TRACK 2.0

W.A. Houlberg, P.I. Strand

July 1, 2002

1 Introduction

The **TRACK** module has been in continuous use since the early 1980's, although it has experienced occasional revisions. The present version is written in Fortran 90/95, using features that are relatively easy to adapt to different Fortran compilers and to link to other languages. In addition, the algorithms for stepping along a line segment and finding intersections and tangencies have been extensively reworked in the present version for greater clarity and computational efficiency. **TRACK** requires geometry information (conversions between Cartesian, cylindrical and magnetic flux coordinates) that must be obtained from an MHD equilibrium interface. The 2D and 3D conversions for inverse coordinate representations that were included in the original version of **TRACK** have been moved to the **AJAX** module, and are called directly from **TRACK**. In addition, the module **AJAX_XPLASMA** maps the **AJAX** calls to the 2D MHD equilibrium interface routines in **XPLASMA**. We are indebted to D. McCune for running test cases and correcting some faults in the stepping algorithm, and to A. Pankin for the generic makefiles.

Following is a description of the **TRACK** module, its availability, and its F90/95 features. The module is actually named **TRACK_MOD** and the primary routine in the module is **TRACK**. However, in this documentation the module will be referred to by its root name **TRACK**. In subsequent sections we describe each of the routines in the module. Also included are sections on revision history, limitations and known problems, future extensions, and simplified code diagrams.

In this documentation **teletype** style identifies names of Fortran modules, routines, variables and other coding.

Description of TRACK

The **TRACK** module tracks a segmented path (a connected series of straight segments in real space) through a 3D toroidal plasma and finds the intersections with a set of magnetic flux surfaces. Although originally written to map the wave damping along rays generated by an RF code in order to provide a radial profile of plasma heating, it has been used for other source applications that need plasma information along a line of sight in real space (e.g., neutral beam and pellet injection), and for inversion of chordal diagnostics to obtain profiles — see Ref [1] for the original computational algorithms, an example of inverting chordal CO_2 interferometric data to obtain the evolution of density profiles from pellet injection, as well as other applications.

There are a couple of auxiliary routines in the module that are called called by the main **TRACK** routine and are declared **PUBLIC** for possible use in other applications. One provides the length of the segment and its unit length vector. The other gives the coordinates (Cartesian, cylindrical and flux) of a point located a distance d along a segment characterized by its initial point and unit length vector.

Public Routines

- **TRACK**
- **TRACK_D**
- **TRACK_G**

Availability

This module is available with:

- Stand-alone test driver
- Test cases used in the development
- External documentation
- IDL graphic procedures for viewing results of the test cases

When all documentation and testing is complete enough for release, these should be obtained through the Module Library:

```
http://w3.pppl.gov/rib/repositories/NTCC/catalog/
```

For a beta release version, comments, suggestions, or other information, contact:


```
Address: Wayne A. Houlberg
         Fusion Energy Division
         Oak Ridge National Laboratory
         P.O. Box 2009
         Oak Ridge, TN 37831-8071

Phone:   865-574-1350
Fax:     865-576-7926
e-mail:  houlbergwa@ornl.gov
```

Fortran 90/95 Features

This module uses Fortran 90/95 features that enhance portability, flexibility and efficiency. For the most part we have following the recommendations in the document 'European Standards for Writing and Documenting Exchangeable Fortran 90 Code,' which is available on the Web at:

`http://www.met-office.gov.uk/research/nwp/numerical/fortran90/f90_standards`

These are accomodated by:

- Use of **KIND** to declare the precision of all **REAL** variables:
  - Replaces **REAL*4** and **REAL*8**, which are deprecated in Fortran 90/95
  - Set in a companion module, **SPEC_KIND_MOD**
- There are no **COMMON** blocks or any other coding features that are deprecated in Fortran 90/95
- **PRIVATE** features are used to minimize conflict with other parts of linked coding:
  - Data
  - Computational methods and numerical procedures
- Generic names are used for all intrinsic functions
- Use of optional I/O in the arguments to routines allows a user to:
  - Reduce allocation and CPU for information that is not necessary for a specific application
  - Let the routine use default assumptions
  - Extend the application to add other optional I/O information without changing existing calling routines that do not need the new information
  - **SUBROUTINE GETINFO(a,b,c,X,Y,Z)** with required I/O **a, b, c**, and **OPTIONAL** I/O **X, Y, Z** can be called in various ways from another Fortran 90/95 routine:

    ```
    CALL GETINFO(a1,b1,c1,x1,y1)
    CALL GETINFO(a1,b1,c1,Y=y1,X=x1)
    ```

    where the variable names with 1 appended are variables in the calling routine. These calls obey the rules that all required I/O appears first in the list of arguments and must be called in sequence. Optional variables can be called in sequence without using **GETINFO** names as keywords, but the **OPTIONAL** arguments may be accessed in any order if keywords are used.
- Dynamic and automatic allocation of local variables are used to reduce storage:
  - Dynamic allocation is invoked by an **ALLOCATE** statement that uses input information (variable that sets the number of elements to be included in array operations, or a check of the **SIZE** of an array when all elements are to be operated on), and always deallocated if it is not necessary to store the information for future calls
  - Automatic allocation is accomodated in the declarations at the beginning of a routine using input variables to set the dimension of arrays
- Compilation in either fixed or free format for portability

- Array syntax is used to simplify coding by removing many **DO** loops and to allow the compiler to generate more optimum coding, e.g.,
  - Initialization of a fourth rank array to zero, and all fourth rank elements with the first three indices $i = 1$, $j = 3$, $k = 5$ set to unity with the precision **PARAMETER rspec** set in **SPEC_KIND_MOD**:

    ```
    f(:,:,:,:) = 0.0_rspec
    f(1,3,5,:) = 1.0_rspec
    ```

  - A volume integral of the product of density and temperature over plasma zones 13–20 for plasma species 2 can be written as:

    ```
    i1 = 13
    i2 = 20
    energy = SUM(den(2,i1:i2)*temp(2,i1:i2)*vol(i1:i2))
    ```

    where the first rank of the density and temperature arrays indicates species and the second rank indicates radial grid

Coding Style

The coding style in this and other related Fortran 90/95 transport simulation modules developed at ORNL is followed fairly strictly, so that once a few basic rules are learned, the code is much easier to decifer, modify and debug (for other users as well as the original developers). Among the major style features are:

- All variables are declared, including the **INTENT** of all I/O arguments to each routine
- All arguments to a routine are described in a comment block at the beginning of the routine, including their units enclosed in square brackets, [units]
- There is no I/O to external devices in general physics modules; all I/O with external resources is handled by routines that are designed specifically for that purpose
- Error handling uses an error flag (**iflag** < 0 for warnings, > 0 for errors, and $= 0$ for no problems) and message (**message**) system with no termination while inside the module
- Although Fortran is not case sensitive, upper case is used to emphasize all reserved words and procedures, i.e.,
  - System calls
  - Internal Fortran functions
  - **FUNCTION** and **SUBROUTINE** names
  - Names of optional arguments

  otherwise all variables are in lower case
- There is extensive use of naming conventions to improve readability and facilitate debugging:
  - Routines in a module begin with the module root name (i.e., minus the **_MOD** descriptor)
  - Underscores in a name generally indicate some characteristic of the variable that is often designated as a subscript or superscript in mathematical notation, description of the variable describing the rank of an array, etc.
  - Options begin with **k_**
  - Logical variables begin with **l_**
- Comments, indentation, and blank lines are used to identify blocks of related coding
- There is ony one exit to each routine, with any early exits sent to a line:

  ```
  9999 continue
  ```

  followed by any relevant deallocations or other closing activities before exiting
- All **DO** and **END** statements are identified, i.e.:

  ```
  MODULE Y
  CONTAINS
  SUBROUTINE X
  ...
  DO i=1,n !Over radial nodes
    ...
  ENDDO !Over radial nodes
  END SUBROUTINE X
  END MODULE Y
  ```

- Compatibility with free and fixed formats is accomodated by using **&** in column 80 of the line to be continued (free form requirement of **&** at the end of a line, and fixed form restriction that anything beyond column 72 is ignored), and using a second **&** in column 6 of the continued line (fixed form requirement of placement, and an allowably redundant free form continuation symbol). Although this is not very elegent, it is an often used transitional step between free and fixed form codes.
- Indentation to identify code structure:
  - **DO** loop constructions
  - **IF** constructions

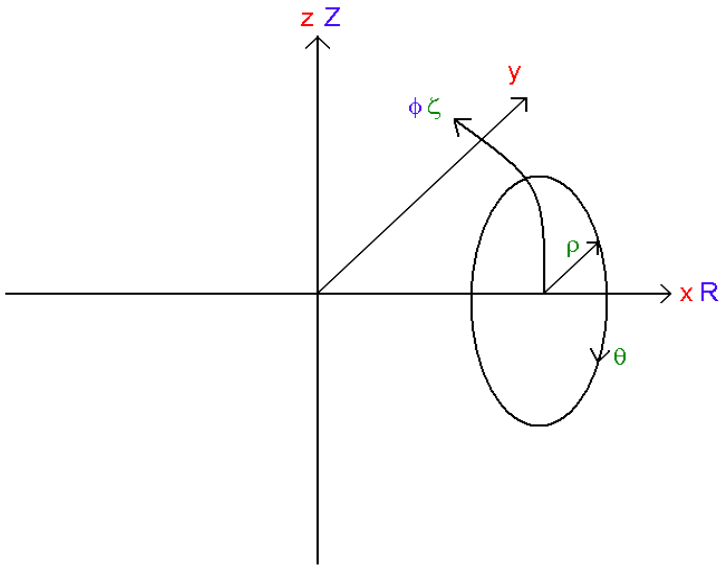

Figure 1: The Cartesian (x, y, z), cylindrical (R, ϕ, Z), and magnetic flux (ρ, θ, ζ) coordinate systems showing their relative orientations.

2 Public Routines

The coordinate systems used in the **TRACK** module consist of three right-handed systems: Cartesian (x, y, z), cylindrical (R, ϕ, Z), and magnetic flux (ρ, θ, ζ). These are illustrated in Fig. 1 along with their relative orientations. Note that the cylindrical and magnetic flux toroidal angles are assumed to be identical, $\zeta = \phi$, with the positive direction being counterclockwise when viewed from above the torus. The positive direction of the poloidal angle in magnetic flux coordinates is down on the outside of the torus with $\theta = 0$ at the outside midplane.

TRACK(n_rho, rho, n_seg, r_seg, n_int, irho_int, s_int, iflag, message,
K_SEG, IZONE_INT, RFLX_INT, RCYL_INT, RCAR_INT, SDOTB_INT,
SDOTPHI_INT)

General Description

TRACK steps along a segmented path specified by $n_{\rm seg}$ nodes in (default) cylindrical coordinates, $\vec{r}_{\rm seg}$, to find the intersections, $n_{\rm int}$, with a user-supplied set of n_ρ flux surfaces, ρ_i, $i = 1, n_\rho$. The first flux surface may or may not define the magnetic axis ($\rho_1 = 0$), but the surfaces are assumed to be simply nested with the ρ_i monotonically increasing. It is also possible to specify a single surface. In addition to the indices of the surfaces at the intersections, $i_{\rho,\rm int}$, the distances from the beginning of the path to the intersections are returned, $s_{\rm int}$. As an option, the user can use Cartesian or flux coordinates to define the segments. There are also optional output variables that characterize the set of intersections. The set of intersections includes the segment endpoints, which are identified by $i_{\rho,\rm int} = 0$.

The basic algorithm consists of stepping along each segment of the path in sequence. It uses the radial flux coordinates of the step end points to determine whether a flux surface of interest has been passed. It also

uses the derivatives of the flux coordinates at the endpoints to determine whether the segment is headed inward or outward with respect to the magnetic axis and to determine whether tangencies have been passed. Because it is possible that portions of the path may be far outside the plasma where the flux coordinate determination is not valid, the stepping algorithm allows for invalid transformations and assumes those points are outside the plasma. One possible failure is when an invalid coordinate transformation occurs while the segment is known to be inside the domain of the set of flux surfaces of interest, which means the plasma geometry and/or conversion routines represented by **AJAX** or **XPLASMA** is at fault.

Mathematical Description

There are three levels of position notation in the algorithm that need to be clarified. Let s represent the distance from the beginning of the segmented path, let d represent the distance from the beginning of a straight segment in the path, let 0 refer to the beginning of a step within the segment, and let 1 designate the end of the step. All the output refers to the distances represented by s; the distances within a segment and information about an individual step are only used as private information for obtaining s. Information for each of these three levels needs to be initialized.

Path Initiation

The general initialization consists of three steps: setting characteristic lengths, nulling the output arrays, and setting the cylindrical coordinates defining the segments. The characteristic major and minor radii of the toroidal plasma, R_0 and $\rho_{\max}$ respectively, are obtained from the MHD equilibrium interface (**AJAX** or **XPLASMA**) and used to set upper and lower limits on step sizes and characteristic gradients:

$$\epsilon_R = 10^{-4} \tag{1}$$

$$\Delta s_{\min} = \epsilon_R R_0 \tag{2}$$

$$\Delta s_{\max} = 10^2 \Delta s_{\min} \tag{3}$$

$$\Delta \rho_{\min} = \epsilon_R \rho_{\max} \tag{4}$$

$$\left.\frac{\partial \rho}{\partial s}\right|_{\min} = \frac{\Delta \rho_{\min}}{\Delta s_{\max}} \tag{5}$$

After setting all the output arrays to null values, the cylindrical coordinates defining the segments are set: using the input values if the default cylindrical coordinate specification is used, or converted from input magnetic flux or Cartesian coordinates by appropriate calls to **AJAX** or **XPLASMA** when the optional input switch is set to $k_{\text{seg}} = 1$ or 2 for magnetic or Cartesian specification, respectively.

Segment Initialization

The next stage of initialization consists of setting information at the beginning of the first segment for the loop over segments. The position of the first segment relative to the beginning of the path is given by $s_{\text{seg}} = 0$. The Cartesian and flux coordinates of this point are determined using appropriate calls to **AJAX** or **XPLASMA**. The largest flux surface that satisfies the condition $\rho_0 \geq \rho_i$ is located and designated $i = \text{low}$, allowing for the possibility that the point lies on a flux surface if it is within the tolerance, $|\rho(i_{\text{low}}) - \rho_0| < \Delta\rho_{\min}$. The fixed and optional output variables at this starting point are then recorded: $n_{\text{int}} = 1$, $s_{\text{int}}(1) = 0$, the plasma zone being entered $i_{z,\text{int}}$, the magnetic flux, Cartesian and cylindrical

coordinates, the sine of the angle of intersection with the magnetic field $d\hat{l} \cdot \vec{B}/|B|$, and the sine of the angle with respect to the toroidal direction $d\hat{l} \cdot \hat{\phi}$. The latter two are useful for determining the pitch angle of particles for Fokker-Planck calculations or parallel momentum input and toroidal momentum input, respectively.

Step Initiation

The loop over segments contains a loop over steps within a segment, and includes the possibility that a step need to be reduced because it is too long (using the step halving counter, m_h, discussed later). Prior to stepping along the segment, the general segment information and information at the beginning of the first step needs to be set. The segment length, $d_{\rm seg}$, and unit length vector along the segment, $d\hat{l}$, are set by calling **TRACK_G**. A call to the auxiliary routine **TRACK_D** sets the Cartesian, magnetic flux and cylindrical coordinates at the beginning of first step, along with $d\rho/ds|_0$, the 2-D Jacobian, τ_0, and derivatives of R and Z with respect to the flux coordinates. If the Jacobian fails, we set $d\rho/ds|_0 = 0$, and if $|d\rho/ds|_0| > d\rho/ds|_{\rm min}$ we set it to the minimum value with the appropriate sign. This allows us to use the null condition to indicate a bad Jacobian. The step reduction coefficient and position of the end of the step are initialized: $m_h = 0$ and $d_1 = 0$, respectively.

Step Loop

The step loop is terminated by error exit when there are too many halvings of the step size (set to $m_h > 5$), or terminated normally when the end of the previous step (d_1) is within the minimum step size of the end of the segment:

$$d_1 > d_{\rm seg} - \Delta s_{\rm min} \tag{6}$$

Otherwise the new step size is set by the following algorithm that uses **ilo** to designate the index of the next lower flux surface below point 0:

$$\begin{aligned}
\Delta s &= \Delta s_{\rm max}, \quad \text{for} \quad \left|\frac{\partial \rho}{\partial s}\right|_0 < \left.\frac{\partial \rho}{\partial s}\right|_{\rm min} && (7)\\
&= \Delta s_{\rm max}, \quad \text{for} \quad \left.\frac{\partial \rho}{\partial s}\right|_0 < 0, \quad \text{ilo} = 0 && (8)\\
&= 1.2\left|\frac{\rho_1 - \rho_{\rm ilo}}{\partial \rho/\partial s|_0}\right|, \quad \text{for} \quad \left.\frac{\partial \rho}{\partial s}\right|_0 < 0, \quad \text{ilo} \neq 0 && (9)\\
&= 1.2\left|\frac{\rho_{\rm ilo+1} - \rho_1}{\partial \rho/\partial s|_0}\right|, \quad \text{for} \quad \left.\frac{\partial \rho}{\partial s}\right|_0 > 0, \quad \text{ilo} < n_\rho && (10)\\
&= \Delta s_{\rm max}, \quad \text{for} \quad \left.\frac{\partial \rho}{\partial s}\right|_0 > 0, \quad \text{ilo} \geq n_\rho && (11)
\end{aligned}$$

where the factors of 1.2 have been found to be reasonably optimal for stepping just past the next surface, noting that the radius of curvature may be increasing and to ensure that the algorithm doesn't leave the point always short of the target surface. The step size is adjusted to ensure that it doesn't go past the end of the segment, and is halved in case the step size has been determined to be too large on the previous step:

$$\Delta s = \frac{\min(\Delta s, d_{\rm seg} - d_0)}{2^{m_h}} \tag{12}$$

A second check is then made to ensure that the step size is at least the minimum, $\Delta s \geq \Delta s_{\rm min}$, and the coordinates and other relevant information at point 1 are determined by a call to **TRACK_D**.

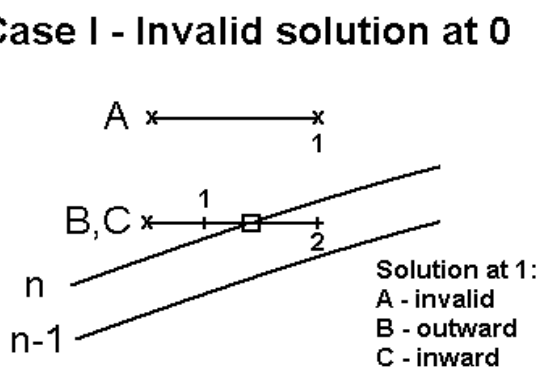

Figure 2: In Case I the Jacobian of the initial point (the left end of each horizontal line) is invalid and assumed to be outside the plasma. The stepping proceeds until a valid point just outside the plasma is obtained. An x designates an invalid Jacobian, a + designates a valid Jacobian, and a square designates the intersection with a surface.

There are three conditions on $\partial\rho/\partial s$ at each endpoint that yield nine basic cases. Each of these has several subcases determined by whether a surface of interest has been hit, whether a tangency has been passed, whether the step needs to be reduced, and whether to advance the step. The algorithm is set up this way to allow for either concave or convex local curvature of the surface, which can occur in either axisymmetric or non-axisymmetric plasmas. Because $\partial\rho/\partial s$ is used in the algorithm to determine the step size and the set of cases, we must be careful in the vicinity of $\partial\rho/\partial s = 0$. For an invalid Jacobian we set $\partial\rho/\partial s = 0$ and reset $\partial\rho/\partial s$ to the minimum value with the appropriate sign if it is smaller than the minimum but otherwise valid.

These nine cases and their subcases are illustrated in Figs 2–4. Cases I, II, and III refer to the conditions $\partial\rho/\partial s$ invalid, > 0, and < 0, respectively, at the start of the step, point 0. Cases A, B, abd C refer to the conditions $\partial\rho/\partial s$ invalid, > 0, and < 0, respectively, at the end of the step, point 1. For each line representing the step, the initial position (point 0) is on the left, and the endpoint is labelled by a number representing the subcase. We now go through the decision process on a case-by-case basis, setting appropriate flags and parameters that dictate actions at the end of the set of cases. Let $i_{\rm hit}$ designate the index of a flux surface that has been intersected, in which case d_1 is reset to the position of the intersection. The condition $i_{\rm hit} = 0$ will be taken to mean that no surface has been hit. Let $m_h = 0$ be the condition that the step has been successfully completed and doesn't have to be retaken. Otherwise, it needs to be incremented to halve the step size.

Cases I.A

The Jacobian at both 0 and 1 is invalid and the only action is to proceed with the next step:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= 0 \end{aligned} \tag{13}$$

Cases I.B

The Jacobian at 0 is invalid and the segment is headed outward at 1. There are two conditions at 1 that lead to subcases 1 and 2: even though the segment is headed outward at 1 and 0 is assumed to be outside the plasma, the step may have been large enough to cross a flux surface and pass a tangency.

In Case I.B.1 point 1 is outside the target surface and the step can be completed:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{14}$$

By default (in not meeting other subcase conditions) then in Case I.B.2 point 1 is inside the plasma and the step should be shortened to get a valid point just outside the plasma so the location of the intersection can be found by interpolation:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= m_h + 1 \end{aligned} \tag{15}$$

Cases I.C

The Jacobian at 0 is invalid and the segment is headed inward at 1. The same subcases considered in I.B are applicable.

In Case I.C.1 point 1 is outside the target surface and the step can be completed:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{16}$$

By default then in Case I.C.2 point 1 is inside the plasma and the step should be shortened to get a valid point just outside the plasma so the location of the intersection can be found by interpolation:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= m_h + 1 \end{aligned} \tag{17}$$

Cases II.A

The segment is headed outward at 0 and the Jacobian is invalid at 1. There are two subcases that depend on whether point 1 is inside or outside the outermost flux surface supplied by the user.

In Case II.A.1 point 0 is outside the outermost surface and, because it is headed outward, it is assumed the step simply entered a region further out where the Jacobian is bad. So the step can be completed:

$$i_{\text{hit}} = 0$$

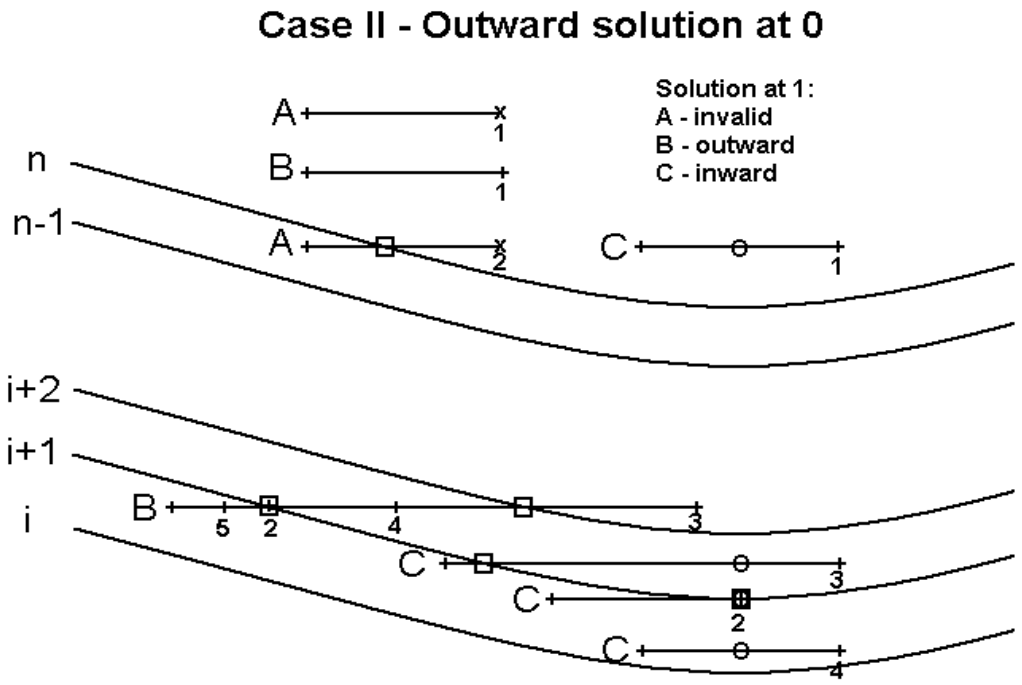

Figure 3: In Case II the segment is directed outward at the initial point (the left end of each horizontal line) and information at the other (right) end of the segment is used to determine whether the segment continues outward (B) or passes a tangency and turns inward (B), and whether it intersects a surface of interest in between. In Case II.A the Jacobian at the right end is invalid and the step must be reduced to obtain a valid point for interpolation. The notation is the same as in Case I, except with the addition of an *o* to designate a tangency.

$$m_h = 0 \tag{18}$$

In Case II.A.2 point 0 is inside the outermost surface and the step should be shortened to get a valid point at 1 so the location of the intersection can be found by interpolation:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= m_h + 1 \end{aligned} \tag{19}$$

Cases II.B

The segment is headed outward at both 0 and 1, leading to 5 subcases that must be considered in sequence.

In Case II.B.1 point 0 is outside the outermost surface (and presumably 1 is also because it is also headed outward) so there can be no intersection and the step can be completed:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= 0 \end{aligned} \tag{20}$$

In the other subcases the point 0 is then inside the outermost surface and point 1 can meet one of four conditions relative to the target surface.

In Case II.B.2 point 1 has hit the target surface, $\rho_{\rm ilo+1}$, within the tolerance $\Delta\rho_{\rm min}$. A hit can be recorded, a new target surface identified, and the step completed. The position of the hit along the segment is extrapolated from the conditions at 1 (which should be closer to the target surface than point 0 in virtually every case), the value of d_1 is overwritten with the hit position, and the step size reset:

$$\begin{aligned} i_{\rm hit} &= {\rm ilo} + 1 \\ m_h &= 0 \\ {\rm ilo} &= {\rm ilo} + 1 \\ d_1 &= d_1 + \frac{\rho_{\rm hit} - \rho_1}{\partial\rho/\partial s|_1} \\ ds &= d_1 - d_0 \end{aligned} \tag{21}$$

In Case II.B.3 the point 1 has crossed more than 1 target surface and the step needs to be halved until only one target surface is crossed:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= m_h + 1 \end{aligned} \tag{22}$$

In Case II.B.4 the point 1 has crossed a single surface. The position of the hit is recorded using a finite difference approximation to $\partial\rho/\partial s$ at the two end points by overwriting the distance to the intersection, the target surface is advanced to the next larger surface, and the step size is reset:

$$\begin{aligned} i_{\rm hit} &= {\rm ilo} + 1 \\ m_h &= 0 \end{aligned}$$

$$\begin{aligned} \text{ilo} &= \text{ilo} + 1 \\ d_1 &= d_1 + \frac{\rho_{\text{hit}} - \rho_0}{\rho_1 - \rho_0} \\ ds &= d_1 - d_0 \end{aligned} \tag{23}$$

By default then in Case II.B.5 no surfaces have been crossed and the step is advanced:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{24}$$

Cases II.C

The segment is headed outward at 0 and inward at 1, meaning that it has passed a tangency. The subcases have to consider whether an intersection has also been passed, so the position of the tangency is first estimated from the end point derivatives, and the step size is reset:

$$\begin{aligned} d_1 &= \frac{d_1 \frac{\partial\rho}{\partial s}\big|_0 - d_0 \frac{\partial\rho}{\partial s}\big|_1}{\frac{\partial\rho}{\partial s}\big|_0 - \frac{\partial\rho}{\partial s}\big|_1} \\ ds &= d_1 - d_0 \end{aligned} \tag{25}$$

The coordinates at the tangency are determined by a call to **TRACK_D** at the new position 1, and $\partial\rho/\partial s$ is overwritten to be the minimum value past the tangency:

$$\left.\frac{\partial\rho}{\partial s}\right|_1 = -\left.\frac{\partial\rho}{\partial s}\right|_{\min} \tag{26}$$

In Case II.C.1 the lower bound surface is the outermost surface and no surface could have been hit so the step is advanced:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{27}$$

In Case II.C.2 the tangency lies within the tolerance of a hit on the target surface, $\rho_{\text{ilo}+1}$, within the tolerance $\Delta\rho_{\min}$. Note that the lower bounding surface remains **ilo**+1 and the direction of progress has already been forced to be beyond the intersection. A hit can be recorded and the step completed:

$$\begin{aligned} i_{\text{hit}} &= \text{ilo} + 1 \\ m_h &= 0 \end{aligned} \tag{28}$$

In Case II.C.3 $\rho_{\text{ilo}+1}$ has been passed before the tangency. The hit is recorded as a linear interpolation between 0 and the tangency, the next outer surface is targeted, the forced gradient at point 1 is unforced by setting it to twice the minimum, and the step completed:

$$\begin{aligned} i_{\text{hit}} &= \text{ilo} + 1 \\ m_h &= 0 \end{aligned}$$

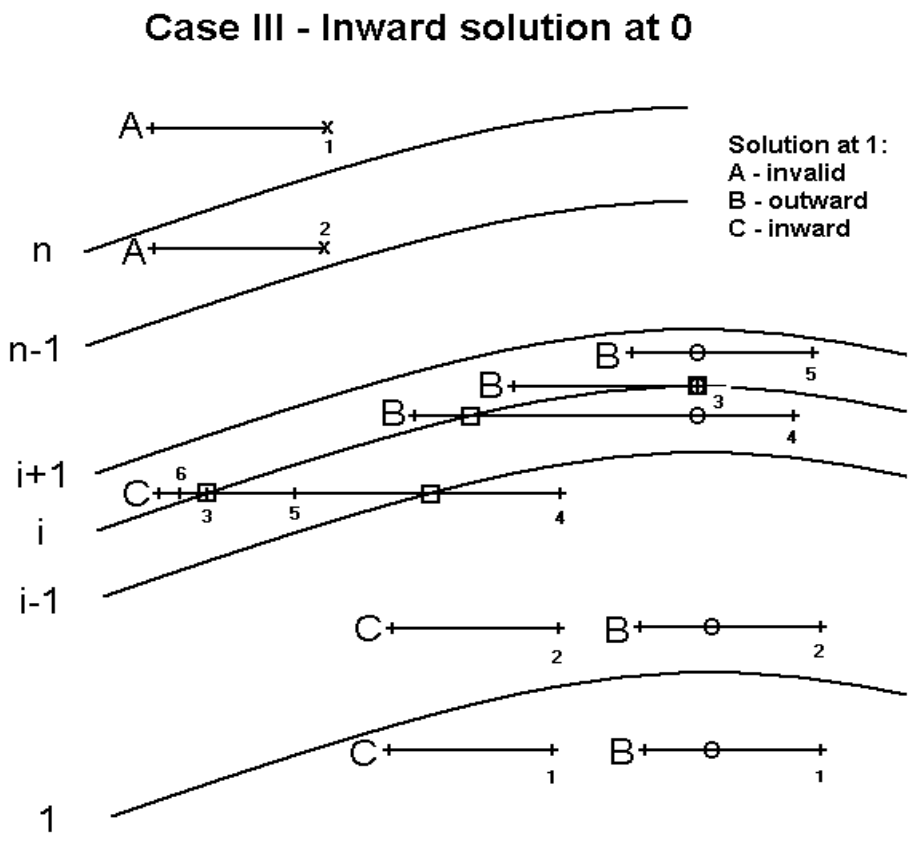

Figure 4: In Case III the segment is directed inward at the initial point (the left end of each horizontal line) and information at the other (right) end of the segment is used to determine whether the segment continues inward (C) or passes a tangency and turns outward (C), and whether it intersects a surface of interest in between. In Case III.A the Jacobian at the right end is invalid and the step must be reduced to obtain a valid point for interpolation if the initial point is outside the plasma, or fails if the initial point was inside the plasma. The notation is the same as in the previous cases.

$$\begin{aligned} \mathrm{ilo} &= \mathrm{ilo}+1 \\ d_1 &= d_0 + ds\frac{\rho_{\mathrm{hit}}-\rho_0}{\rho_1-\rho_0} \\ ds &= d_1 - d_0 \\ \left.\frac{\partial\rho}{\partial s}\right|_1 &= 2\left.\frac{\partial\rho}{\partial s}\right|_{\min} \end{aligned} \tag{29}$$

By default then in Case II.C.4 no surface was crossed and the step can be completed:

$$\begin{aligned} i_{\mathrm{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{30}$$

Cases III.A

There is a valid inward solution at 0 and an invalid solution at 1. There are only two subcases to consider.

In Case III.A.1 point 0 is outside the plasma and we can proceed with another step:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= 0 \end{aligned} \tag{31}$$

By default then in Case III.A.2 point 0 is inside the plasma, and since the segment was headed further inward at that point, a valid ransformation should have been obtained at 1. Since the solution is invalid at 1 we set the error flag to indicate a fatal error and exit **TRACK**.

Cases III.B

There is a valid inward solution at 0 and a valid outward solution at 1, meaning that a tangency has been passed. The tangency is again estimated from the end point derivatives and the step size is reset:

$$\begin{aligned} d_1 &= \frac{d_1 \frac{\partial\rho}{\partial s}\big|_0 - d_0 \frac{\partial\rho}{\partial s}\big|_1}{\frac{\partial\rho}{\partial s}\big|_0 - \frac{\partial\rho}{\partial s}\big|_1} \\ ds &= d_1 - d_0 \end{aligned} \tag{32}$$

The coordinates at the tangency are determined by a call to **TRACK_D** at the new position 1, and $\partial\rho/\partial s$ is overwritten to be the minimum value past the tangency:

$$\left.\frac{\partial\rho}{\partial s}\right|_1 = \left.\frac{\partial\rho}{\partial s}\right|_{\rm min} \tag{33}$$

The primary difference from the corresponding outward Case II.C is to consider whether the tangency is inside the innermost surface (axis not specified as one of the surfaces) or if the axis is specified, to not bother finding the intersection there. This leads to five subcases.

In Case III.B.1 tangency is inside the innermost surface and the step can be completed:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= 0 \end{aligned} \tag{34}$$

In Case III.B.2 the inner surface is within $\Delta\rho_{\rm min}$ of the axis and its possible intersection is ignored and a new step taken:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= 0 \end{aligned} \tag{35}$$

In Case III.B.3 the tangency is within $\Delta\rho_{\min}$ of the surface and its intersection is recorded and the step completed:

$$\begin{aligned} i_{\text{hit}} &= i_{\text{ilo}} \\ m_h &= 0 \end{aligned} \tag{36}$$

The index of the lower surface is not advanced because that surface still bounds the zone for the beginning of the next step.

In Case III.B.4 surface ρ_{ilo} has been crossed before the tangency. The hit is recorded as a linear interpolation between 0 and the tangency, the next lower surface is targeted, the forced gradient at 1 is unforced by setting it to twice the minimum, and the step is completed:

$$\begin{aligned} i_{\text{hit}} &= \text{ilo} \\ m_h &= 0 \\ \text{ilo} &= \text{ilo} - 1 \\ d_1 &= d_0 + ds\frac{\rho_{\text{hit}} - \rho_0}{\rho_1 - \rho_0} \\ ds &= d_1 - d_0 \\ \left.\frac{\partial\rho}{\partial s}\right|_1 &= 2\left.\frac{\partial\rho}{\partial s}\right|_{\min} \end{aligned} \tag{37}$$

In the default Case III.B.5 none of the above conditions are satisfied, and no surface has been crossed. The step can be competed:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{38}$$

Cases III.C

The segment is headed inward at both 0 and 1. The subcases are similar to Case II.B where both are outward, except for considerations at the lowest surface that may be the axis or away from the axis as in Case III.B. There are six subcases to consider.

In Case III.C.1 the lowest surface is away from the axis and point 1 has not passed a tangency to rehit the lowest surface so the step can be completed:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{39}$$

In Case III.C.2 the lower surface is within $\Delta\rho_{\min}$ of the axis and its possible intersection is ignored and the step completed:

$$\begin{aligned} i_{\text{hit}} &= 0 \\ m_h &= 0 \end{aligned} \tag{40}$$

In Case III.C.3 point 1 is within $\Delta\rho_{\min}$ of the lower surface, the hit is recorded, the index of the next lower surface is reset, and the step completed:

$$\begin{aligned} i_{\rm hit} &= {\rm ilo} \\ m_h &= 0 \\ {\rm ilo} &= {\rm ilo} - 1 \end{aligned} \tag{41}$$

In Case III.C.4 more than one surface has been crossed and the step needs to be reduced:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= m_h + 1 \end{aligned} \tag{42}$$

In Case III.C.5 surface $\rho_{\rm ilo}$ has been crossed, the intersection is recorded, the next lower surface is identified, point 1 is reset to the intersection, the step to the intersection is reset, and the step completed:

$$\begin{aligned} i_{\rm hit} &= {\rm ilo} \\ m_h &= 0 \\ {\rm ilo} &= {\rm ilo} - 1 \\ d_1 &= d_0 + ds\frac{\rho_{\rm hit} - \rho_0}{\rho_1 - \rho_0} \\ ds &= d_1 - d_0 \end{aligned} \tag{43}$$

In default Case III.C.6 none of the above subconditions have been met so no surface has been crossed and the step is completed:

$$\begin{aligned} i_{\rm hit} &= 0 \\ m_h &= 0 \end{aligned} \tag{44}$$

Step Completion

The step completion consists of two actions, recording information at an intersection when $i_{\rm hit} \neq 0$, and advancing the step when $m_h = 0$. In recording the intersection, which is at point 1, we want to note the possibility of a forced gradient at 1 to get past a tangency in the degenerate situation where the intersection is at a tangency within the specified tolerances. The gradient at 1 is temporarily stored while the coordinates and other information are obtained at 1 by a call to **TRACK_D**, then used to overwrite the value returned by **TRACK_D**, which may have found the point short of the tangency within numerical accuracy. The output is recorded and $i_{\rm hit}$ is reset to 0. If the step is not to be halved ($m_h = 0$) the coordinates and other information are copied from point 1 to point 0.

(

Segment Completion

At the end of the segment the starting position of the next segment is advanced, $s_{\rm seg} = s_{\rm seg} + d_{\rm seg}$, and the output information is recorded with $i_{\rho,{\rm int}} = 0$.

I/O

```
!Input:
!  n_rho               -number of radial nodes [-]
!  rho(n_rho)          -radial nodes [-]
!  n_seg               -number of points defining the segments = segments+1 [-]
!  r_seg(3,n_seg)      -coordinates defining the segments (see K_SEG)
!Output:
!  n_int               -number of nodes + intersections [-]
!  irho_int(n_int)     -radial nodes of intersections [-]
!                      =0 if point is a node of the ray
!  s_int(n_int)        -length along path to intersections [m]
!  iflag               -error and warning flag [-]
!                      =-1 warning
!                      =0 none
!                      =1 error
!  message             -warning or error message [character]
!Optional input:
!  K_SEG               -option for coordinates specifying the segments [-]
!                      =1 flux coordinates (rho,theta,zeta)
!                      =2 Cartesian coordinates (x,y,z)
!                      =else default cylindrical coordinates (R,phi,Z)
!Optional output:
!  IZONE_INT(n_int)    -zone being entered [-]
!  RFLX_INT(3,n_int)   -flux coordinates [rho,rad,rad]
!  RCYL_INT(3,n_int)   -cylindrical coordinates [m,rad,m]
!  RCAR_INT(3,n_int)   -Cartesian coordinates [m,m,m]
!  SDOTB_INT(n_int)    -cos of angle between path and B [-]
!  SDOTPHI_INT(n_int)  -cos of angle between path and phi [-]
```

TRACK_D(d, r_cars, g_cars, r_flx, r_car, r_cyl, drhods, g_cyl, tau, iflag, message)

General Description

TRACK_D determines the flux ($\vec{\rho}$), Cartesian ($\vec{x}$), and cylindrical ($\vec{r}$) coordinates at a distance d along a straight segment characterized by an initial point, $\vec{x}_0$, and the unit length vector along the segment, $\hat{dl}$. It also returns the derivative of the radial flux coordinate along the segment, $d\rho/ds$, an array of the potentially non-trivial derivatives of the cylindrical coordinates with respect to the flux coordinates, $(dR/d\vec{\rho}, dZ/d\vec{\rho})$, and the 2D Jacobian of the conversion from flux to cyindrical coordinates, τ, at the end point.

Mathematical Description

The starting position of the segment in Cartesian coordinates, the gradients along the segment, and distance determine the position of the end point in Cartesian coordinates:

$$\vec{x}_1 = \vec{x}_0 + d\,\hat{dl} \tag{45}$$

The cylindrical and flux coordinates at the end point are obtained from calls to AJAX or XPLASMA cylindrical to flux coordinate conversion routines. These also return the derivatives of the cylindrical coordinates with respect to the flux coordinates and the 2D Jacobian, $\tau = R_\theta Z_\rho - R_\rho Z_\theta$. The rate of variation of ρ along the chord, $d\rho/ds$, is useful for determining whether the segment is proceeding inward or outward with respect to the flux surfaces, and for setting an estimate of the step size to intersect the next surface. Successive application of the chain rule gives:

$$\frac{d\rho}{ds} = \left(\rho_R x - \frac{\rho_\phi}{R} y\right)\frac{1}{R}\hat{dl}_x + \left(\rho_R y + \frac{\rho_\phi}{R} x\right)\frac{1}{R}\hat{dl}_y + \rho_Z \hat{dl}_z \tag{46}$$

$$\rho_R = -\frac{Z_\theta}{\tau} \tag{47}$$

$$\rho_Z = \frac{R_\theta}{\tau} \tag{48}$$

$$\rho_\phi = \frac{R_\zeta Z_\theta - R_\theta Z_\zeta}{\tau} \tag{49}$$

where subscripts on the variables designate derivatives with respect to that variable. It is worthwhile to note here the distinction between the two toroidal coordinates ϕ and ζ, even though we assume $\phi = \zeta$. In applying the chain rule in any given coordinate system the derivative with respect to one coordinate is taken while the other two coordinates in that system are held constant. Therefore, in a non-axisymmetric plasma R_ζ is generally non-zero because the derivative is taken tangent to the surface, while R_ϕ vanishes everywhere. In an axisymmetric plasma, of course, $R_\zeta = R_\phi = Z_\zeta = Z_\phi = 0$.

I/O

```
!Input:
!  d                  -distance along segment [m]
!  r_cars(3)          -Cartesian coordinates of start of segment [m]
!  g_cars(3)          -unit length vector along segment [-]
```

```
!Input/output:
!  r_flx(3)            -flux coordinates (rho,theta,zeta) [rho,rad,rad]
!Output:
!  r_car(3)            -Cartesian coordinates (x,y,z) [m,m,m]
!  r_cyl(3)            -cylindrical coordinates (R,phi,Z) [m,rad,m]
!  drhods              -drho/ds at position d along the chord [rho/m]
!  g_cyl(6)            -R,Z derivatives
!                      =(R_rho,R_theta,R_zeta,Z_rho,Z_theta,Z_zeta)
!                       [m/rho,m,      m,     m/rho,m,      m     ]
!  tau                 -2-D Jacobian in phi=zeta=constant plane [m**2/rho]
!  iflag               -error and warning flag [-]
!                      =-1 warning
!                      =0 none
!                      =1 error
!  message             -warning or error message [character]
```

TRACK_G(r_cyl0, r_cyl1, d, g_car)

General Description

TRACK_G determines the length of a segment, d, specified by two points in cylindrical coordinates, $\vec{r}_0$ and $\vec{r}_1$. It also determines the unit vector along the segment in Cartesian coordinates, $d\hat{l}$.

Mathematical Description

The input cylindical coordinates, $\vec{r}_0 = (R_0, \phi_0, Z_0)$ and $\vec{r}_1 = (R_1, \phi_1, Z_1)$, are first converted to Cartesian coordinates, (x_0, y_0, z_0) and (x_1, y_1, z_1), by calling the appropriate AJAX or XPLASMA conversion routines. From the Cartesian oordinates the segment length, d, and unit length vector, $d\hat{l}$, are trivially determined:

$$d = \left[(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2\right]^{1/2} \tag{50}$$

$$d\hat{l} = \frac{d\vec{x}}{ds} \tag{51}$$

$$= \left(\frac{x_1 - x_0}{d}, \frac{y_1 - y_0}{d}, \frac{z_1 - z_0}{d}\right) \tag{52}$$

I/O

```
!Input:
!  r_cyl0(3)          -cylindrical coordinates at start of segment [m,rad,m]
!  r_cyl1(3)          -cylindrical coordinates at end of segment [m,rad,m]
!Output:
!  d                  -length of segment [m]
!  g_car(3)           -unit length vector along segment [-]
```

3 Files and Test Cases

From the tarred/zipped files you should get the directory structure:

```
\track          -main directory plus configure and make files
  \bin          -binaries
  \dat          -data
  \doc          -documentation
  \lib          -libraries
  \results      -results
  \src          -source code
```

There are two versions of the test code – one that links to AJAX and one that links to XPLASMA – and each of these has test cases. Each test case generates 3 output files:

1d_s_track.dat	1D output along path for IDL postprocessing
sum_track.dat	1D output along path for viewing with editor
msg_track.dat	Error, warning and other messages

Sample output files for each test case have a case name appended and either **4** or **8** designating the precision set in **SPEC_KIND.MOD**, which can be editted to change the precision. **TRACK/AJAX** can be run in either 'single' or 'double' precision through the use of **KIND** declarations, and **TRACK/XPLASMA** can only be run in 'double' precision throught the use of **REAL*8** declarations. The precision parameters are therefore set to 'double' in the distribution to cover both.

TRACK/AJAX and TRACK/XPLASMA on a UNIX system

On a UNIX system you can configure the makefile with the command:

./configure --with-library-path=[library source]

where [library source] is needed for the **TRACK/XPLASMA** option and includes the **xplasma** and **pspline** libraries available through the NTCC library (at PPPL these are located at **/usr/ntcc/lib**). Also, the **lapack** and **blas** libraries are needed.

For SUN Solaris systems the f77compat library needs to be specified, if the library is not in the standard lib directory (/usr/lib, /usr/local/lib, /lib, and etc). For example, for the PPPL Orion workstation it takes the form:

**./configure --with-library-path=/usr/ntcc/lib/ **
--with-f77compat-library=/afs/pppl.gov/sun4x_56/opt/SUNWspro/SC5.0/lib

The f77compat library is not required for non-SUN systems and configure will look like:

./configure --with-library-path=/usr/ntcc/lib/

Then **make** can be run with the following options:

make ajax	to make **TRACK/AJAX**
make xplasma	to make **TRACK/XPLASMA**
make atesttok	to run **TRACK/AJAX** on generic tokamak case
make atestnstx	to run **TRACK/AJAX** on NSTX tokamak case
make atestncsx	to run **TRACK/AJAX** on NCSX stellarator case
make xtestnstx	to run **TRACK/XPLASMA** on NSTX tokamak case
make clean	to remove ***.o, *.mod** files
make realclean	to remove ***.o, *.mod** plus library and binary files

The output from each test case is compared with the reference output and the results are stored in the results directory

TRACK and AJAX Sources and Test Cases

Source routines:

TRACK_AJAX_DR.F90	Driver for **TRACK** coupled to **AJAX**
SPEC_KIND.F90	Precision specification module
TRACK_MOD.F90	**TRACK** module
WRITE_MOD.F90	Formatted output module
SETUP_AJAX.F90	**AJAX** setup routine
AJAX_MOD.F90	**AJAX** module
LINEAR1_MOD.F90	1D linear interpolation module
SPLINE1_MOD.F90	1D spline interpolation module

Test Cases:

Simple tokamak test case.

Copy **nml_track_tok.dat** to **nml_track.dat** and execute.

NSTX (low A tokamak) test case.

Copy **nml_track_nstx.dat** to **nml_track.dat** and execute. This case gets VMEC inverse coordinate expansion coefficients from **in_nstx_8.dat**.

NCSX (stellarator) test case.

Copy **nml_track_ncsx.dat** to **nml_track.dat** and execute. This case gets VMEC inverse coordinate expansion coefficients from **in_ncsx_8.dat**.

TRACK and XPLASMA Sources and Test Case

Source routines:

TRACK_XPLASMA_DR.F90	Driver for **TRACK** coupled to **AJAX**
SPEC_KIND.F90	Precision specification module
TRACK_MOD.F90	**TRACK** module
WRITE_MOD.F90	Formatted output module
SETUP_XPLASMA.F90	**AJAX** setup routine
AJAX_XPLASMA_MOD.F90	Conversion of **AJAX** calls to **XPLASMA** calls

In addition, you will need the libraries discussed above.

Test case:

NSTX (low A tokamak) test case.

Copy **nml_trackx_nstx.dat** to **nml_track.dat** and execute. This case gets inverse coordinate expansion coefficients from **11114p06.treq_data**.

Documentation

This documentation was generated in LaTeX. Contact me (houlbergwa@ornl.gov) for the LaTeX source files.

4 Revision History

TRACK 2.0 is a major rewrite of an earlier version that includes the following:

- Conversion to an F90/95 module
- Cleaner implementation of the stepping algorithm
- Replacing the root finding algorithms for intersections and tangencies with much faster, but slightly less accurate interpolations. The step sizes have been reduced to compensate and test cases typically yield mm accuracy – well within experimental resolution.

5 Limitations and Known Problems

In the coupling to XPLASMA there is a potential compatibility issue between the **KIND** specification in the F90/95 modules and the **REAL*4** and **REAL*8** specifications in the F77 routines.

6 Future Extensions

None planned.

7 Code Variables and Mathematical Symbols

d	d	distance from start of segment
d0	d_0	distance from start of segment to start of step
d1	d_1	distance from start of segment to end of step
drhods	$d\rho/ds$	gradient
drds_min	$d\rho/ds\|_{\min}$	minimum gradient
drds0	$d\rho/ds\|_0$	gradient at beginning of step
drds1	$d\rho/ds\|_1$	gradient at end of step
drho_min	$\Delta\rho_{\min}$	flux surface resolution
ds	Δs	step size
dseg	$d_{\rm seg}$	length of segment
ds_max	$\Delta s_{\max}$	maximum step size
ds_min	$\Delta s_{\min}$	minimum step size
g_cars	$d\hat{l}$	unit length vector along segment
ihit	$i_{\rm hit}$	index of fux surface intersected
ilo	$i_{\rm low}$	index of surface below start of step
irho_int	$i_{\rho,{\rm int}}$	indices of flux surfaces intersected
izone_int	$i_{z,{\rm int}}$	indices of zones entered at intersections
k_seg	$k_{\rm seg}$	option for coordinates specifying segments
mhalf	m_h	step size halving exponent
n_int	$n_{\rm int}$	number of intersections with surfaces
n_rho	n_ρ	number of flux surfaces
n_seg	$n_{\rm seg}$	number of segments in path
r000	R_0	characteristic major radius of torus
r_car	$\vec{x}$	Cartesian coordinates
rcar_int	$\vec{x}_{\rm int}$	Cartesian coordinates of intersections
r_cars	$\vec{x}_o$	Cartesian coordinates at beginning of segment
r_cyl	$\vec{r}$	cylindrical coordinates
rcyl_int	$\vec{r}_{\rm int}$	cylindrical coordinates of intersections
r_flx	$\vec{\rho}$	flux coordinates
rflx_int	$\vec{\rho}_{\rm int}$	flux coordinates of intersections
rho	ρ	set of flux surface labels
rhomax	$\rho_{\max}$	radial grid at plasma boundary
r_seg	$\vec{r}_{\rm seg}$	coordinates of segment nodes
s_int	$s_{\rm int}$	distance from beginning of path to intersections
sdotb_int	$d\hat{l}\cdot\vec{B}/\|B\|$	sine of angle of path with B at intersections
sdotphi_int	$d\hat{l}\cdot\hat{\phi}$	sine of angle of path with ϕ at intersections
sseg	$s_{\rm seg}$	distance from beginning of path to start of segment
tau	τ	2D Jacobian of coordinate transformation
tol	ϵ_R	tolerance for $\Delta\rho_{\min}$ and $\Delta s_{\min}$

8 Code Diagrams

An $*$ before the names of routines each module indicate that they are **PUBLIC** and can be accessed directly by the user. If they are listed in the tree of another routine they are not given a separate listing. See the **AJAX** documentation for a description of the **AJAX_...** routines and the **XPLASMA** documentation for a description of the **EQ...** routines.

```
TRACK_MOD

  *TRACK
     |--AJAX_B
     |--AJAX_CAR2CYL
     |--AJAX_CYL2CAR
     |--AJAX_CYL2FLX
     |--AJAX_FLX2CYL
     |--AJAX_GLOBALS
     |--*TRACK_D
     |     |--AJAX_CAR2CYL
     |     |--AJAX_CYL2FLX
     |--*TRACK_G
           |--AJAX_CYL2CAR

AJAX_XPLASMA_MOD

  *AJAX_B
     |--EQ_XCYL
     |--EQXYZ_BGET

  *AJAX_CAR2CYL
     |--EQ_RCYL

  *AJAX_CYL2CAR
     |--EQ_XCYL

  *AJAX_CYL2FLX
     |--EQ_INV
     |--EQ_GETJAC

  *AJAX_FLX2CYL
     |--EQ_RZGET

  *AJAX_GLOBALS
     |--EQ_RHOLIM
     |--EQ_RZGET
```

References

[1] S.E. Attenberger, W.A. Houlberg, S.P. Hirshman, "Some Practical Considerations Involving Spectral Repersentations of 3D Plasma Equilibria," *J. Comput. Phys.* **72** (1987) 435.