

**xspech**

main program

[called by: .] [calls: [global:readin](#), [global:wrtend](#), [preset](#), [packxi](#), [volume](#), [pc00aa](#), [newton](#), [dforce](#), [hesian](#), [ra00aa](#), [bnorml](#), [sc00aa](#), [jo00aa](#) and [pp00aa](#).]**contents**

<b>1 xspech</b>	<b>1</b>
1.1 reading input, allocating global variables . . . . .	1
1.2 preparing output file <code>.ext.sp.its</code> . . . . .	1
1.3 “packing” geometrical degrees-of-freedom into vector . . . . .	1
1.4 initialize adiabatic constants . . . . .	1
1.5 solving force-balance . . . . .	1
1.6 post diagnostics . . . . .	2
1.7 free-boundary: re-computing normal field . . . . .	2
1.8 output files: vector potential . . . . .	2
1.9 final diagnostics . . . . .	2
1.10 restart files . . . . .	2
1.11 subroutine : ending . . . . .	2

**1.1 reading input, allocating global variables**

1. The input namelists and geometry are read in via a call to [global:readin](#). A full description of the required input is given in [global](#).
2. Most internal variables, global memory etc., are allocated in [preset](#).

**1.2 preparing output file `.ext.sp.its`**

1. The file `.ext.sp.its` is created. This file contains the interface geometry at each iteration, which is useful for constructing movies illustrating the convergence. The format of this file is:

```

open(zunit,file="."//trim(ext)//".sp.its",status="unknown",form="unformatted")
write(zunit) mn, Mvol, Nfp ! integer;
write(zunit) im(1:mn) ! integer;
write(zunit) in(1:mn) ! integer;
! begin do loop over iterations;
write(zunit) wflag, iflag, Energy, rflag ! written in global.h:wrtend; perhaps redundant;
write(zunit) iRbc(1:mn,0:Mvol) ! real;
write(zunit) iZbs(1:mn,0:Mvol) ! real;
write(zunit) iRbs(1:mn,0:Mvol) ! real;
write(zunit) iZbc(1:mn,0:Mvol) ! real;
! end do loop over iterations;

```

**1.3 “packing” geometrical degrees-of-freedom into vector**

1. 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)`.

**1.4 initialize adiabatic constants**

1. If [Ladiabatic.eq.0](#), then the “adiabatic constants” in each region,  $P_v$ , are calculated as

$$P_v \equiv p_v V_v^\gamma, \quad (1)$$

where  $p_v \equiv$  [pressure\(vvol\)](#), the volume  $V_v$  of each region is computed by [volume](#), and the adiabatic index  $\gamma \equiv$  [gamma](#).

**1.5 solving force-balance**

1. If there are geometrical degress of freedom, i.e. if `NGdof.gt.0`, then

If [Lfindzero.gt.0](#), call [newton](#) to find extremum of constrained energy functional using a Newton method, [NAG: C05PDF](#);

## 1.6 post diagnostics

1. The pressure is computed from the adiabatic constants from Eqn.(1), i.e.  $p = P/V^\gamma$ .
2. The Beltrami/vacuum fields in each region are re-calculated using `dforce`.
3. If `Lcheck.eq.5 .or. LHevalues .or. LHevectors .or. Lperturbed.eq.1`, then the “Hessian” matrix is examined using `hesian`.

## 1.7 free-boundary: re-computing normal field

1. If `Lfindzero.gt.0` and `Lfreebound.eq.1` and `mfreeits.ne.0`, then the magnetic field at the computational boundary produced by the plasma currents is computed using `bnorml`.
2. The “new” magnetic field at the computational boundary produced by the plasma currents is updated using a Picard scheme:

$$(\mathbf{B} \cdot \mathbf{n})_{m,n}^{i+1} = \lambda(\mathbf{B} \cdot \mathbf{n})_{m,n}^i + (1 - \lambda)(\mathbf{B} \cdot \mathbf{n})_{m,n}, \quad (2)$$

where  $i$  labels free-boundary iterations, and  $\lambda \equiv \text{gBnbld}$ .

3. 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.

## 1.8 output files: vector potential

1. The vector potential is written to file using `ra00aa`.

## 1.9 final diagnostics

1. `sc00aa` is called to compute the covariant components of the magnetic field at the interfaces; these are related to the singular currents;
2. if `Lcheck = 1`, `jo00aa` is called to compute the error in the Beltrami equation;
3. `pp00aa` is called to construct the Poincaré plot;

## 1.10 restart files

1. `global:wrtend` is called to write the restart files.

## 1.11 subroutine : ending

1. Closes output files, writes screen summary.