A User's Guide to TRAK: A Customizable Particle Following Code

J.W. Johnson

October 1984

UWFDM-597

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

A User's Guide to TRAK: A Customizable Particle Following Code

J.W. Johnson

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

October 1984

UWFDM-597

A User's Guide to TRAK: A Customizable Particle Following Code

J. W. Johnson

Fusion Technology Institute
Nuclear Engineering Department
University of Wisconsin–Madison
Madison, Wisconsin 53706

October 1984

UWFDM-597

Contents

List of Figures

List of Tables

1 Introduction

TRAK is a simple, user-extensible particle following code which runs on the CDC 7600 or Cray-1 computers at the National Magnetic Fusion Energy Computer Center. Briefly, TRAK integrates the guiding center equations using a fourth order predictor-corrector integrator. While some users may not find the version of the code described here very useful, TRAK is easily modified due to its modular construction. This report is an attempt to document some of the features of the code so that prospective users will be able to modify TRAK to suit their own purposes.

In the next section, the subroutine which does the actual integration of the particle orbits will be described. In this context, integration methods, the equations of motion, magnetic and electric field models and output will be discussed. Section 3 contains a description of the "top level" of TRAK and will address issues of initialization and job control. In Section 4 some subroutines which are included in TRAK but not used in the current version will be described. These include an interpolation subroutine and a pitch angle scattering subroutine. Care and use of the code are documented in Section 5. There are three Appendices to this report; in the first, some additional applications of the code are discussed. These applications require nontrivial but straightforward modifications to the code. Appendix B takes up the question of plotting data from TRAK — a brief guide to the code PLOTS is included. Finally, Appendix C gives an example of how the output from TRAK can be used in a data analysis program.

Before continuing, it would be a good idea for the reader to obtain a copy of TRAK. See Section 5.1 for the proper incantation to retrieve the code from the mass storage facility at the NMFECC.

2 Subroutine STEP

In this section, the portion of the code which does the orbit integrations will be detailed. It may seem strange to discuss this subroutine before the main part of the code (which is described in Section 3), but the orbit calculation is the main purpose of TRAK. The rest of the code can be considered as job control or initialization before STEP is called.

STEP takes five arguments; the starting position, energy and cosine of the pitch angle, a parameter called `mflag` and the output parameter `nwrite` (which is called `nskip` in

the main program) which will be discussed in Section 2.5. Other data, such as the time step, maximum integration time and angular modulus are passed through the common block **/endchk/**. In the current version of TRAK, **mflag** can only take on two values, corresponding to normal execution and the mirror end condition, *i.e.* the particle escapes from the mirror. In some previous versions of the code, the value of **mflag** could also specify a poloidal orbit end condition, the use of the interpolation subroutine as well as some output criteria.

Normal execution means that there are only two end conditions; when the running time exceeds **tmax** or when the number of angular periods exceeds **nangpd**.

The first section of STEP does the initialization of the integration variables and outputs the first point. The subroutine DERIV contains the implementation of the guiding center equations (see Section 2.1). A Runge–Kutta integrator is used for the first three time steps, then a predictor–corrector scheme is used. The integration scheme will be discussed in Section 2.2. During the integration, the subroutine PERIOD is used to keep the ϕ coordinate in the range $0 \leq \phi <$ **angmod**.

In this version, neither the interpolation subroutine nor the pitch angle scattering subroutine are called; comments indicate where the call statement for each routine should be placed. When integration terminates, the character string **endmsg** can be used to pass a message back to the main program (or a controlling subroutine).

2.1 The Guiding Center Equations

The equations of motion are found by carrying out the guiding center expansion as in Northrop[1]. For a particle of mass m and charge q, the full (non-relativistic) equations of motion are:

$$\frac{d^2\underline{r}}{dt^2} = \underline{g} + \frac{q}{m}\left(\underline{E} + \frac{d\underline{r}}{dt} \times \underline{B}\right) \tag{1}$$

where $\underline{g}$ is the acceleration due to gravity and $\underline{E}$ and $\underline{B}$ are the electric and magnetic fields respectively. $\underline{r}$ is the particle position.

The position of the particle can be decomposed into the position of the guiding center $\underline{R}$ and a vector from the guiding center to the particle, $\underline{\rho}$. Thus we can write:

$$\underline{r} = \underline{R} + \underline{\rho} \tag{2}$$

Substituting this relationship into the equations of motion, and realizing that $|\underline{\rho}|$ is of $O(\epsilon)$,

we can carry out the guiding center expansion. The result[1] is:

$$\ddot{\underline{R}} = \underline{g} + \frac{q}{m}\left(\underline{E} + \dot{\underline{R}} \times \underline{B}\right) - \frac{\mu}{m}\nabla B \tag{3}$$

$$\dot{\underline{R}}_{\perp} = \frac{m}{qB}\left(\underline{g} - \ddot{\underline{R}}\right) \times \underline{b} + \frac{\underline{E} \times \underline{b}}{B} - \frac{\mu}{qB}\nabla B \times \underline{b} \tag{4}$$

where $B = |\underline{B}|$, μ is the magnetic moment, $\underline{b}$ is a unit vector in the $\underline{B}$ direction and $\dot{\underline{R}}_{\perp}$ is the motion of the guiding center perpendicular to $\underline{b}$. Terms of order ϵ^2 have been neglected. Note that the first term in Equation 4 contains the gravitational and curvature drifts, the second term is the $\underline{E} \times \underline{B}$ drift velocity and the third term is the ∇B drift.

This system of equations can be simplified by introducing a parallel velocity $u = \underline{b} \cdot \dot{\underline{R}}$. Differentiating, we have:

$$\dot{u} = \underline{b} \cdot \ddot{\underline{R}} + \dot{\underline{b}} \cdot \dot{\underline{R}} \tag{5}$$

$$\dot{u} = \underline{b} \cdot \underline{g} + \frac{q}{m}\underline{b} \cdot \underline{E} - \frac{\mu}{m}\underline{b} \cdot \nabla B + \frac{\partial \underline{b}}{\partial t} \cdot \dot{\underline{R}} + \dot{\underline{R}} \cdot \nabla \underline{b} \cdot \dot{\underline{R}} \tag{6}$$

All that remains is to eliminate the $\ddot{\underline{R}}$ in the expression for $\dot{\underline{R}}_{\perp}$. Because $\ddot{\underline{R}}$ occurs with a coefficient of $\frac{m}{qB}$ we can neglect the $\mathcal{O}(\epsilon)$ terms. Thus $\ddot{\underline{R}}$ can be expressed as:

$$\ddot{\underline{R}} = \frac{d}{dt}\left(u\underline{b} + \underline{u}_E\right) \tag{7}$$

$$\ddot{\underline{R}} = \dot{u}\underline{b} + u\frac{d\underline{b}}{dt} + \frac{d\underline{u}_E}{dt} \tag{8}$$

where $\underline{u}_E$ is the $\underline{E} \times \underline{B}$ drift velocity, which may be large enough to be included. The other drift velocities are all of order ϵ. Whether the $\underline{u}_E$ term is large enough to be included depends on the electric and magnetic field models, however, it is not "wrong" to include it even if it is small. The first term in Equation 8 can be neglected because the cross product of $\ddot{\underline{R}}$ and $\underline{b}$ is what we need. If we define $\underline{u}' = u\underline{b} + \underline{u}_E$ we can write the result in a compact form:

$$\ddot{\underline{R}} = u\left(\frac{\partial \underline{b}}{\partial t} + \underline{u}' \cdot \nabla \underline{b}\right) + \frac{\partial \underline{u}_E}{\partial t} + \underline{u}' \cdot \nabla \underline{u}_E \tag{9}$$

If we insert this result back into Equation 8 we have:

$$\dot{\underline{R}} = \underline{u}' + \frac{\underline{b}}{qB} \times \left(\mu \nabla B + m\left(u\frac{\partial \underline{b}}{\partial t} + u\underline{u}' \cdot \nabla \underline{b} + \frac{\partial \underline{u}_E}{\partial t} + \underline{u}' \cdot \nabla \underline{u}_E\right) - m\underline{g}\right) \tag{10}$$

Equations 6 and 10 represent the full guiding center equations of motion. I have been fairly sloppy about including some second order terms (for example in the $\dot{\underline{R}} \cdot \nabla \underline{b} \cdot \dot{\underline{R}}$ term),

but these make little difference. If we make some assumptions about the magnetic and electric fields, the guiding center equations can be simplified. For example, suppose that the $\underline{u}_E$ term in $\ddot{\underline{R}}$ can be neglected. Then Equation 10 can be simplified to:

$$\dot{\underline{R}} = u\underline{b} + \underline{u}_E + \frac{\underline{b}}{qB} \times \left(\mu \nabla B + m \left(u \frac{\partial \underline{b}}{\partial t} + u^2 \underline{b} \cdot \nabla \underline{b} \right) - m\underline{g} \right) \tag{11}$$

If we further assume that the electric and magnetic fields do not vary in time, we have

$$\dot{u} = \underline{b} \cdot \underline{g} + \frac{q}{m} \underline{b} \cdot \underline{E} - \frac{\mu}{m} \underline{b} \cdot \nabla B + \dot{\underline{R}} \cdot \nabla \underline{b} \cdot \dot{\underline{R}} \tag{12}$$

$$\dot{\underline{R}} = u\underline{b} + \underline{u}_E + \frac{\underline{b}}{qB} \times \left(\mu \nabla B + m u^2 \underline{b} \cdot \nabla \underline{b} - m\underline{g} \right) \tag{13}$$

where $\underline{u}_E$ is defined by:

$$\underline{u}_E = \frac{\underline{E} \times \underline{b}}{B} \tag{14}$$

This system of equations may be further simplified by neglecting the "gravitational" acceleration, which is frequently of little interest. Equation 12 may be slightly modified to improve energy conservation[2]. The result is:

$$\dot{u} = \underline{b} \cdot \underline{g} + \frac{q}{m} \underline{b} \cdot \underline{E} - \frac{\mu}{m} \underline{b} \cdot \nabla B + \dot{\underline{R}} \cdot \nabla \underline{b} \cdot \dot{\underline{R}} - \frac{\mu \underline{b}}{qB} \times (u\underline{b} \cdot \nabla \underline{b}) \cdot \nabla B \tag{15}$$

This "extra" term has a simple physical explanation. The first term of Equation 15 is the change in $\dot{u}$ caused by the particle moving into a region of different magnetic field strength. This is from the ***parallel*** motion of the particle. The new term is similar, except that the motion is the result of the ***drift*** motion of the particle, specifically the curvature drift. The new term is one of the higher order terms left out in Equation 12.

The importance of this additional term depends on how big the curvature drift velocity is. In most cases, it turns out that this term improves energy conservation by less than 10%. If the magnetic geometry is highly curved the improvement can be 40% or more. The inclusion of this term is up to the user; the improvement in energy conservation may not be worth the small increase in CPU time. In Section 5.3, we will see that the error in energy conservation from the guiding center expansion is not usually very large.

The subroutine DERIV implements Equations 12 and 13. DERIV calls FIELD and EF to compute $\underline{B}$ and $\underline{E}$; see Sections 2.3 and 2.4 for details. In this version of TRAK there is no additional force acting on the particle, so $\underline{g}$ has been set to zero. An implementation of the full guiding center equations (Equations 6 and 10) is contained in the file `deriv1`;

see Section 5.1 if you wish to access this file. The extra "energy conservation" term in Equation 15 has been added to Equation 6.

It should be noted that with one exception the physical quantities used in TRAK are all assumed to be in SI units. The exception is the particle energy (a dependent quantity) which is in electron volts.

2.2 The Predictor–Corrector Method

A fourth order predictor–corrector algorithm is used to do the integration. This method was chosen because it gives fourth order accuracy with only two function evaluations per step, while other fourth order schemes, like the Runge–Kutta method, require four function evaluations. For the case of particle following the function evaluations (*i.e.* calls to DERIV) require calculation of the magnetic fields, and thus require many operations.

A disadvantage of the predictor–corrector method is that it requires data at three previous points before it can calculate the fourth point. This means that some other method must be used to do the first several integrations; in TRAK the Runge–Kutta method is used. Also, the predictor–corrector method is limited to a fixed step size.

The predictor–corrector method (and the Runge–Kutta method) are discussed in many books on numerical analysis; see for example references [3] and [4]. Briefly, the predictor step is defined by:

$$\tilde{x}_{n+1} = x_n + \frac{\Delta t}{24}\left(55v_n - 59v_{n-1} + 37v_{n-2} - 9v_{n-3}\right) \tag{16}$$

where x_n represents the position at the nth step, v_n is the velocity at the nth step, Δt is the time step and $\tilde{x}_{n+1}$ is the "prediction" for the position at the $n+1$st step. The corrector step can be expressed as:

$$x_{n+1} = x_n + \frac{\Delta t}{24}\left(9\tilde{v}_{n+1} + 19v_n - 5v_{n-1} + v_{n-2}\right) \tag{17}$$

For our purposes, the quantities x and v represent a four dimensional position and velocity — the position is defined by $\underline{R}$ and u, the velocity is $\dot{\underline{R}}$ and $\dot{u}$ which are defined in Section 2.1. It is common to include the "time of flight" along the trajectory as a fifth position variable.

In subroutine STEP there are three separate phases to the integration procedure. First, the Adams–Bashforth predictor step (Equation 16) is applied to the four principal integration variables; the three spatial variables and the parallel velocity. This yields an estimate

for the new position, which is in array xx. Next, the arrays x and v are shuffled back, deleting the oldest point and making space for the new point which is about to be calculated. Finally, the Adams–Moulton corrector step (Equation 17) is applied to all of the integration variables. The new velocity is calculated at this point, and the whole integration procedure is repeated.

Other integration variables besides time can easily be included; just increase the dimension of the arrays x and v and insert an expression for the derivative of the new quantity in DERIV. For example, suppose we want to compute the average value of the parallel velocity. This involves integrating $v_{||}$ over the particle orbit and dividing by the time. First, increase the dimension of the arrays from 5 to 6. Next, note that the value of $v_{||}$ is located in `x(4)`. In DERIV, we add the expression:

```
v(6) = x(4)
```

All that remains is to change the limits on the loops used in the predictor and corrector steps. Note that the average of $v_{||}$ is derived from the four position variables, so it is not necessary to include it in the predictor step. This is true of most of the "auxiliary" integration variables.

2.3 The Magnetic Field Model

One of the most important issues in a particle following code is the question of a magnetic field model. In general there are two kinds of models for $\underline{B}$; a "real" field based on the Biot–Savart law applied to a current distribution and an "approximate" field which is usually a simplified form of the "real" field. The advantage of the the "approximate" field is that it is generally very simple to evaluate $\underline{B}$ at an arbitrary point; unlike the "real" field which can be very time consuming to evaluate. The "real" field has good curl and divergence properties, which is not necessarily true of an "approximate" field.

The magnetic field model presented in this version of TRAK combines the good curl and divergence properties of a "real" magnetic field with the ease of evaluation of an "approximate" field. The model consists of three parts: a long, thin filament along the $\hat{z}$ axis, used to simulate B_ϕ in toroidal geometry; a constant vertical field; and a number of current loops which can be used to simulate the poloidal field in toroidal devices, a simple magnetic mirror or a quadrupole. It should be noted that the current loops are constrained to lie in a plane perpendicular to the $\hat{z}$ axis with the center of the loop on the axis.

For particle following, we are interested not only in $\underline{B}$ but also in the gradients of $\underline{B}$,

denoted by $\nabla \underline{B}$. Analytic expressions for $\underline{B}$ (and the vector potential) for a current loop are easily calculated (see Jackson[5] for example), but computing the four nonzero quantities in $\nabla \underline{B}$ is a little more complicated. These calculations are given in reference 2, and are implemented in subroutine BCIRS, which is called by FIELD. After returning from BCIRS, the entries in `db` which represents $\nabla \underline{B}$ look like this:

$$\nabla \underline{B} = \begin{pmatrix} \frac{\partial B_r}{\partial r} & \frac{\partial B_\phi}{\partial r} & \frac{\partial B_z}{\partial r} \\ \frac{1}{r}\frac{\partial B_r}{\partial \phi} & \frac{1}{r}\frac{\partial B_\phi}{\partial \phi} & \frac{1}{r}\frac{\partial B_z}{\partial \phi} \\ \frac{\partial B_r}{\partial z} & \frac{\partial B_\phi}{\partial z} & \frac{\partial B_z}{\partial z} \end{pmatrix} \tag{18}$$

From this, we can compute $\nabla|\underline{B}|$, which is also needed in the guiding center equations. Given:

$$\nabla|\underline{B}|^2 = 2|\underline{B}|\,\nabla|\underline{B}| \tag{19}$$

$$|\underline{B}|^2 = B_r^2 + B_\phi^2 + B_z^2 \tag{20}$$

it is easy to compute $\nabla|\underline{B}|$ from $\underline{B}$ and $\nabla \underline{B}$.

Now that we have computed $\nabla|\underline{B}|$, the only remaining magnetic field quantity that must be computed is $\nabla \underline{b}$ where $\underline{b}$ is the unit vector along the magnetic field. First, we must add two terms to $\nabla \underline{B}$ to reflect the choice of cylindrical coordinates for our computations.

$$\nabla \underline{B} = \begin{pmatrix} \frac{\partial B_r}{\partial r} & \frac{\partial B_\phi}{\partial r} & \frac{\partial B_z}{\partial r} \\ \frac{1}{r}\frac{\partial B_r}{\partial \phi} - \frac{B_\phi}{r} & \frac{1}{r}\frac{\partial B_\phi}{\partial \phi} + \frac{B_r}{r} & \frac{1}{r}\frac{\partial B_z}{\partial \phi} \\ \frac{\partial B_r}{\partial z} & \frac{\partial B_\phi}{\partial z} & \frac{\partial B_z}{\partial z} \end{pmatrix} \tag{21}$$

The addition of the curvature terms makes calculation of $\nabla \cdot \underline{B}$ and $\nabla \times \underline{B}$ very easy; for example, $\nabla \cdot \underline{B}$ is just the trace of $\nabla \underline{B}$. Just as we were able to calculate $\nabla|\underline{B}|$ from $\nabla \underline{B}$, so we can compute $\nabla \underline{b}$ from $\nabla \underline{B}$ and $\nabla|\underline{B}|$. If we remember that $\underline{B} = |\underline{B}|\underline{b}$ we can write:

$$\nabla \underline{B} = |\underline{B}|\,\nabla \underline{b} + \frac{1}{|\underline{B}|}\,\tilde{\nabla} \underline{B} \tag{22}$$

where $\tilde{\nabla} \underline{B}$ is the outer product of $\nabla|\underline{B}|$ and $\underline{B}$. Equation 22 determines $\nabla \underline{b}$.

Some further notes on subroutine FIELD; the argument `io` was added to allow the magnetic field and other quantities of interest to be printed out after the calculation. This same feature is present in DERIV and EF, it is intended for debugging purposes. The variable `ibf` in common block `/bfield/` is a flag used to prevent multiple evaluations of

the magnetic field at a given position. It is particularly useful if subroutine PSCATR (Section 4.1) is included in the code.

2.4 The Electric Field Model

A relatively simple electric field model is included in TRAK. The electric field is calculated in subroutine EF from parameters which are read in at run time. This allows particle orbit calculations to be done for different electric field values from the same input file.

The model for $\underline{E}$ consists of three parts: a "toroidal" field component E_ϕ, E_z which is computed from a potential and a time dependent piece $\hat{E}_z$ to represent the "near field" of ICRF heating. E_ϕ is a constant and was used in earlier versions of TRAK which simulated a tokamak. The potential is assumed to be a Gaussian function, with three parameters determining its actual shape. These are: `phi0` the amplitude; `zphi0` the z location of the center of the potential; and `wphi0` the "width" of the potential. Given the potential, E_z is easily computed using the relation $E_z = -\nabla\Phi$.

Calculation of $\hat{E}_z$ is a bit more complicated. It is assumed that $\hat{E}_z$ is a function of z and t only. The time dependence is assumed to be a cosine, with `omega` as the frequency and `wphase` the offset. `wez0` is the amplitude of the "wave". The shape of the "wave" is calculated using cubic B-spline interpolation[6][7]. The spline coefficients are read in as part of the initialization process, and the actual evaluation of the function value occurs in subroutine SVAL. The program used to compute the spline coefficients is WAVE. Use of spline interpolation allows many different shapes to be tried for $\hat{E}_z$ without recompiling TRAK.

2.5 Format of the Output File

A fairly flexible output file format is used in TRAK. The format used to write data is:

```
a4,e16.8,3(4x,e16.8)
```

where the four `e16.8` fields are used for data and the `a4` field is used for a label. Labels usually consist of three characters, insuring a space between the label and the first data field. There are four labels which have a special meaning and should not be used for anything else: "`par`" or "`lin`" for the beginning of a set of data for a single particle, "`end`" for the end of the data block and "`tim`" for the end of the data file. More than one data block can be contained in a single data file.

There are two distinct ways in which data is written into the data block. The first is when "global" data are written. Near the beginning of STEP, the initial magnetic and electric fields are written out. If a scattering subroutine were included, the scattering frequency should also be written out. Another example is in PERIOD, when the angular variable wraps around, some data is written out. The bulk of the data block comes from the write statements in OUTFIL. Here the user can specify what is to be written at each time step.

The user can use almost any character string for a label. The four labels mentioned above should not be used for anything else; similarly the labels used in PERIOD should not be used. This ensures compatibility with PLOTS (see Appendix B). It is also a good idea to have the first label written from OUTFIL remain "x ", with the data on this record consisting of the particle's (r, ϕ, z, t) position. Also, use of the labels "`nil`" and "`;;;`" is discouraged; these are ignored by some data analysis programs. The user is cautioned that each label should be unique.

Printing out the particle's position state at every time step can lead to an extremely large data file. The variable which controls the frequency at which OUTFIL is called is `nwrite`. `nwrite = 0` will call OUTFIL at every step, a value of `1` will skip one step between calls, etc. A negative value of `nwrite` will suppress the calls to OUTFIL. Because the final position of the particle is frequently very important, if `nwrite` is greater than or equal to `-1` the final state will be printed, unless it already has been printed. Thus to suppress *all* calls to OUTFIL, use `nwrite = -2`.

3 The Main Program

In this section, the "top level" of TRAK will be discussed. The main program performs two main duties: initialization and "job control", i.e. it uses input parameters to determine how STEP will be called. Also, utility subroutines such as GOOF and TTYOUT will be described.

3.1 Subroutine RCONTRO

After setting values for some constants, the main program calls RCONTRO. The purpose of this subroutine is twofold: first to assign and open the i/o channels to the various files and the terminal, and second, to read the file that contains the magnetic field model.

Use of the FORTLIB routine `link` allows the user to specify some or all of the input and output files on the command line, as is shown in Section 5.2. Default names for the files are provided.

The code for reading the magnetic field model is fairly well commented; an example of a `model` file is given in Section 5.2. One of the interesting and useful features of the `model` file is that it may contain comments.

Before returning control to the main program, a call to ZLOOPS is made. ZLOOPS computes the maximum and minimum values for the $\hat{z}$ coordinate of the current loops in the magnetic field model. This information is necessary for the mirror termination condition in STEP.

3.2 The Read/Execute Loop

Following the call to RCONTRO, the spline coefficients which form part of the electric field model are read in. Next, default values for the namelist quantities are established. The namelist variables contain information about the electric field model, the charge and mass of the particle to be followed and important job control information. The read from the namelist is the first statement in the Read/Execute loop.

Inside the loop, further initialization is done. If the user decides to specify the particle by name rather than by mass and charge, the appropriate values of mass and charge are assigned to the variables. Note that this will override the assignment of the mass and charge from the namelist read. If the scattering subroutine is to be used for the first time, the seed for the random number generator is chosen. Finally, the phase of the wave portion of the electric field model is converted to radians.

The key job control parameter in the main program is **`flag`**. The next section of the code has an **`if-then-else`** structure based on the value of **`flag`**. If **`flag`** is set to **`end`**, execution terminates and the CPU time used will be printed in the output file.

If **`flag = eject`**, the current output file is closed and a new one is opened. The name of the new file is taken from the namelist variable **`filename`**. If **`flag = debug`**, the subroutine DDT is called. This allows the user to interactively specify a location and have various quantities of interest, like the drift velocity or magnetic field for example, printed to the terminal or output file. The user should consult the comments in DDT before the **`debug`** option. One suggestion for further improvements to TRAK would be to allow STEP to be called from DDT; thus allowing the user to interactively choose the initial point for the

orbit calculation. This could be quite useful for some applications.

If `flag = scan`, data for plots of the electric field, potential and $|\underline{B}|$ are generated. Execution will terminate when the `scan` option is used; this is because some of the namelist variables are used in ways that are inconsistent with their use elsewhere.

`flag = rip` allows a large number of orbit calculations to be done without specifying each one in the command file. In this mode, the cosine of the pitch angle is varied, keeping all other arguments to STEP constant. Note that `mflag = 20` sets up the mirror end condition.

If `flag` has not matched one of the above options, it is assumed that the user desires a simple call to STEP. The remaining clause in the `if-then-else` structure assigns a value to `mflag` based on `flag`, and then calls STEP.

Before TRAK terminates, the amount of CPU time used is computed and written to the `output` file.

3.3 GOOF and TTYOUT

These two subroutines can be called from almost any other subroutine or the main program. GOOF provides a simple means of handling errors; should an error condition be detected, for example some unknown object in the `model` file, GOOF is called with the appropriate arguments and an error message is written to the `output` file and execution is terminated. The call to `exit` from GOOF does not delete the dropfile so the NMFECC debugging program DDT may be used.

TTYOUT provides a means for the user to get information on the current state of the calculation. When the user types anything on the terminal, the variable `iiflag` is set to 1. At many places throughout the code, a check is made on the value of `iiflag`. If it is 1, a call to TTYOUT is made. In TTYOUT, the string that the user typed is compared with various conditions, much like the value of `flag` is checked in the main read/execute loop. If the string matches one of the conditions, the desired information will be typed to the terminal. Control is returned to where the call to TTYOUT was made or execution is terminated, depending on what the user desires. Information is passed to TTYOUT through its four arguments and the common block `/tty/`.

4 Bells and Whistles

In this section some of the subroutines which are included in TRAK, but not used in the current version of the code are discussed. These subroutines include a pitch angle scattering routine and an interpolation routine, both of which can be very useful. Subroutines to compute the first and second derivatives of a function defined by B-spline interpolation are also included.

4.1 Pitch Angle Scattering

Pitch angle scattering is modeled by a Lorentz collision operator:

$$\frac{\partial f}{\partial t} = \frac{\nu}{2}\frac{\partial}{\partial \lambda}(1-\lambda^2)\frac{\partial f}{\partial \lambda} \tag{23}$$

where f is the particle phase space density, ν is the collision frequency, and λ is the cosine of the pitch angle. In reference [8] a Monte Carlo equivalent operator based on the binomial distribution is derived:

$$\lambda = \lambda_0(1-\nu\tau) + x\sqrt{(1-{\lambda_0}^2)\nu\tau} \tag{24}$$

where λ is the cosine of the pitch angle after scattering, λ_0 is the cosine of the pitch angle before scattering, and τ is the time step. x is randomly chosen to be ± 1. The advantage to using this collision operator is that $|\lambda|$ will never exceed 1. If τ is chosen to be smaller than the integration time step (so the scattering operator is applied several times for one time step), the net result will approach a Gaussian distribution. Instead of following this procedure, x is chosen to be a normally distributed random number ($\sigma^2 = 1$), which is obtained using the Box–Muller transformation[9]. Thus the scattering operator resembles those described in references [10] and [11]. The advantage is that this is much faster, however, it will be possible for $|\lambda|$ to exceed 1. Fortunately, for small ν, the problem with $|\lambda|$ only occurs for particles with $|\lambda| \approx 1$, *i.e.* the well circulating or passing particles. So in practice, if the value of x yields a nonphysical result for λ for these particles, a new x is chosen.

This scattering operator was tested by following a particle for several hundred thousand time steps, using a large value of ν to simulate the actual collision frequency calculated from the Braginskii collision frequency. It is expected that the distribution of the particle's λ values should approach a uniform distribution, and this is indeed the case.

This scattering subroutine PSCATR works by calculating a new value of μ and $v_{\|}$ given the new value of λ. Thus it should be called *before* the call to DERIV at the new particle position. Also note that the collision frequency is assumed to be a constant and is not calculated from any plasma parameters. The subroutine RND computes two normally distributed random numbers.

4.2 Interpolation

A good interpolation subroutine is essential to many applications of a code like TRAK. A familiar application is the creation of "punch plots" of flux surfaces in complicated magnetic geometries. The interpolation routine INTERP is based on the Adams–Moulton polynomial which is implicit in the corrector step of the integration. This is probably not the optimal choice for an interpolating polynomial, and other choices are being investigated.

The subroutine is well commented, so the discussion here will be brief. Seven arguments are passed to INTERP; `ink` is the pointer to the known dependent variable, `tolnce` is a precision parameter, `x` is the current position, `xold` is the previous position, `xx` is the returned (interpolated) position, `f` is the complete velocity array and `ds` is the integration step. Note that `xx(ink)` should contain the known value.

First, a check is made to see if `x` or `xold` is within `tolnce` of the interpolation point. Assuming this check fails, the polynomial coefficients are unwrapped from `f`. Then INTERP iterates on the normalized variable `del` until the convergence criteria are satisfied. The values of `xx` are then calculated and returned.

Should the iteration loop not converge, GOOF is called and program execution terminates. This may not be desirable for all applications; an alternative suggestion would be to return values based on a linear fit between `x` and `xold`.

One possible cause of convergence problems is the use of the subroutine PSCATR. Because PSCATR must be called before the call to DERIV, and DERIV must be called before INTERP, the scattering algorithm can cause anomalous behavior of the interpolation algorithm. It is suggested that before INTERP is called, PSCATR should not be called. This presents no problem if INTERP is used as part of the end criteria of STEP; otherwise DERIV must be called twice, though two evaluations of $\underline{B}$ can be avoided by using the flag `ibf`.

4.3 Derivatives of B-splines

The functions SDIF1 and SDIF2 compute the first and second derivatives of a function which is defined by B-spline coefficients. They are analogous to the function SVAL which is used as part of the electric field model. These functions are not used in the current version of TRAK because the guiding center equations do not contain terms with $\frac{\partial E_z}{\partial z}$, for example. Those terms are small for the test fields used with TRAK and were neglected (see Section 2.1).

5 Running TRAK

In this section, the procedures used to obtain a working copy of TRAK will be detailed. Samples of the necessary input files will be given and some hints on error checking are also given.

5.1 Obtaining the Code

The TRAK source file is stored in the FILEM mass storage facility at the NMFECC along with several other files that will be of interest to the prospective user. To access these files, run the FILEM program and when the . prompt appears, type `list .14353 .trak` to generate a listing of the files. The command to retrieve a file is `read`. It would be a good idea to read the NMFECC FILEM documentation.

The source file is `trak5` where the `5` is the current version number. The version number may change from time to time as bugs are found and fixed. Should you have problems in accessing the files contained in FILEM, please use the TELL electronic mail program to inform me; my user number is 14353.

The user should note that the source file was written to FILEM from a Cray and should be read from one of the Cray computers. The code will also run on the CDC 7600. Because the code is not vectorizable, identical runs will take approximately 2.4 times as much CPU time on the CDC 7600 compared to a Cray-1.

The source file contains COSMOS statements which will use the appropriate programs to compile and link the source file, depending on which machine the code is resident. All that the user must do is type

```
cosmos i=trak5 / t v
```

```
circle
     0.000000000e+00      0.000000000e+00     -5.000000000e-01
     0.000000000e+00      0.000000000e+00
     1.000000000e+00      0.000000000e+00
                          5.000000000e+04
circle
     0.000000000e-00      0.000000000e-00      5.000000000e-01
     0.000000000e+00      0.000000000e+00
     1.000000000e+00      0.000000000e+00
                          5.000000000e+04
comment
 a simple mirror set for testing purposes.
end
```

Table 1: A Sample Magnetic Field Model File

t and v are the time and value. The code will be compiled and linked. The executable file is **xtrak**.

Other files which may be of interest are: **plots77**, a program which plots the output data from TRAK (see Appendix B); **wave3**, the program used to generate the B-spline coefficient representation of the time dependent electric field; **deriv1**, a version of the subroutine DERIV which includes all of the terms described in Section 2.1; **pf.dvi**, which is this report minus the three figures contained in **f3trak0x**; **gobble12**, a simple data analysis program (see Appendix C); and several sample input files.

5.2 Samples of Input Files

Three input files are needed to run TRAK and one output file is generated. The input files have default names of **input**, **model** and **coeff**. The default output file name is **output**. The form of the data written to **output** is specified in subroutine OUTFIL. Briefly, the file **coeff** contains the spline coefficients needed to compute the spatial dependence of the electric field due to the wave. The coefficients are stored in binary form and are generated by the program WAVE. The file **model** contains the information for the magnetic field model and **input** is the file which has the namelist variables which are needed by the "top level" of TRAK.

Table 1 contains a sample of a **model** file. The file gives the parameters necessary to

```
xstart(1) = 0.5 xstart(2) = 0.0 xstart(3) = -0.1
nskip = 99 estart = 9.00e+2 tmax = 2.0e-6 dt = 1.0e-10
phi0 = 100.0 zphi0 = -0.1 wphi0 = 0.05
particle = "electron" flag = "mirror"
omega = 5.0e+6 wez0 = -500.0 wshift= -0.1 wsigma= 0.05
cospa = -0.8  $
cospa = -0.4  $
cospa = 0.0  $
cospa = 0.4  $
cospa = 0.8  $
flag = "end"  $
```

Table 2: A Sample Input File

calculate the magnetic field properties of a simple magnetic mirror.

An example of an `input` file is contained in Table 2. Note that this file is in a namelist format; the **$** delimits the data that is to be read on each pass through the read loop of the main program. Thus this file contains input information for 5 particle orbit calculations. Of particular importance are the variables **xstart**, **estart** and **cospa** which are the starting position, kinetic energy and cosine pitch angle respectively.

When TRAK is run, the user may specify the input and output file names on the command line. This is because the FORTLIB subroutine `link` is used to open or create the files. For example, the user might type

```
xtrak i=intest, m=mirror, c=coeff1, o=outtest / t v
```

to execute the code.

5.3 Error Checking

There are two fairly sensitive ways of detecting errors using TRAK and PLOTS (see Appendix B for information about PLOTS). In the absence of collisions and if the wave amplitude is zero, the total particle energy and the magnetic moment should be conserved. The implementation of the guiding center equations assumes that μ is conserved, so the fluctuation in the particle energy gives a good idea of the accuracy of the integration scheme *and* the accuracy of the guiding center expansion. The total energy has three components, the potential energy from the electric field, the energy in the parallel velocity and the

perpendicular kinetic energy which is proportional to μ.

Because we have neglected some of the higher order terms in the guiding center expansion, energy will not be exactly conserved. A periodic ripple can be observed in the energy; the period should be the bounce frequency or the poloidal orbital frequency, depending on the magnetic configuration. However, the ripple should *not* be a function of the integration step size and after each bounce (or orbit) the energy should return to its original value. If there is a roughly linear drift of the energy in addition to the ripple, the integration step size (Δt or `dt`) is too large. If the step size is halved, the slope of the drift decreases by a factor of 32 which is as expected for a fourth order integration scheme.

Figures 1, 2 and 3 show the change in energy for three different values of Δt. The ripple due to the guiding center expansion is clearly shown in Figure 1. Comparing the slopes of the energy plots in Figures 2 and 3, we see that the energy "drift rate" decreases by a factor of approximately 30 as the time step is halved. It is hard to make a similar comparison between the slopes of Figures 1 and 2 because the slope of Figure 1 is nearly zero.

The time steps used to generate these plots may seem quite small. This is because the orbit calculations were made for an electron. The energy drift is primarily due to the accuracy of the parallel velocity integration. For heavier particles (at the same energy) this problem will be less severe.

Another test of the integration scheme can be done by integrating for a given time, then reversing the integration and integrating back towards the initial point. The distance between the final point and the initial point will be a measure of the integration accuracy. Again, if the step size is halved, the distance should decrease by a factor of 32.

A Some Other Applications

In this Appendix, I will address some applications of TRAK which may require extensive modifications to the code as it currently exists. The two applications discussed here are modifying TRAK to follow field lines and implementing TRAK on a VAX/VMS system. In each case I will outline what I think must be done to the code; there may be other complications which I have not foreseen, so let the user beware!

A.1 Field Line Following

Modifying TRAK to follow magnetic field lines is really very easy. All that needs to

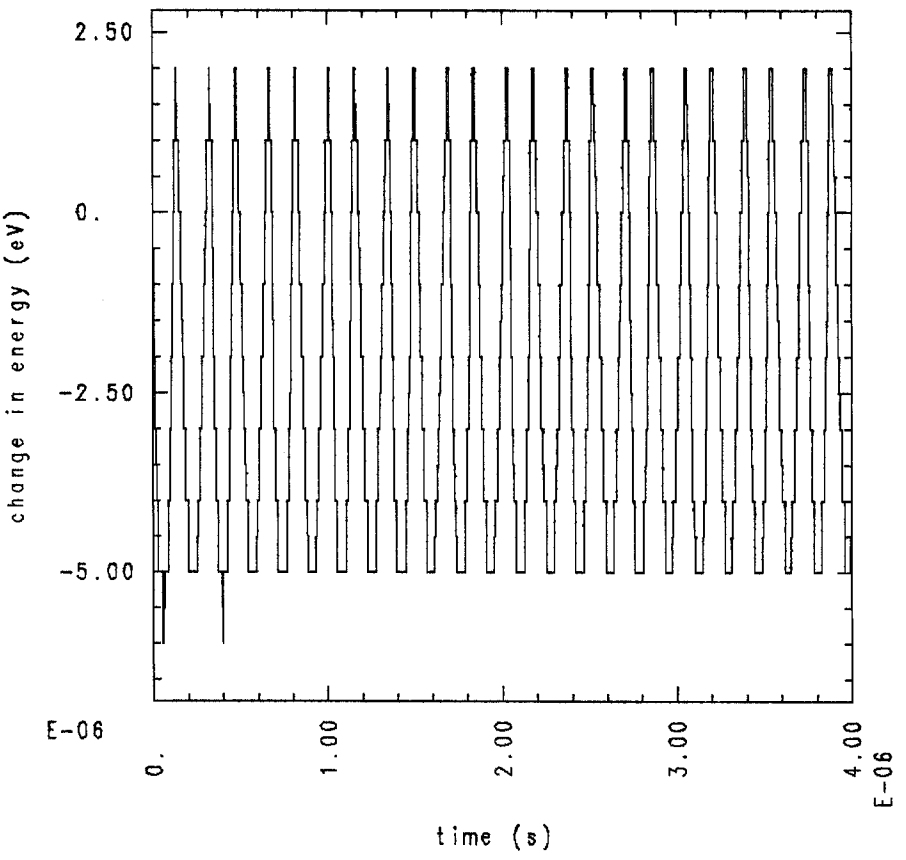

Figure 1: Change in Energy *vs.* time; $\Delta t = 1 \times 10^{-10}$ s

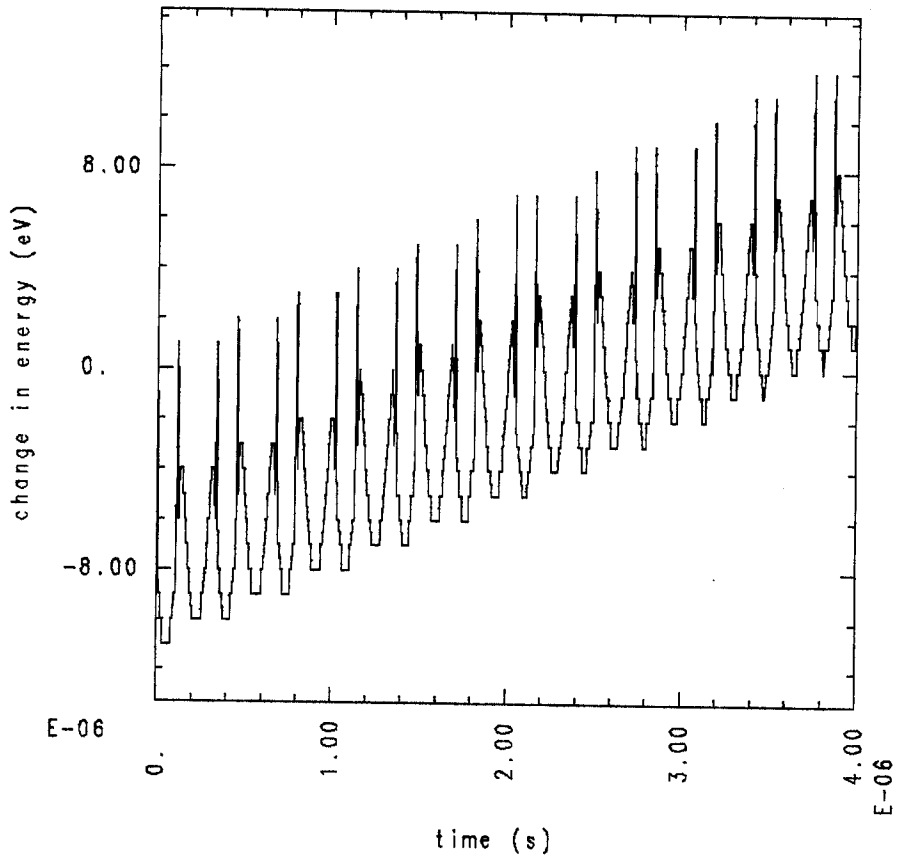

Figure 2: Change in Energy *vs.* time; $\Delta t = 2 \times 10^{-10}$ s

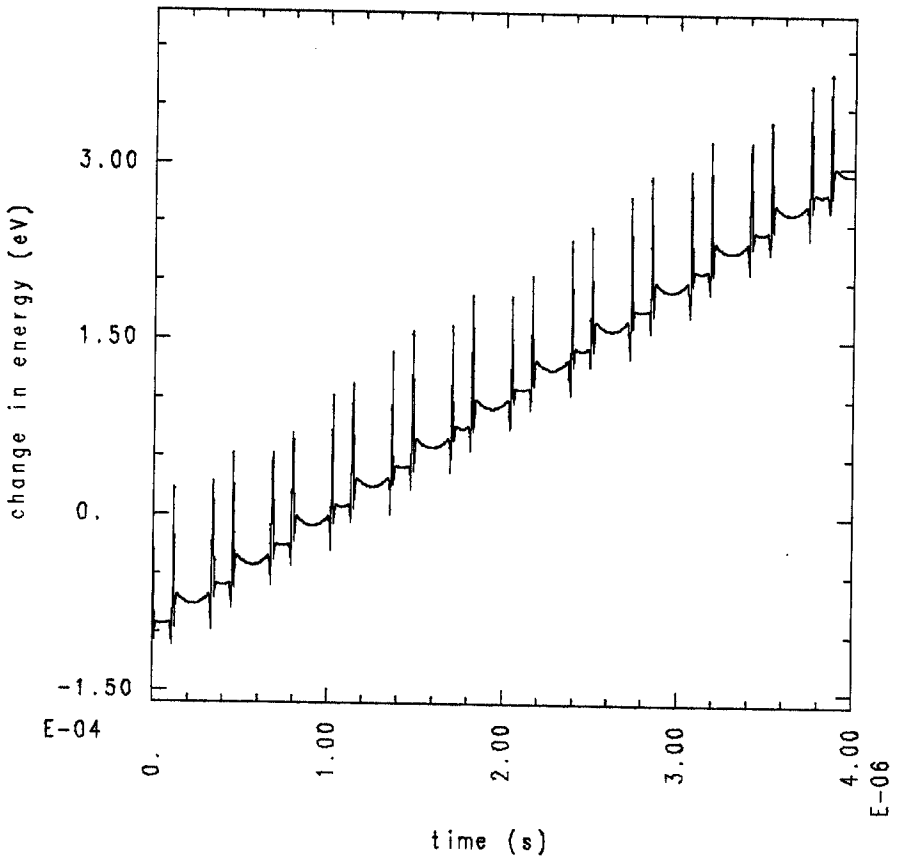

Figure 3: Change in Energy *vs.* time; $\Delta t = 4 \times 10^{-10}$ s

be done is to substitute the equations of "motion" of a field line for the guiding center equations used in TRAK.

The "motion" of a field line in cylindrical coordinates is described by the following set of equations:

$$\frac{dr}{B_r} = \frac{r\,d\phi}{B_\phi} = \frac{dz}{B_z} = \frac{dl}{B} \tag{25}$$

where l is the distance along the the field line and B is $|\underline{B}|$. l is the independent variable, and the three spatial coordinates are the dependent variables. The quantities $\frac{dr}{dl}$, $\frac{d\phi}{dl}$ and $\frac{dz}{dl}$ are analogous to the guiding center velocities in TRAK.

Now, simply substitute the expressions for the "velocity" of the field line into the subroutine DERIV. TRAK has become a field line following code. Because the expressions for the "velocity" are so simple, it is more efficient to have STEP call FIELD directly and eliminate the call to DERIV.

A.2 Porting TRAK to a VAX/VMS System

Getting TRAK to run under VMS should be a fairly simple procedure, although to my knowledge neither TRAK nor its ancestors have ever been run on a VAX. TRAK was written using FORTLIB, which allows some of the constructs that are found in Fortran 77, so there should be no problems with the bulk of the code, although the user is cautioned about the use of character strings (which will have to be declared as such on a VAX) and the lower machine precision on a VAX (although double precision could be used if needed). Thus most of the changes needed will occur at the "top level" rather than in the subroutine STEP and its subordinates.

First, all COSMOS command statements must be deleted. They won't work and it is a lot easier to write command procedures in VMS anyway. Next, all FORTLIB subroutine calls must either be modified or deleted. Most of the math subroutines will not need to be changed, but things like the call to `dropfile` will have to be deleted. Elimination of these subroutine calls means that TTYOUT will no longer work as written. Either TTYOUT must be deleted from the code or a new asynchronous trap (AST) routine must be inserted. AST routines are not terribly hard to write in VAXFortran.

Another important FORTLIB subroutine which will have to be replaced is `link`. Opening files for input and creating them for output can be done with the `open` statement.

The last major change involves the use of the namelist statement. The chances of

the FORTLIB and VAXFortran implementations of `namelist` being compatible are pretty small, so the user is cautioned that this may present some difficulty.

B Using PLOTS

In this Appendix, the use of the PLOTS code to display data from the output file generated by TRAK will be outlined. The format of the output file has been described in Section 2.5; by taking advantage of this format, PLOTS can plot any two distinct quantities in the output file. The purpose of this Appendix is to allow the user to run PLOTS; no attempt will be made to explain how the code works. The first version of PLOTS was written nearly three years ago; the code has grown in unforeseen ways and is not very well commented.

There are two different kinds of plots which can be generated using PLOTS; a histogram plot of one quantity and a plot of one quantity as a function of another. The histogram option is not terribly useful unless one is debugging a scattering routine. The data plots can also be divided into two types: "global" and "local". The global plot chooses one data point from every data block in the data file; a local plot chooses data points from a single data block.

PLOTS may be retrieved from the mass storage facility in the same manner as TRAK. PLOTS can be used on either the CDC 7600 or Cray-1 computers at NMFECC; the file `plots77` (the current version) should be read from FILEM from one of the Cray computers. To compile the code, merely type:

```
cosmos i=plots77 / t v
```

and the code will be compiled. The executable file will be named `xplots`. To use the code, do

```
xplots i=datafile / t v
```

where `datafile` is the name of the file containing the data to be plotted.

The first thing that PLOTS will ask the user for is a type of plot. There are 5 options (besides quitting): a local data plot, a projection plot (a special case of local data plotting), a global data plot, "data vs. time" (another special case of local data plotting) and a histogram. First, let's discuss the histogram option. If this is chosen, the user will be prompted for the label and then the pointer to the desired data entry. Next the low and high data values will be read in, along with the number of bins. Control passes to subroutine

`hplot`. Here the user is asked for an end criterion, which may be a number of points or the value of time (or length along a field line). It should be noted that the histogram option falls into the category of "local" rather than "global" plots. PLOTS will read the data file to generate the data to be plotted. The user will be asked if any statistical analysis of the data is to be done. Next the user will be asked if the value of `scale` should be changed. `scale = 1.0` means that the top of the frame corresponds to the maximum plotted value, `scale = 2.0` means that the top of the frame is twice the maximum value, etc. Now the plot is drawn, and control passes back to the main program.

One important point about the histogram option concerns the end criterion. Similar queries will be made for the other plotting options. If the end of the data block is reached or if the number of points read in exceeds the allowed size of the data structure, no more points are read in and the user is informed. One way to plot all of the points in a data block is to specify a length or time much greater than will be found in any data block. Because this is a very common usage, the end criterion "todo", which reads the whole data block, has been added.

A further note: use of the time/length end criterion depends on the time (or length if the data is for a field line) appearing in the fourth slot of the record with the label "x ". This is also important for the projection plot option.

Now let us consider the local data plotting case; the two special cases of projection plots and "data vs. time/length" can be dealt with at the same time. To do a local plot, we need the label and pointer to the horizontal data entry and another label and pointer to the vertical entry. For the generic local plot, the user is asked to specify the horizontal and vertical labels and then the pointers. For the "data vs. time/length" plots, only the vertical label and pointer are needed because the horizontal data is assumed to be in the fourth slot of the "x " record. If projection plots are desired, the user will be asked to decide between "top" and "phi" projections. The first yields a projection in the (r, ϕ) plane; the second, a projection in the (r, z) plane.

Control now passes to subroutine SPLOT which begins by asking for the end criterion. Next the number of points to be skipped by the data reading algorithm must be specified. If a projection plot has not been asked for, the user may be asked if the angular modulus check should still be applied to the second slot in the "x " record. PLOTS will only inquire about this if this data entry is to be used in the plot.

Next, the user will be asked if a constant is to be subtracted from the horizontal and/or

vertical data. Then the question of the inverse of the data entries is addressed. Finally the data is read in from the data file. This section of the code is extremely opaque; the user who desires to modify PLOTS is urged to be cautious.

Now some data processing is done before handing the horizontal and vertical data off to the plotting subroutine. If the "top" projection mode has been specified, the data is converted to Cartesian coordinates from cylindrical. If projection plots have been specified, the user is asked whether to "close" the plot by drawing the segment between the first and last data point. Next, the user is prompted for the value of the parameter `ipass`. `ipass` = 0 has no effect, `ipass` = 3 means that you will always be asked to specify the plot limits, the other values determine when the plot parameters are to be set. Before handing control to the plotting subroutine GRAFX the subtractions and/or reciprocals are executed.

GRAFX is somewhat easier to follow from the code listing than SPLOT. The first thing that is done is to determine if a call to the "mapping" routine is necessary. For most cases, it will be necessary, so the first guess at the plot limits is the maximum and minimum values of the horizontal and vertical data arrays. The user then has the chance to modify these limits by specifying values or specifying a scaling parameter. Next the user will be asked for data concerning the plot itself; the limits of the picture plane, the number of tic marks, should "nice" numbers be used on the axes, etc. The axis labels are also specified at this time. These are contained in two arrays, `labelx` and `labely`. The first array location should contain the number of characters in the label, the second should be the label as a string in double quotes. Note that this is a namelist read and should conclude with a $.

The subroutine PBRMPS is used to draw the axes and labels. The TV80LIB routine `trace` is used to draw the curve. Control is returned to the main program.

The global data plotting option is very similar to the local data plotting routine. The horizontal and vertical data entries are specified by the user in the same manner. Instead of the call to SPLOT, a call to TPLOT is made. Because of the global nature of the plot, the only end criterion is a number of points, for which the user is prompted. The algorithm walks through the data file taking the first data entries which meet the specifications from each data block. When the end criterion has been met or the end of the file has been reached, GRAFX is called. The plotting is done as indicated above. Control is returned to the main program.

When the main program resumes, first a check is made to see if "multiple rip mode" is active. If so, the data file is positioned at the start of the next data block, `frame` is called

and another plot is done. Otherwise, the user is asked whether to call **frame**. The user is then asked to decide what to do next. The simplest thing to do is to plot new data; the user is returned to the choice between the 5 plotting options and the whole process repeats. A single rip mode freezes some of the options, like **ipass** and the subtractions/reciprocals and does another plot in the same plotting option. **ipass** is important here because it will determine if the same plot limits are to be used. Other possibilities are to rewind the whole data file, rewind to the start of the current data block, call **frame** or open a new data file. The last choice is to enter "multiple rip mode" which works much like the single rip mode except that the user specifies the number of repetitions and **frame** will be called between each data block.

After the user makes a choice, the file is positioned at the start of the data block and the process repeats.

Because of the interactive nature of PLOTS it is extremely likely that some input mistakes will be made. PLOTS can catch some of the errors and ask for the correct information, but this is not foolproof. When the user desires to quit, it is best to wait until quitting is given as an option. This ensures a normal termination so that whatever has been plotted is not lost or munged.

C Data Analysis

I would like to conclude this report with a few words on data analysis. One of the design criteria of TRAK was to keep the code simple and efficient. Thus TRAK itself contains no plotting subroutines and only rudimentary data analysis features. TRAK simply integrates the equations of motion and generates an output file, which contains whatever information the user desires. In the previous Appendix, use of the code PLOTS was described. In many ways PLOTS is only a debugging tool; it allows the user to see the data generated by TRAK in a convenient form. If TRAK is to be used in a transport calculation, for example, software will have to be written which can compute the desired quantities from the output file. An example of such a code is GOBBLE, which was written for use with the particle following code TRACK — a precursor to TRAK.

GOBBLE is not a complete data analysis code, though it contains many of the features of such a code. GOBBLE contains a simple section of code which reads the output file and saves some of the "global" data as well as the initial and final particle positions. From

this data, some calculations are made, and the data are saved in another file. The code is much less complex than PLOTS because it is small, so the need to recompile it in response to changes in the output file structure is not a major consideration. Note that the file generated by GOBBLE is in the same format as the output file. This allows the use of PLOTS if desired.

Essentially, GOBBLE was the "first pass" of the data analysis process. Its principal function was to compress the output file from the code TRACK for ease of transmission over the ARPAnet. The rest of the data analysis process was done by codes written in the computer language Lisp.

Acknowledgment

Support for this work has been provided by the U.S. Department of Energy.

Bibliography

[1] T. G. Northrop, *The Adiabatic Motion of Charged Particles*, John Wiley and Sons, New York, 1963.

[2] J. W. Johnson, L. M. Lidsky, K. Molvig and K. Hizanidis, "Monte Carlo Calculation of the Pinch and Bootstrap Neoclassical Transport Coefficients", MIT Plasma Fusion Center Report PFC/RR-83-34, December, 1983.

[3] G. Dahlquist, Å. Björck, and N. Anderson, *Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, 1974.

[4] Forman S. Acton, *Numerical Methods That Work*, Harper & Row, New York, 1970.

[5] J. D. Jackson, *Classical Electrodynamics*, John Wiley and Sons, New York, 1975.

[6] J. G. Aspinall, "Spline Techniques for Magnetic Fields", MIT Plasma Fusion Center Report PFC/RR-83-27, June, 1984.

[7] Carl de Boor, *A Practical Guide to Splines*, Springer–Verlag, New York, 1978.

[8] A. H. Boozer and G. Kuo-Petravic, *Phys. Fluids*, **24**, 851, (1981).

[9] G. E. P. Box and M. E. Muller, *Ann. Math. Stat.*, **29**, 610, (1958).

[10] R. Shanny, J. M. Dawson and J. M. Greene, *Phys. Fluids*, **10**, 1281, (1967).

[11] G. G. Lister, D. E. Post and R. Goldston, *3rd Varenna Symposium on Plasma Heating in Toroidal Devices*, Editrice Compositori, Bologna, Italy, 1976.