* )

init field line tracing output group and create array datasets

**Parameters**

| in | *numTrajTotal* | total number of Poincare trajectories |
|---|---|---|

References allglobal::mvol, inputlist::nppts, and allglobal::nz.

#### 11.37.2.2  write_poincare()   subroutine sphdf5::write_poincare (
            integer, intent(in) *offset,*
            real, dimension(:,:,:), intent(in) *data,*
            integer, dimension(:), intent(in) *success* )

write a hyperslab of Poincare data

**Parameters**

| *offset* | |
|---|---|
| *data* | |
| *success* | |

References inputlist::nppts, and allglobal::nz.

#### 11.37.2.3  write_transform()   subroutine sphdf5::write_transform (
            integer, intent(in) *offset,*

```
            integer, intent(in) length,
            integer, intent(in) lvol,
            real, dimension(:), intent(in) diotadxup,
            real, dimension(:,:), intent(in) fiota )
```

write rotational transform output from field line following

**Parameters**

| offset | |
|--------|---|
| length | |
| lvol | |
| diotadxup | |
| fiota | |

**11.37.2.4   write_vector_potential()**   subroutine sphdf5::write_vector_potential (
            integer, intent(in) *sumLrad,*
            real, dimension(:,:), intent(in) *allAte,*
            real, dimension(:,:), intent(in) *allAze,*
            real, dimension(:,:), intent(in) *allAto,*
            real, dimension(:,:), intent(in) *allAzo* )

write the magnetic vector potential Fourier harmonics to the output file group /vector_potential

**Parameters**

| sumLrad | |
|---------|---|
| allAte | |
| allAze | |
| allAto | |
| allAzo | |

## 11.38   spsint.f90 File Reference

(integrals) ! Calculates volume integrals of Chebyshev-polynomials and metric elements for preconditioner.

**Functions/Subroutines**

- subroutine [spsint](lquad, mn, lvol, lrad)

  *Calculates volume integrals of Chebyshev-polynomials and metric elements for preconditioner.*

### 11.38.1   Detailed Description

(integrals) ! Calculates volume integrals of Chebyshev-polynomials and metric elements for preconditioner.

### 11.38.2 Function/Subroutine Documentation

**11.38.2.1 spsint()** `subroutine spsint (`
          `integer, intent(in) ` *lquad,*
          `integer, intent(in) ` *mn,*
          `integer, intent(in) ` *lvol,*
          `integer, intent(in) ` *lrad* `)`

Calculates volume integrals of Chebyshev-polynomials and metric elements for preconditioner.

**Parameters**

| *lquad* | |
|---------|--|
| *mn* | |
| *lvol* | |
| *lrad* | |

References allglobal::cpus, allglobal::ddttcc, allglobal::ddttcs, allglobal::ddttsc, allglobal::ddttss, allglobal↩
::ddtzcc, allglobal::ddtzcs, allglobal::ddtzsc, allglobal::ddtzss, allglobal::ddzzcc, allglobal::ddzzcs, allglobal↩
::ddzzsc, allglobal::ddzzss, allglobal::dtoocc, allglobal::dtoocs, allglobal::dtoosc, allglobal::dtooss, allglobal↩
::gaussianabscissae, allglobal::gaussianweight, get_cheby(), get_zernike(), allglobal::guvijsave, constants::half,
allglobal::im, allglobal::in, allglobal::ki, allglobal::kija, allglobal::kijs, allglobal::lcoordinatesingularity, allglobal::mne,
inputlist::mpol, allglobal::mvol, allglobal::myid, allglobal::ncpu, allglobal::notstellsym, constants::one, fileunits::ounit,
constants::pi, constants::pi2, allglobal::pi2pi2nfp, allglobal::pi2pi2nfphalf, allglobal::regumm, numerical::small,
numerical::sqrtmachprec, allglobal::tdstcc, allglobal::tdstcs, allglobal::tdstsc, allglobal::tdstss, allglobal::tdszcc,
allglobal::tdszcs, allglobal::tdszsc, allglobal::tdszss, allglobal::ttsscc, allglobal::ttsscs, allglobal::ttsssc, allglobal↩
::ttsssss, constants::two, numerical::vsmall, inputlist::wmacros, allglobal::yesstellsym, and constants::zero.

Referenced by dfp100().

Here is the call graph for this function:
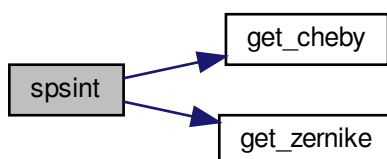
Here is the caller graph for this function:
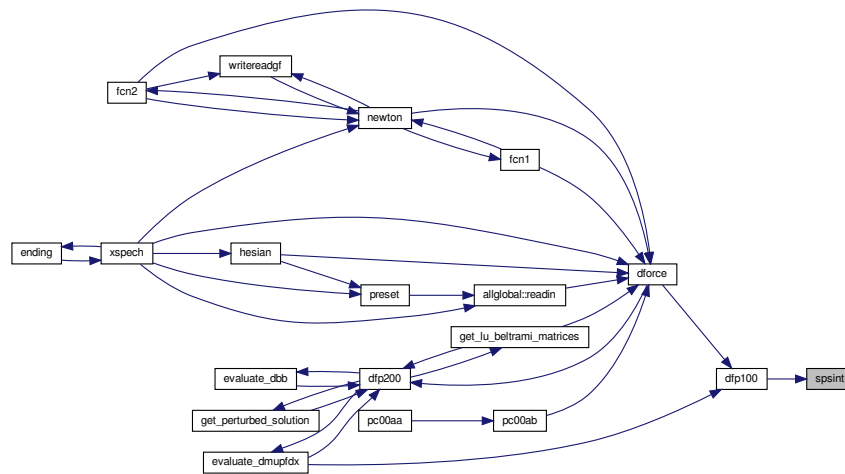


## 11.39   spsmat.f90 File Reference

(build matrices) ! Constructs matrices for the precondtioner.

### Functions/Subroutines

- subroutine **spsmat** (lvol, mn, lrad)

  *Constructs matrices for the precondtioner.*
- subroutine **push_back** (iq, nq, NN, vA, vD, vjA, qA, qD, qjA)

  *push a new element at the back of the queue*
- subroutine **clean_queue** (nq, NN, qA, qD, qjA)

  *clean the queue*
- subroutine **addline** (nq, NN, qA, qD, qjA, ns, nrow, dMAS, dMDS, jdMAS, idMAS)

  *add the content from the queue to the real matrices*

### 11.39.1   Detailed Description

(build matrices) ! Constructs matrices for the precondtioner.

### 11.39.2   Function/Subroutine Documentation

**11.39.2.1   spsmat()**  subroutine spsmat (
        integer, intent(in) *lvol,*
        integer, intent(in) *mn,*
        integer, intent(in) *lrad* )

Constructs matrices for the precondtioner.

**Parameters**

| *lvol* | |
|--------|--|
| *mn* | |
| *lrad* | |

References addline(), clean_queue(), allglobal::cpus, allglobal::dma, allglobal::dmas, allglobal::dmb, allglobal::dmd, allglobal::dmds, allglobal::dmg, allglobal::idmas, allglobal::im, allglobal::in, allglobal::jdmas, allglobal::liluprecond, inputlist::mpol, allglobal::myid, allglobal::nadof, allglobal::ncpu, allglobal::ndmas, allglobal::ndmasmax, allglobal↩ ::notstellsym, constants::one, fileunits::ounit, push_back(), numerical::small, constants::two, inputlist::wmacros, allglobal::yesstellsym, and constants::zero.

Referenced by dfp100().

Here is the call graph for this function:
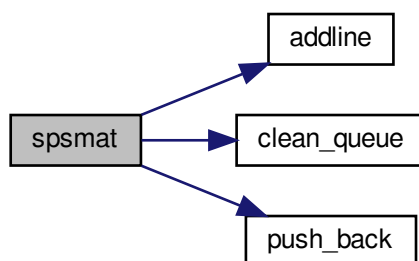


Here is the caller graph for this function:
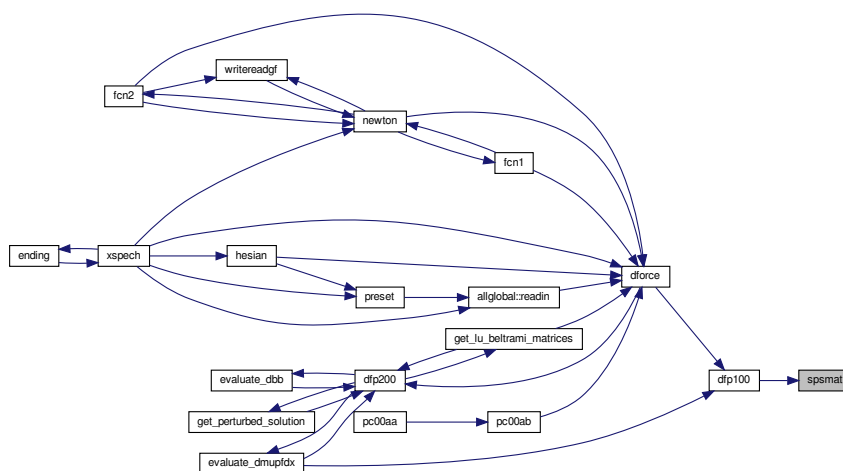
**11.39.2.2   push_back()** `subroutine push_back (`
       `integer, intent(in) iq,`
       `integer, dimension(4), intent(inout) nq,`
       `integer, intent(in) NN,`
       `real, intent(in) vA,`
       `real, intent(in) vD,`
       `integer, intent(in) vjA,`
       `real, dimension(nn,4), intent(inout) qA,`
       `real, dimension(nn,4), intent(inout) qD,`
       `integer, dimension(nn,4), intent(inout) qjA )`

push a new element at the back of the queue

**Parameters**

| | |
|-----|---|
| *iq* | |
| *nq* | |
| *NN* | |
| *vA* | |
| *vD* | |
| *vjA* | |
| *qA* | |
| *qD* | |
| *qjA* | |

References constants::zero.

Referenced by spsmat().

Here is the caller graph for this function:
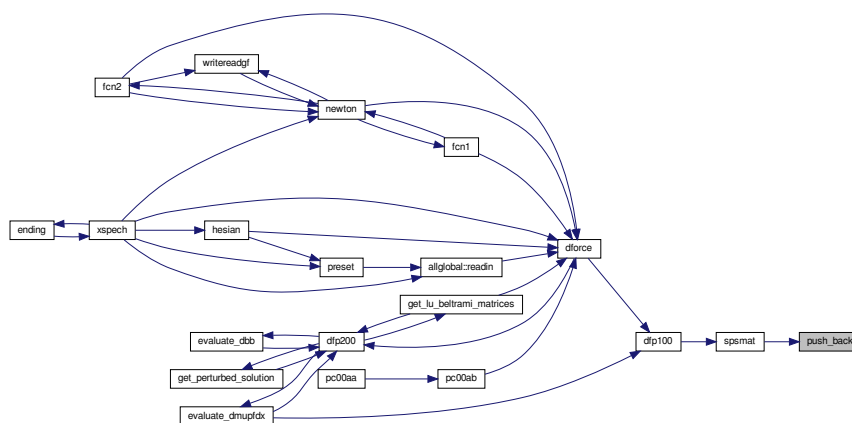


**11.39.2.3   clean_queue()** `subroutine clean_queue (`
       `integer, dimension(4), intent(inout) nq,`
       `integer, intent(in) NN,`

```
        real, dimension(nn,4), intent(inout) qA,
        real, dimension(nn,4), intent(inout) qD,
        integer, dimension(nn,4), intent(inout) qjA )
```

clean the queue

**Parameters**

| nq | |
|---|---|
| NN | |
| qA | |
| qD | |
| qjA | |

References constants::zero.

Referenced by spsmat().

Here is the caller graph for this function:
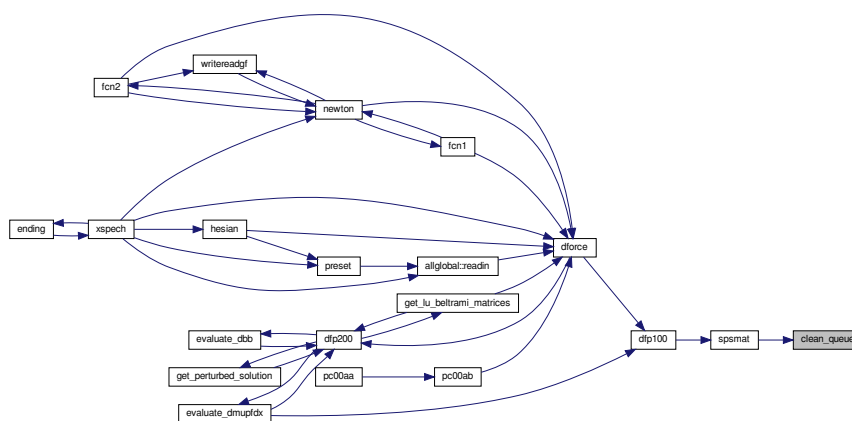


**11.39.2.4 addline()** `subroutine addline (`
```
        integer, dimension(4), intent(inout) nq,
        integer, intent(inout) NN,
        real, dimension(nn,4), intent(inout) qA,
        real, dimension(nn,4), intent(inout) qD,
        integer, dimension(nn,4), intent(inout) qjA,
        integer, intent(inout) ns,
        integer, intent(inout) nrow,
        real, dimension(*) dMAS,
        real, dimension(*) dMDS,
        integer, dimension(*) jdMAS,
        integer, dimension(*) idMAS )
```

add the content from the queue to the real matrices

**Parameters**

| nq | |
|-------|---|
| NN | |
| qA | |
| qD | |
| qjA | |
| ns | |
| nrow | |
| dMAS | |
| dMDS | |
| jdMAS | |
| idMAS | |

Referenced by spsmat().

Here is the caller graph for this function:
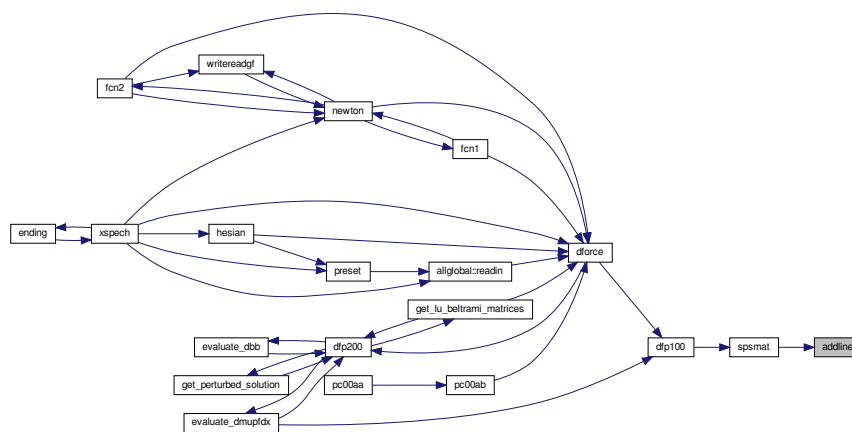


## 11.40  stzxyz.f90 File Reference

Calculates coordinates, $\mathbf{x}(s, \theta, \zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, at given $(s, \theta, \zeta)$.

**Functions/Subroutines**

- subroutine stzxyz (lvol, stz, RpZ)

  *Calculates coordinates, $\mathbf{x}(s, \theta, \zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, at given $(s, \theta, \zeta)$.*

### 11.40.1  Detailed Description

Calculates coordinates, $\mathbf{x}(s, \theta, \zeta) \equiv R\,\mathbf{e}_R + Z\,\mathbf{e}_Z$, and metrics, at given $(s, \theta, \zeta)$.

## 11.41 tr00ab.f90 File Reference

Calculates rotational transform given an arbitrary tangential field.

**Functions/Subroutines**

- subroutine tr00ab (lvol, mn, NN, Nt, Nz, iflag, ldiota)

  *Calculates rotational transform given an arbitrary tangential field.*

### 11.41.1 Detailed Description

Calculates rotational transform given an arbitrary tangential field.

## 11.42 volume.f90 File Reference

Computes volume of each region; and, if required, the derivatives of the volume with respect to the interface geometry.

**Functions/Subroutines**

- subroutine volume (lvol, vflag)

  *Computes volume of each region; and, if required, the derivatives of the volume with respect to the interface geometry.*

### 11.42.1 Detailed Description

Computes volume of each region; and, if required, the derivatives of the volume with respect to the interface geometry.

## 11.43 wa00aa.f90 File Reference

Constructs smooth approximation to wall.

**Functions/Subroutines**

- subroutine wa00aa (iwa00aa)

  *Constructs smooth approximation to wall.*
- subroutine vacuumphi (Nconstraints, rho, fvec, iflag)

  *Compute vacuum magnetic scalar potential (?)*

**Variables**

- logical [laplaces::stage1](#)
    *what is this ?*
- logical [laplaces::exterior](#)
    *what is this ?*
- logical [laplaces::dorm](#)
    *what is this ?*
- integer [laplaces::nintervals](#)
    *what is this ?*
- integer [laplaces::nsegments](#)
    *what is this ?*
- integer [laplaces::ic](#)
    *what is this ?*
- integer [laplaces::np4](#)
    *what is this ?*
- integer [laplaces::np1](#)
    *what is this ?*
- integer, dimension(:), allocatable [laplaces::icint](#)
    *what is this ?*
- real [laplaces::originalalpha](#)
    *what is this ?*
- real, dimension(:), allocatable [laplaces::xpoly](#)
    *what is this ?*
- real, dimension(:), allocatable [laplaces::ypoly](#)
    *what is this ?*
- real, dimension(:), allocatable [laplaces::phi](#)
    *what is this ?*
- real, dimension(:), allocatable [laplaces::phid](#)
    *what is this ?*
- real, dimension(:,:), allocatable [laplaces::cc](#)
    *what is this ?*
- integer [laplaces::ilength](#)
    *what is this ?*
- real [laplaces::totallength](#)
    *what is this ?*
- integer [laplaces::niterations](#)
    *counter; eventually redundant; 24 Oct 12;*
- integer [laplaces::iangle](#)
    *angle ; eventually redundant; 24 Oct 12;*
- real [laplaces::rmid](#)
    *used to define local polar coordinate; eventually redundant; 24 Oct 12;*
- real [laplaces::zmid](#)
    *used to define local polar coordinate; eventually redundant; 24 Oct 12;*
- real [laplaces::alpha](#)
    *eventually redundant; 24 Oct 12;*

### 11.43.1  Detailed Description

Constructs smooth approximation to wall.

## 11.44 xspech.f90 File Reference

Main program.

**Functions/Subroutines**

- program xspech

    *Main program of SPEC.*

- subroutine ending

    **todo remark**

### 11.44.1 Detailed Description

Main program.

### 11.44.2 Function/Subroutine Documentation

#### 11.44.2.1 xspech() `program xspech`

Main program of SPEC.

**Returns**

References inputlist::adiabatic, allglobal::ate, allglobal::ato, allglobal::aze, allglobal::azo, allglobal::bbe, allglobal←
::bbo, bnorml(), allglobal::btemn, allglobal::btomn, allglobal::bzemn, allglobal::bzomn, allglobal::cfmn, allglobal←
::cpus, dforce(), allglobal::dpflux, allglobal::dtflux, allglobal::efmn, ending(), inputlist::ext, allglobal::forceerr,
inputlist::gamma, inputlist::gbnbld, inputlist::gbntol, inputlist::helicity, hesian(), allglobal::ibnc, allglobal::ibns,
inputlist::igeometry, allglobal::iie, allglobal::iio, allglobal::im, allglobal::imagneticok, allglobal::in, allglobal←
::iquad, allglobal::irbc, allglobal::irbs, inputlist::isurf, allglobal::ivnc, allglobal::ivns, inputlist::ivolume, allglobal←
::izbc, allglobal::izbs, jo00aa(), inputlist::ladiabatic, inputlist::lautoinitbn, lbpol(), inputlist::lcheck, inputlist←
::lconstraint, allglobal::lcoordinatesingularity, inputlist::lfindzero, inputlist::lfreebound, allglobal::lgdof, inputlist←
::lhevalues, inputlist::lhevectors, inputlist::lhmatrix, numerical::logtolerance, inputlist::lperturbed, allglobal←
::lplasmaregion, inputlist::lrad, fileunits::lunit, allglobal::lvacuumregion, inputlist::lzerovac, inputlist::mfreeits,
inputlist::mu, constants::mu0, allglobal::mvol, allglobal::myid, allglobal::ncpu, newton(), inputlist::nfp, allglobal←
::ngdof, allglobal::notstellsym, inputlist::nppts, inputlist::nptrj, inputlist::nvol, inputlist::odetol, allglobal::ofmn,
constants::one, fileunits::ounit, packxi(), inputlist::pflux, inputlist::phiedge, constants::pi2, pp00aa(), preset(),
inputlist::pressure, inputlist::pscale, ra00aa(), inputlist::rbc, inputlist::rbs, allglobal::readin(), allglobal::sfmn,
inputlist::tflux, inputlist::vcasingtol, volume(), numerical::vsmall, allglobal::vvolume, inputlist::wmacros, allglobal←
::wrtend(), allglobal::yesstellsym, inputlist::zbc, inputlist::zbs, and constants::zero.

Referenced by ending().

Here is the call graph for this function:
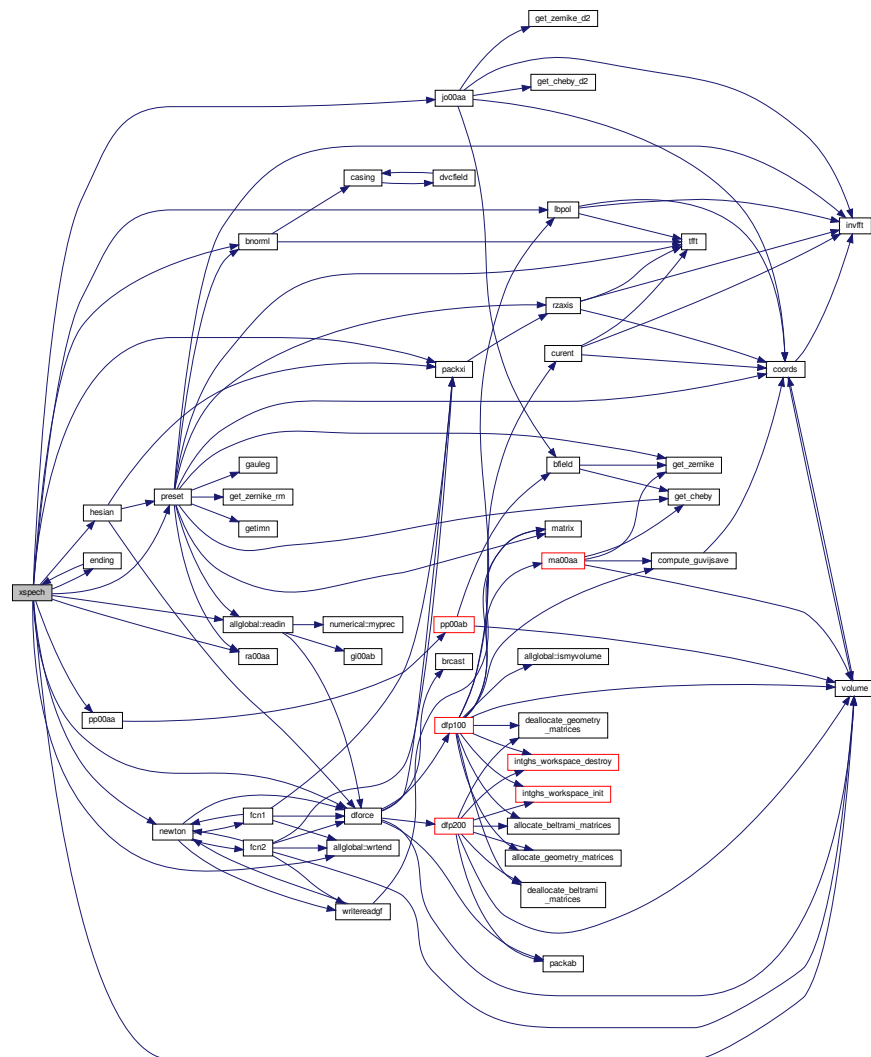


Here is the caller graph for this function:



**11.44.2.2    ending()**  `subroutine ending`

**todo remark**

**Todo** The following belongs to the docs of the program xspech, not to the ending() subroutine. If you know how to attach the docs to the program xspech, please fix this.

### reading input, allocating global variables

- The input namelists and geometry are read in via a call to readin() . A full description of the required input is given in global.f90 .

- Most internal variables, global memory etc., are allocated in preset() .

- All quantities in the input file are mirrored into the output file's group /input .

### preparing output file group iterations

- The group /iterations is created in the output file. This group contains the interface geometry at each iteration, which is useful for constructing movies illustrating the convergence. The data structure in use is an unlimited array of the following compound datatype:

```
DATATYPE  H5T_COMPOUND {
      H5T_NATIVE_INTEGER "nDcalls";
      H5T_NATIVE_DOUBLE "Energy";
      H5T_NATIVE_DOUBLE "ForceErr";
      H5T_ARRAY { [Mvol+1][mn] H5T_NATIVE_DOUBLE } "iRbc";
      H5T_ARRAY { [Mvol+1][mn] H5T_NATIVE_DOUBLE } "iZbs";
      H5T_ARRAY { [Mvol+1][mn] H5T_NATIVE_DOUBLE } "iRbs";
      H5T_ARRAY { [Mvol+1][mn] H5T_NATIVE_DOUBLE } "iZbc";
}
```

### packing geometrical degrees-of-freedom into vector

- If NGdof.gt.0 , where NGdof counts the geometrical degrees-of-freedom, i.e. the $R_{bc}$, $Z_{bs}$, etc., then packxi() is called to "pack" the geometrical degrees-of-freedom into position(0:NGdof) .

### initialize adiabatic constants

- If Ladiabatic.eq.0 , then the "adiabatic constants" in each region, $P_v$, are calculated as

$$P_v \equiv p_v V_v^{\gamma}, \tag{294}$$

where $p_v \equiv$ pressure(vvol) , the volume $V_v$ of each region is computed by volume() , and the adiabatic index $\gamma \equiv$ gamma .

### solving force-balance

- If there are geometrical degress of freedom, i.e. if NGdof.gt.0 , then

    - **Todo** If Lminimize.eq.1 , call pc00aa() to find minimum of energy functional using quasi-Newton, preconditioned conjugate gradient method, E04DGF

    - If Lfindzero.gt.0 , call newton() to find extremum of constrained energy functional using a Newton method, C05PDF .

### post diagnostics

- The pressure is computed from the adiabatic constants from Eqn. (294), i.e. $p = P/V^{\gamma}$.

- The Beltrami/vacuum fields in each region are re-calculated using dforce() .

- If `Lcheck.eq.5` .or. `LHevalues` .or. `LHevectors` .or. `Lperturbed.eq.1` , then the force-gradient matrix is examined using hesian() .

**free-boundary: re-computing normal field**

- If `Lfreebound.eq.1` and `Lfindzero.gt.0` and `mfreeits.ne.0` , then the magnetic field at the computational boundary produced by the plasma currents is computed using bnorml() .

- The "new" magnetic field at the computational boundary produced by the plasma currents is updated using a Picard scheme:

$$\mathrm{Bns}_i^j = \lambda\,\mathrm{Bns}_i^{j-1} + (1-\lambda)\mathrm{Bns}_i, \tag{295}$$

where $j$ labels free-boundary iterations, the "blending parameter" is $\lambda \equiv$ `gBnbld` , and `Bns` $_i$ is computed by virtual casing. The subscript "$i$" labels Fourier harmonics.

- If the new (unblended) normal field is *not* sufficiently close to the old normal field, as quantified by `gBntol` , then the free-boundary iterations continue. This is quantified by

$$\sum_i |\mathrm{Bns}_i^{j-1} - \mathrm{Bns}_i|/N, \tag{296}$$

where $N$ is the total number of Fourier harmonics.

- There are several choices that are available:

  - if `mfreeits=-2` : the vacuum magnetic field (really, the normal component of the field produced by the external currents at the computational boundary) required to hold the given equilibrium is written to file. This information is required as input by FOCUS [9] for example. (This option probably needs to revised.)

  - if `mfreeits=-1` : after the plasma field is computed by virtual casing, the vacuum magnetic field is set to exactly balance the plasma field (again, we are really talking about the normal component at the computational boundary.) This will ensure that the computational boundary itself if a flux surface of the total magnetic field.

  - if `mfreeits=0` : the plasma field at the computational boundary is not updated; no "free-boundary" iterations take place.

  - if `mfreeits>0` : the plasma field at the computational boundary is updated according to the above blending Eqn. (295), and the free-boundary iterations will continue until either the tolerance condition is met (see `gBntol` and Eqn. (296)) or the maximum number of free-boundary iterations, namely `mfreeits` , is reached. For this case, `Lzerovac` is relevant: if `Lzerovac=1` , then the vacuum field is set equal to the normal field at every iteration, which results in the computational boundary being a flux surface. (I am not sure if this is identical to setting `mfreeits=-1` ; the logic etc. needs to be revised.)

**output files: vector potential**

- The vector potential is written to file using ra00aa() .

**final diagnostics**

- sc00aa() is called to compute the covariant components of the magnetic field at the interfaces; these are related to the singular currents;

- if `Lcheck=1` , jo00aa() is called to compute the error in the Beltrami equation;

- pp00aa() is called to construct the Poincaré plot;

**restart files**

- wrtend() is called to write the restart files.

Closes output files, writes screen summary.

References allglobal::cpus, inputlist::ext, inputlist::ltiming, allglobal::myid, fileunits::ounit, cputiming::treadin, cputiming::twrtend, inputlist::wmacros, xspech(), and constants::zero.

Referenced by xspech().

Here is the call graph for this function:
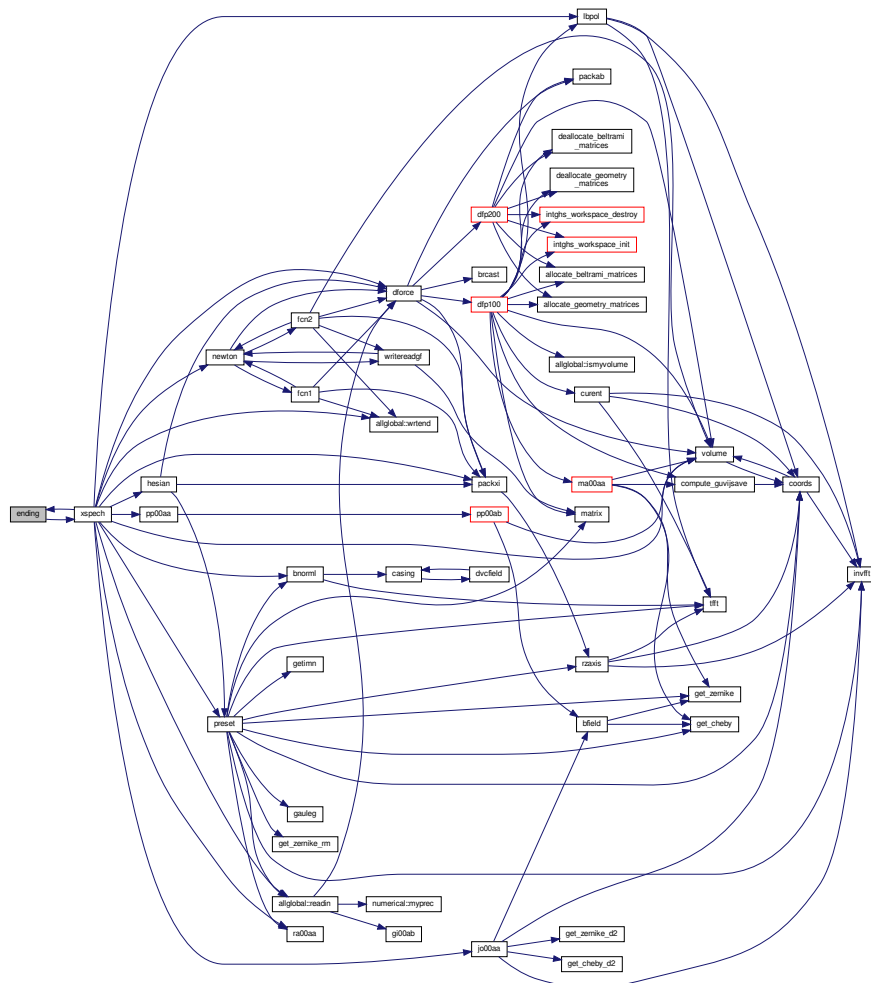


Here is the caller graph for this function:

# References

[1] J. D. Hanson. The virtual-casing principle and Helmholtz's theorem. *Plasma Phys. and Contr. Fusion*, 57(11):115006, sep 2015. 26

[2] S. P. Hirshman and J. Breslau. Explicit spectrally optimized Fourier series for nested magnetic surfaces. *Phys. Plas.*, 5(7):2664–2675, 1998. 46

[3] S. P. Hirshman and H. K. Meier. Optimized Fourier representations for three-dimensional magnetic surfaces. *Phys. Fluids*, 28(5):1387–1391, 1985. 46

[4] S.P. Hirshman, K. S. Perumalla, V. E. Lynch, and R. Sanchez. BCYCLIC: A parallel block tridiagonal matrix cyclic solver. *J. Comp. Phys.*, 229(18):6392 – 6404, 2010. 3

[5] S. R. Hudson, R. L. Dewar, M. J. Hole, and M. McGann. Non-axisymmetric, multi-region relaxed magnetohydro-dynamic equilibrium solutions. *Plasma Phys. and Contr. Fusion*, 54(1):014005, dec 2011. 17

[6] S. A. Lazerson. The virtual-casing principle for 3D toroidal systems. *Plasma Phys. and Contr. Fusion*, 54(12):122002, nov 2012. 26

[7] S. A. Lazerson, S. Sakakibara, and Y. Suzuki. A magnetic diagnostic code for 3D fusion equilibria. *Plasma Phys. and Contr. Fusion*, 55(2):025014, jan 2013. 137

[8] V. D. Shafranov and L. E. Zakharov. Use of the virtual-casing principle in calculating the containing magnetic field in toroidal plasma systems. *Nucl. Fusion*, 12(5):599–601, sep 1972. 26

[9] C. Zhu, S. R. Hudson, Y. Song, and Y. Wan. New method to design stellarator coils without the winding surface. *Nucl. Fusion*, 58(1):016008, nov 2017. 265

# Index