TEQ library user manual

Johan Carlsson, Tech-X Corporation
teq-users@fusion.txcorp.com

$Id: usermanual.tex,v 1.3 2007/07/31 20:47:37 pearls Exp $

Introduction

The TEQ library solves the Grad-Shafranov equation to calculate the free-boundary (or direct) MHD equilibrium. It can also calculate the fixed-boundary (or inverse) MHD equilibrium. The TEQ library is quite powerful and flexible.

The TEQ module depends on the separatrix module and on LAPACK. The latest versions can be downloaded from:

- `http://fusion.txcorp.com/~johan/teq.tgz`
- `http://fusion.txcorp.com/~johan/separatrix.tgz`
- `http://www.netlib.org/lapack/lapack.tgz`

Unpack `teq.tgz` and see the file `README` for build instructions.

The floating-point precision is determined by the kind parameter `rq`, which is defined in the file `utils.f90` in the separatrix module. By default `rq` is chosen to make `real(rq)` an 8-byte floating-point type. By changing line 7 of `utils.f90` from `rq=r8` to `rq=r4`, `real(rq)` becomes a 4-byte floating-point type instead.

Free-boundary equilibria can currently either be generated from scratch ("dead start") or loaded from ASCII TEQ save files. The "dead start" option can involve extensive user intervention. The ability to load free-boundary equilibria from EQDSK a- and g-files will be added in a future release.

Inverse equilibria can be loaded from ASCII TEQ save files or generated directly from a direct equilibrium.

The remainder of this manual is organized as follows: The user interface routines are listed on page 2, followed by a full description of the arguments, i.e., the inputs, outputs, and flags. The interface routines themselves are described in a bit more detail on page 8. Commented usage examples can be found in the "Examples" section beginning on page 9. A brief "History" is given on page 13, followed by a sample output plot. An Appendix on files for the dead-start procedure is on page 16.

Interface

The top-level subroutines are listed below. The catch in their names is an indication that they enable the user to recover from impending serious failure modes by executing a longjump to the calling point if `stop_on_error = .false.`. This is ilustrated in the demonstration driver.

(1) `call catch_dead_start(ierr,glb);`
as previously mentioned, this call starts up from configuration-specific input files and could require significant user intervention. Our policy has been to provide savefiles for new configurations so that a user can avoid this procedure.

(2) `call catch_load_savefile(ierr,glb,dir,pf,inv,prof);`
this call reads in a savefile and calculates the appropriate equilibrium.

(3) `call catch_inveq(ierr,inv,prof,glb);`
this call provides an inverse (prescribed boundary) equilibrium.

(4) `call catch_direq(ierr,dir,pf,inv,prof,glb);`
this call provides a free-boundary equilibrium. It is important to point out that the inverse equilibrium solver can be used to generate the plasma current on the (R, Z) mesh and then used by the free-boundary solver. At convergence the plasma boundary is self-consistent. Again, this is illustrated in the demonstration driver.

Here the main arguments are the derived types. The definitions (with comments) of the derived types follow, after which we describe what each routine does in some detail.

```
TYPE(profile_type)::prof
TYPE(inv_type) :: inv
TYPE(dir_type) ::dir
TYPE(pf_type) :: pf
TYPE(global_type) :: glb

TYPE :: global_type
   INTEGER, POINTER :: logunit,ioun7,ioun12,ioun13
   CHARACTER(len=64), POINTER :: logname
```

```
      CHARACTER(len=64) :: filename,tokamak,pfcoil
   END TYPE global_type

   TYPE :: eq_type
      REAL(rq), DIMENSION(:),   POINTER :: x,y
      REAL(rq), DIMENSION(:,:), POINTER :: f,fx,fy,fs
      INTEGER, POINTER :: nx,ny
   END TYPE eq_type

   TYPE :: inv_type
      TYPE(eq_type) :: R,Z
      REAL(rq), DIMENSION(:), POINTER :: rbnd,zbnd
      REAL(rq), POINTER :: raxis,zaxis,dpsi0,dpsi00,dphi0,dphi00,plc,fwall,cur
      REAL(rq), POINTER :: epsrk,x
      INTEGER, POINTER :: p,k,eq,nht,mpsi,mth,flag,ls
      LOGICAL, POINTER :: fix_bnd
! (R%x,R%y)     = (theta,psibar)
! (R%f,R%fx,R%fy,R%fs) = (R,R_theta,R_psi,B)
! (Z%x,Z%y)     = (theta,psibar)
! (Z%f,Z%fx,Z%fy,Z%fs) = (Z,Z_theta,Z_psi,Jacobian)
! (raxis,zaxis) = position of axis cm
! (dpsi0,dpsi00)= poloidal flux/radian; (dynamic,desired)
! (dphi0,dphi00)= toroidal flux/radian
! x             = position in dimensionless flux beyond which solver adjusts q
!                 indirectly so as to force FF'=0 at the edge.
! plc           = desired plasma current MA
! cur           = plasma current A
! fwall         = vacuum RBtor gauss-cm
! epsrk         = accuracy for polar1 (defaulted to 1e-6)
! k             = (0 use qs) (1 use fs) (2 use jts) (3 use jpars) as profile.
!               = For a time dependent (predictive mode) Ohm's law requires
!                 that the q-profile be inputed.
! p             = (-1 use dpsi00) (0 use fwall) (1 use plc) as constraint,
! eq            = -2: startup from direct solve, uses internal information,
!               = -1; vanilla flavored startup, uses no internal data, and is
!                     recommended for a "large" boundary change in a sequence
!                     of inverse solves. If this option fails with inv_k=0,
!                     q-profile as input, try a current profile. If it still
!                      fails try eq=0. If it still fails contact ldp@llnl.gov.
!                =  0; from an existing inverse solve, sets the boundary to
!                      (inv%rbnd,inv%zbnd); this is the recommended option if
!                      the boundary is moved.
!                =  1; from an existing inverse solve, but the boundary
!                      relaxes to (inv%rbnd,inv%zbnd); used for time dependent
!                      evolution predictive mode, not snap shots and is needed
!                      for numerical stability.
! nht            = maximum number of iterations
! mpsi           = flux grid size
! mth            = theta gridsize
! ls             = stretching in psi grid: ls=-22  (sin(i*pi/2))**2; i=1,mpsi
!                                            = 22 quadratic axi; polynomial
!                                            = 11  uniform
!                                            = -1=-22;1=11 anachroism
! flag           = 0, uses polar1; 1, uses esc if loaded (libesc.a)

  END TYPE inv_type

  TYPE :: profile_type
     REAL(rq), DIMENSION(:), POINTER :: q,jt,jpar,f,fd,p,pd,bp,rsqi,cu,phi,psi
     REAL(rq), DIMENSION(:), POINTER :: qs,jts,jpars,fs,ps,vpr,w,v,psin,r,a,bsq
! (q,qs)        = q-profile; (dynamic,desired)
! (jt,jts)      = <J.B>/(J,grad phi> amps/cm
! (jpar,jpars)  = J.B>/<B.B> amps/cm**2/gauss
! (f,fs)        = RBt gauss-cm
! fd            = df/dpsi 1/cm
! (p,ps)        = pressure ergs/cm**3
! pd            = dp/dpsi
! bp            = <Bpol> gauss
! bsq           = <B**2> qauss**2
! rsqi          = <1/R**2> 1/cm**2
! cu            = Mu0*I(psi) gauss-cm
! vpr           = dV/dpsi cm/gauss
! <w,v>         = node values for the two input spline profiles
!                 Spline at nodes F**2-F(msrf)**2)/(8*pi*dcpsi0*prsx)
!                 Spline at nodes P/(betaj*prsx)
! psin          = dimensionless poloidal flux: input
! (a,r)        = (minor radius, major radius) of flux surface at
!                 level of magnetic axis
! phi          = dimensionless toroidal flux, output
! psi          = dimensionless poloidal flux, input
! <>           = surface average
  END TYPE profile_type

  TYPE :: dir_type
     TYPE(eq_type) :: psi
     REAL(rq), DIMENSION(:), POINTER :: rxpr,zxpr,alfa,betp,li,betap
     REAL(rq), DIMENSION(:), POINTER :: rxps,zxps,pxps,rsep,zsep
     REAL(rq), POINTER :: rxpt,zxpt,pxpt,paxis,pbnd,rn,rx,ro,zx,prsx
     REAL(rq), POINTER :: betaj,thc,epsj,epsb,aap
     INTEGER,  POINTER :: ipscl,ipf,ipj,ipp,irl,liml,f_wp,nxps,limd,ixpt,msep
     LOGICAL,  POINTER :: fix_bnd
!
! (rsep,zsep)      = separatrix boundary
! (psi%f,psi%fx,psi%fy,psi%fs)=(psi,psi_r,psi_z,bpol)
! rxpr(2) zxpr(2)  = determines box to look for X-point cm
! (alfa(0:1),betp(0:1),aap)= profile factors for analytic forms: see ipp below
! (zxpt,rxpt,pxpt) = Xpoint location and flux/radian cm,cm,gauss-cm**2
! (paxis,pbnd      = poloidal flux at on (axis,boundary)
! (nxps,rxps,zxps,pxps) = number, flux, R and Z of Xpoints found by separatrix
! (rn,rx,zx)       = boundary of computational grid cm (up/down sym. grid)
! limd             = 0 not limited
! ro               = radial mid-point of grid (btor=fwall/ro)
! prsx             = scaling coefficient need for w,v the spline points
! betaj            = scaling factor multiplying the pressure for input forms
! thc              = fraction away from X-point in dimensionless poloidal flux
! ixpt             = 2 use Xpoint; = 1 if limiter is active use limiter
! ipscl            = constrain for dir solve (-1 none 0 plc 1 dpsi00 )
! ipj              = 0, analytic forms for FF' and P
!                    1  jts,ps arrays
!                    2  analytic forms for jt and p
!                    3  jpars and ps arrays for input (inv%p constraint choice)
!                       scaling (ipscl=-1 recommended)
!                    4  qs and ps arrays; scaling (see ipscl) not recommended
!                    5  qs and ps arrays; no scaling  (inv%p constraint choice)
! (ipf,ipp)         = profile forms for analytic; x dimensionless flux
!                   = 1; Strickler form exp(-alfa*x)-exp(-alfa) for (FF',P');
!                   = 2; difference of two; 2nd form uses alfa*betp
!                   = 3; (1-x**betp)**alfa for (FF,P)
!                   = 4; 3 form but flat out to aap;with betp*(x-aap)/(1-aap)
!                   = 5; hollow profile: constrained to  2<betp<2.5 and
!                        SENSITIVE near upper limit.
!                   = 999; cubic spline for (FF,P); uses (w,v)
! (epsj,epsb)       = residual accuracy required, (comp boundary, GS rhs)
  END TYPE dir_type

  TYPE :: pf_type
     INTEGER, POINTER :: ngroup,ncircuit,nbd,jcir,nfbd,nc,jax,kax,nlim,irl,ngp
     INTEGER, POINTER :: ircwt,nsym,nsymc,iflxc,limw,lmax,kxp
     INTEGER,  DIMENSION(:), POINTER :: nrc,nzc,ic,ix,ict
     REAL(rq), DIMENSION(:), POINTER :: cc,rc,zc,drc,dzc,ps0,psi
     REAL(rq), DIMENSION(:), POINTER :: pci,pc0,rlim,zlim,rlimw,zlimw
     REAL(rq), DIMENSION(:), POINTER :: rbd,zbd,rfbd,zfbd,alfbd
     REAL(rq), DIMENSION(:,:), POINTER :: pfim
     REAL(rq), POINTER :: psix,vltf,cejima,rl,rax,zax,skew,dsep,dsip,rxp,zxp
     REAL(rq), POINTER :: rmajor,rminor

! nc                = number of coils
! cc                = coil current, MA
! (rc,drc)          = coil R and width, M
! (zc,dzc)          = coil Z and width, M
! (nrc,nzc)         = number of filamants= nrc*nzc per coil; code generally
!                     controls
! (psi,ps0)         = flux at coil (calculated, wanted), Wb
! (ic,ix)           = grouping of (coils,circuits); if (ic(1)=n and ic(2)=n
!                     where n=1,2,3,4,... keeps the ratio of current in
!                     cc(1) and cc(2) fixed;similarly for ix and the coil flux.
!                     note that max(ic) must be greater or equal to max(ix)
!                     depending on whether there are additional constraints.
! iflxc             = 1 forces psi to ps0 for all ix/=0
! ngroup            = number of coil groups, max(ic)
! ncircuit          = number of circuits, max(ix)
! (nbd,rbd,zbd)     = number of boundary points and values in cm's, constraints
! psix              = desired flux at magnetic axis, constraint (units as paxis)
! vltf              = desires external flux linkage (VS), constraint
! cejima            = Ejima coefficient, constraint;vltf=psires +
!                     0.4*pi*cejima*plcm*ro/100, with  psires the average
!                     plasma poloidal flux.
! (rl,irl)          = irl=0,  required separation between active and inactive
!                     separatrix in dimensionless flux:(pxpu-pxpl)/dpsi0
!                   = (irl=1,upper Xpoint active);(irl=-1, lower Xpoint active);
!                     rl is the required separation between inactive and active
!                     separatrix in cm’s (>0 outside. <0 inside)
! (kxp,rxp,zxp)       kxp/=0 X-point forces to (rxp,zxp)
! (kax,rax,zax)       kax/=0 O-point forces to (rax,zax): problematic  since
!                     requires user ingenuity to avoid over-constrainingsing;
!                     can also be another X-point
! jcir              = total number of constraints
! (nfbd,alfbd)      = number of fuzzy boundary points and normalization
! (rfbd,zfbd)       = value of points in cm’s not as a minimization
! ircwt             = -1 minimizes (dI/dz)^2;all coils and in sequence.
!                   =  1 minimizes (cc-cc0)!(cc-cc0)
!                   =  2 minimizes cc!cc
!                   =  3 minimizes j**2=(cc/drc/dzc)^2
!                   =  4 minimizes cc!pfim*!cc=I!L!I
!                   =  5 minimizes B!B at the coils
!                   =  6 minimizes (pfpsi-pfps0)^2;change in flux at coils
!                   =  7 minimizes pfpsi!cc
!                   = 11 minimizes (cc*rc)!(cc*rc)
! (rlim(0),zlim(0) = limiter point
! skew              = angle between flux aurface axis and vertical axis
! rminor            = minor radius
! rmajor            = major radius
! dsep              = distance bewtween separatrices at outboard midplane
! dsip              = distance bewtween separatrices at inboard midplane
! (nsym,nsymc;lmax)= (2,2), up/down asymmetric; lmax=mth
!                   = (1,1), up/down symmetrix;  lmax=mth/2+1
! (rlimw,zlimw)     = limiting wall; dimensioned lmax and must be continous.
!                     (rlim(0),zlim(0)) is set to the limiting point if limw=1

  END TYPE pf_type
```

To do a dead start (generate a free-boundary equilibrium from scratch) call the subroutine `catch_dead_start(ierr,glb)`. Here the files `"glb%tokamp"` and `"glb%coils"` contain plasma parameters and coil configuration, respectively. The file formats are described in Appendix A.

The TEQ module also comes with a set of ASCII TEQ save files with input data from pre-calculated equilibria for the major US machines (DIII-D, C-Mod, NSTX and PEGASUS) and some other ones too (JET, KSTAR, ITER, etc.). These equilibria are generated by typing `make test` and are used by the demonstration driver program. This same driver program also contains many examples of changing inputs to obtain new equilibria. In addition it shows how to obtain an inverse equilibrium from an existing free-boundary equilibrium. Users should resort to a dead start only when generating a totally new configuration; it should always be sufficient to start with an equilibrium from one of these supplied files or from a savefile generated by the user as described next.

The very first equilibrium is generated by calling `catch_load_file` which automatically generates the equilibrium contained in the savefile. To write a save file once an equilibrium has been generated, call the subroutine `write_savefile("foo.in")`, which outputs `foo.in`. It is the user's responsibility to first properly initialize all the input variables by calling `catch_load_savefile(ierr,dir,pf,inv,prof,glb)`, where `glb%filename` is the name of a TEQ savefile. The files in the distribution have a `.in` appended to them and are in the teq/input directory. (See the example below.)

A subsequent direct solve, after input values have been changed for a free-boundary equilibrium, is obtained by calling the subroutine `catch_direq(ierr,dir,pf,inv,prof,glb)`. For extensive examples examine `driver.f90`.

An inverse solve, for the fixed-boundary (prescribed-boundary) equilibrium, is obtained by calling the subroutine `catch_inveq(ierr,inv,prof,glb)`. As before, this call is made after an initial inverse equilibrium has been computed and input values are then changed. Note that it is also possible to obtain an inverse solve from an existing direct solve; but first it is necessary to move the plasma boundary away from an existing separatrix. This is done by setting `dir%thc` to a non-zero value and recalculate the direct solve. Next call the subroutine `catch_inveq(ierr,inv,prof,glb)`. The user should examine the detailed comments in `inv_type`. To handle the common situation where the user specifies input using a flux coordinate different from the one used internally in TEQ, the subroutine `translate(psuser,qsuser,prof%psi,prof%qs)` is provided. Here `psuser` must be the user-prescribed dimensionless poloidal flux and `qsuser` is the user-prescribed q-profile. Clearly the inverse transformation is obtained by inverting the order of the arguments.

The user can also interact with the TEQ library by directly setting the value of the variables listed in the module `teqlib_input`. The `teqlib_input` module variables are described by extensive, but not yet complete, comments in the file `teqinit.f90`. The same is true of the varaibles in the module `teqlib_output`. It is our intention that the derived types contain all necessary input and output variables; but in the event that this is not the case the above variables can be accessed. In any case, if users need direct access, please let us know so that the derived type list can be extended.

Examples

In this section we'll walk through and comment on selected parts of the demonstration driver program in `driver.f90`. The TEQ library is accessed through the module `teqlib`. To use the dead-start procedure it is also necessary to use the module `deadstart`. To access the catch routines, which the users should look upon as a requirement, it is necessary to use the module `teqerr`:

```
program driver

  use teqlib
  use deadstart
  use teqerr
```

Next, we load a TEQ save file with an ITER equilibrium and calculate the equilibrium:

```
integer :: ierr=0
character(len = 64) :: which
glp\%filename="iter.sav
call catch_load_savefile(ierr,glb,dir,pf,inv,prof)
if(ierr/=0) then ! It is in this if block that the user can intervene.
  print *, 'An exception occurred, the error message is: ', &
        trim(error_msg), '. Bailing out...'
  stop
end if
```

Calling `set_verbosity(level)` determines how much output will be printed: more for larger values of `level`; zero suppresses all output.

```
call set_verbosity(0)

      inv%mth=81    ! number of points around a flux surface
      inv%mpsi=81   ! number of flux surfaces
      dir%thc = 0.01! move boundary inside separatrix in dimensionless flux;
                    ! necessary for inverse solves and direct solves if q is
                    ! is the inputted profile: ipj=5
      call catch_direq(ierr,dir,pf,inv,prof,glb)
      if(ierr/=0) then
         print *, 'An exception occurred, the error message is: ', &
              trim(error_msg), '. Bailing out...'
         ! user intervenes
         stop
      end if
      ! this option is the "adiabatic'' option
      dir%ipj = 5   ! compute from qsave and psave
      dir%ipscl = -1! do not scale to fix current or flux
      inv%p = 0     ! fixes F on boundary to fwall
                    ! note that in this case the options are the same
                    ! as for the inverse solver
      dir%f_wp=999  ! smoothes current at edge, necessary for dir%ipj=5
      inv%eq=0      ! in case the starting direct solver was using the
                    ! inverse solver for the solution inside the separatrix
      call catch_direq(ierr,dir,pf,inv,prof,glb)
      if(ierr/=0) then
         print *, 'An exception occurred, the error message is: ', &
```

```
                trim(error_msg), '. Bailing out...'
           ! user intervenes
           stop
        end if
```

Next we double the resolution in both the R- and Z-directions and re-solve:

```
        which="both"
        call gridup(dir,pf,inv,prof,glb,which) ! changes grid
        call catch_direq(ierr,dir,pf,inv,prof,glb)
        if(ierr/=0) then
           print *, 'An exception occurred, the error message is: ', &
                trim(error_msg), '. Bailing out...'
           ! user intervenes
           stop
        end if
```

After re-setting the resolution, we set up some parameters that control the inverse solve before calling `catch_inveq(ierr,inv,prof,glb)` to calculate the inverse solution. First, reset the resolution:

```
        which="both"
        call griddown(dir,pf,inv,prof,glb,which) ! changes grid
        call catch_direq(ierr,dir,pf,inv,prof,glb)
        if(ierr/=0) then
           print *, 'An exception occurred, the error message is: ', &
                trim(error_msg), '. Bailing out...'
           ! user intervenes
           stop
        end if
```

Then calculate an inverse equilibrium:

```
        inv%nht = 200 ! maximum number of iterations
inv%epsrk = 1.0e-6 ! required accuracy
        call set_verbosity(2) ! iteration output
        inv%p = 0     ! fixes F_bound to fwall
        inv%k = 0     ! uses q-profile (qsave)
        inv%eq=-2     ! first time after direct equilibrium, uses internal data
        call catch_inveq(ierr,inv,prof,glb) ! first inverse solve
        if(ierr/=0) then
           print *, 'An exception occurred, the error message is: ', &
                trim(error_msg), '. Bailing out...'
           ! user intervenes
           stop
        end if
```

Finally the direct (free-boundary) or the inverse (fixed-boundary) solutions are dumped to a TEQ save file:

```
  call write_savefile("driver.in")
```

There is also an option, by setting `do_test=.true.` in `driver.f90`, to just run a test on the chosen savefile. This option compares a selection of input values from the savefile to output values from the equilibrium computation. For these tests the differences need only be "reasonable" since the savefiles may not be up-to-date or different operating systems can result in different arithmetic roundoff. (During development, tests are run outside of the module in CORSICA, and results are required to be preserved to machine accuracy. Of course, in the event of physics changes, the output can change. At this point the reference comparisions are updated.) Typical output from this test follows:

```
 FOR A FIXED-BOUNDARY SOLVE
 OUTPUT
 q(1) and P(1)                 1.788384140457189        82977.31217853323
 boundary point                204.2538029631154        72.67016035265463
 scaler constraints -dpsi0,cur,fwall- two can be bad since only one is used
   -15652756.49175006          501961.4181976991        3421697.351372659
 INPUT
 q(1) and P(1)                 1.783038640755982        82982.90198660560
 boundary point                204.2541219352515        72.67004163460653
 scaler constraints -dpsi0,cur,fwall- two can be bad since only one is used
   -15652697.76714600          501465.0999999830        3421697.027499858

FOR A FREE-BOUNDARY SOLVE
```

```
OUTPUT
coil currents and pbnd       5.438765524497730        -2.558724001046558
  -6.303532171813806        -4.798250513593817        -7.639674164271554
   17.19991842458036         2.452762230532624        -9.123417851137118
  -20.55179345418781       -20.55179345418781        -11.59032227682597
   2.022765962500211        -48862976.54228014
boundary point               726.6345712382119         291.4429930457409
scaler constraints -dpsi0,cur,fwall- two can be bad since only one is used
  -1296164840.520250         15017824.62734537         32907637.91750462
INPUT
coil currents and pbnd       5.437318955712755        -2.558168310956238
  -6.303835709121671        -4.798670885811258        -7.638831962678855
   17.19868055816492         2.448884347692295        -9.119531194828678
  -20.55424072858904       -20.55424072858904        -11.58888405527327
   2.022972687107095        -48951218.32818943
boundary point               726.6350935926482         291.4434961957248
scaler constraints -dpsi0,cur,fwall- two can be bad since only one is used
  -1296950648.964848         15000000.00000000         32860000.00000000
```

Note that the inverse solver is on a different grid from the input/output grid; hence there will be grid-error differences. Also, for more details with regard to the driver, look at `driver.f90`. A plot of the direct solution from the file `driver.in`, generated by the the IDL script `teqplot.pro`, is shown in Fig. 1.

It is of course also possible to selectively dump chosen output arrays. The most commonly used fixed-boundary output arrays are probably the profiles: `prof%f` (F), `prof%fd` (F'), `prof%q` (q), `prof%jt` (J_ϕ), `prof%jpar` ($J_\|$), `prof%p` (p) and `prof%pd` (p'). Prime denotes the derivative with respect to the stream function $\psi = \psi_p/2\pi$ (poloidal flux over 2π).

History

The source code for the TEQ library was extracted from the Corsica code by L. Donald Pearlstein and Richard Bulmer, LLNL, and Johan Carlsson, Tech-X Corporation, and converted from Fortran77/Basis to Fortran90. The module was first released to users July 2005 (submission to the NTCC Modules Library awaited documentation).

REVISION HISTORY

`TEQ_1_1_0`. July 2007. Expanded the documentation of fixed-boundary and added free-boundary: added a comprehensive list of input and output variables and defining comments. There are minor changes in the fortran. This version is the first submitted to the NTCC Modules Library.

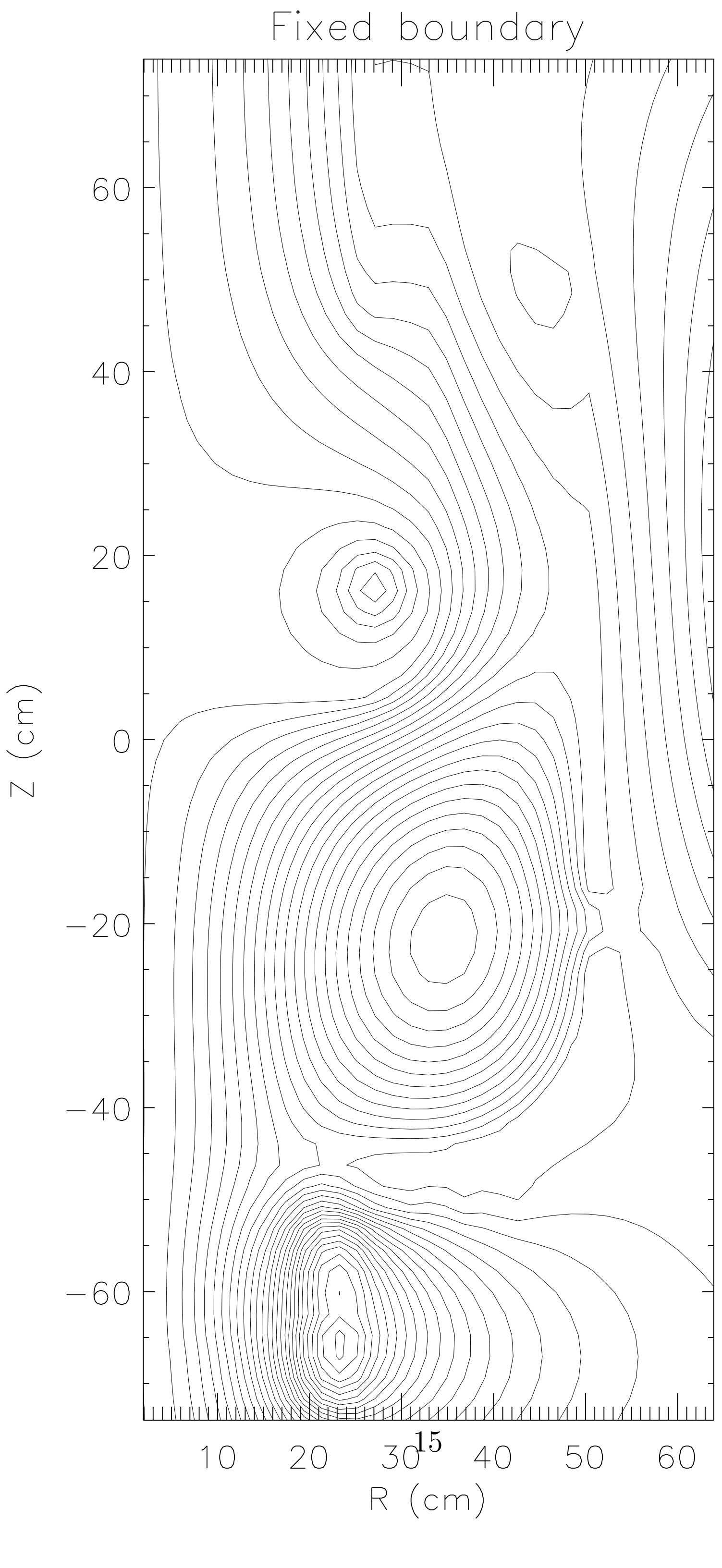

Figure 1: The direct (free-boundary) solution calculated by the demonstration driver.

Appendix A

The dead-start procedure, calling the subroutine `do_dead_start("plasma.inp", "coils.inp")`, requires the two files `"tokamak.inp"` and `"coils.inp"` to be set up by hand. The file formats are fairly self-explanatory.

Sample `tokamak.inp` file:

```
"KSTAR/DN"

Plasma...
      2.00 MA       plasma current
      1.80 m        major radius
      0.50 m        minor radius
      0.00 m        Zaxis
      1.90          95% elongation
      0.30          95% triangularity
      0.01 m        Dsep (DN)
      1.00          poloidal beta
      0.73          li
     -4.00 Wb       External flux linkage

Toroidal field...
3.50 T @ R = 1.80 m

Computational grid...
      1.00 m        Rmin
      2.60 m        Rmax
     -1.50 m        Zmin
      1.50 m        Zmax
 33 x 65            No. grid points (Nr x Nz)

Plot Scales...
      0.00 m        Rmin
      4.00 m        Rmax
     -2.50 m        Zmin
```

```
    2.50 m         Zmax
    2.00 MA/m^2    Current density for drawing coil cross-sections
```

Sample `coils.inp` file:

```
"KSTAR" PF coil set of 05/01/99 from Kim
 14 coils
name    Rc [m]   Zc [m]  DRc [m]  DZc [m]   n_turn    NI_cap  B_cap
"PF1U"  0.5610   0.2470   0.2135   0.4764        1       1       1
"PF2U"  0.5610   0.6932   0.2135   0.3808        1       1       1
"PF3U"  0.5610   0.9960   0.2135   0.1896        1       1       1
"PF4U"  0.5610   1.2510   0.2135   0.2852        1       1       1
"PF5U"  1.0850   2.2960   0.3330   0.3808        1       1       1
"PF6U"  3.0900   1.9200   0.1896   0.3808        1       1       1
"PF7U"  3.7300   0.9600   0.1418   0.2852        1       1       1
"PF1L"  0.5610  -0.2470   0.2135   0.4764        1       1       1
"PF2L"  0.5610  -0.6932   0.2135   0.3808        1       1       1
"PF3L"  0.5610  -0.9960   0.2135   0.1896        1       1       1
"PF4L"  0.5610  -1.2510   0.2135   0.2852        1       1       1
"PF5L"  1.0850  -2.2960   0.3330   0.3808        1       1       1
"PF6L"  3.0900  -1.9200   0.1896   0.3808        1       1       1
"PF7L"  3.7300  -0.9600   0.1418   0.2852        1       1       1
```