ION - A Code to Compute Ion Trajectories in Z-Pinch Plasma Channels

G.A. Moses

June 1986

UWFDM-712

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

ION - A Code to Compute Ion Trajectories in Z-Pinch Plasma Channels

G.A. Moses

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

http://fti.neep.wisc.edu

June 1986

UWFDM-712

ION - A CODE TO COMPUTE ION TRAJECTORIES IN Z-PINCH PLASMA CHANNELS

Gregory A. Moses

Fusion Technology Institute
1500 Johnson Drive
University of Wisconsin-Madison
Madison, Wisconsin 53706

January 1986

UWFDM-712

1. Introduction

Light ion beam driven inertial confinement fusion (ICF) requires that megaamperes of current in the form of light ions (protons through carbon) be focused onto a fusion target about 1 cm in diameter.[1] Ballistic focusing of such large particle currents is only possible over relatively short distances (~ 20-50 cm) due to space charge forces. Although this ion-diode ballistic focusing is the basis of today's light ion fusion experiments it must be replaced by ion propagation over standoff distances of several meters for the high yield experiments and reactors of the future.

One proposed scheme for propagating ions is depicted in Fig. 1. The ions are focused from an extraction diode over a short distance into the entrance of a preformed z-pinch plasma channel.[2] A number of these channels (~ 8-20) terminate at the target in the center of the chamber. Ions from an individual diode propagate down the plasma channel, confined by the B_θ azimuthal magnetic field in the z-pinch plasma. The plasma has high enough conductivity to charge and current neutralize the ion beam. Understanding the formation of these plasma channels and the propagation dynamics of the ion beam in this background magnetized plasma is vital to the success of light ion driven fusion reactor applications.

The complete analysis of this ion propagation demands the solution of Maxwell's equations along with models of the ion beam itself and the background plasma. The delicate interaction of these three components can lead to macroscopic instabilities such as the sausage and hose instabilities and to microscopic instabilities such as the two stream instability or to beam filamentation.[3,4] Theoretical analysis of these instability issues has led to a set of instability threshold criteria[5] that have been implemented in a code

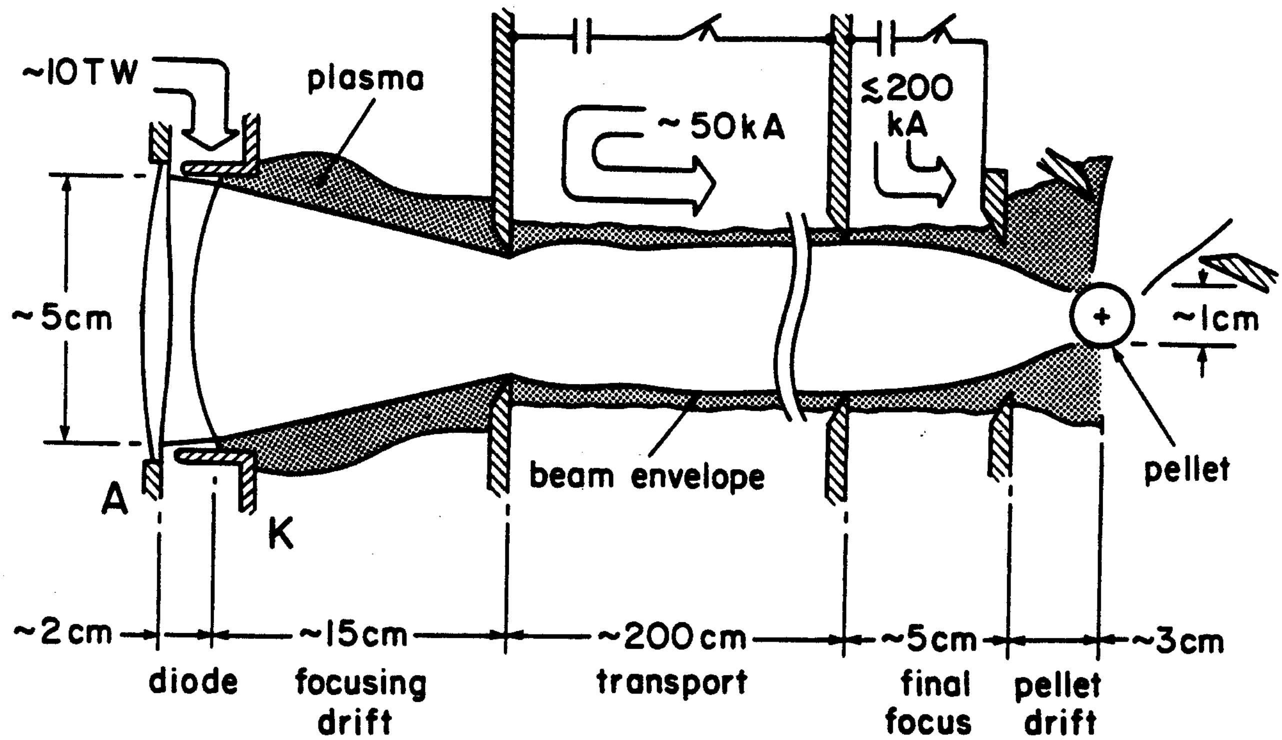

Fig. 1. Z-Pinch plasma channel.

called WINDOW (FENSTER).[6] This code determines the "window" in parameter space where beams are thought to propagate, free of instabilities that might destroy the beam.

The ION code, described in this report, determines the trajectories of individual ions in a plasma channel assuming that the ideal conditions of perfect space charge and current neutralization are in effect. The only forces on the ions are the magnetic field due to the current driven through the channel plasma by an external circuit. By stochastically injecting ions into the channel according to some radial and angular distribution the ION code can graphically determine the envelope of the beam made up of these ions. This will allow the determination of the transport efficiency for varying diode focusing parameters and for different magnetic field configurations in the channel. The code is written in a modular fashion in standard FORTRAN 77 and can be easily modified to add new effects or to measure different beam parameters.

Section 2 of this report outlines the theoretical bases for the methods used in the code. Section 3 describes the features in the code. In the Appendix is a listing of the ION code and a description of two IMSL library routines used by the code.

2. Method of Solution

There are three different ion trajectory models implemented in the ION code. The first two are analytic in nature and are taken from the paper by Ottinger, Mosher, and Goldstein, "Propagation of Intense Ion Beams in Straight and Tapered Z-Discharge Plasma Channels," Ref. 7. The third model uses a numerical integration of the equations of motion of the ion in an arbitrary magnetic field allowing nonzero angular momentum of the ions. Each of these will be briefly discussed in the following three subsections.

2.1 Axially Uniform Plasma Channel

An axially uniform plasma channel (confusingly called a "straight" channel in Ref. 7) with a radially uniform current density results in an azimuthal magnetic field of the form

$$B_\theta = B_o\, r/r_c \qquad r < r_c$$
$$B_\theta = B_o\, r_c/r \qquad r > r_c$$

where r_c is the radius of the uniform current profile and B_o is the magnetic field at this radius. The equations of motion for an ion of mass m_i and charge Q in this linear magnetic field are

$$\ddot{r} = -\omega_{cb}\dot{z}\; r/r_c$$
$$\ddot{z} = \omega_{cb}\dot{r}\; r/r_c$$

where the beam cyclotron frequency is

$$\omega_{cb} = QeB_o/m_i c \; .$$

It is assumed that there is no angular momentum component to the ion motion. Approximate solutions to these equations can be obtained for the condition $\dot{r}/\dot{z} \ll 1$ (i.e., the ions are injected at a shallow angle of injection). The initial conditions for these equations of motion are shown in Fig. 2. The approximate analytic solutions are[7]

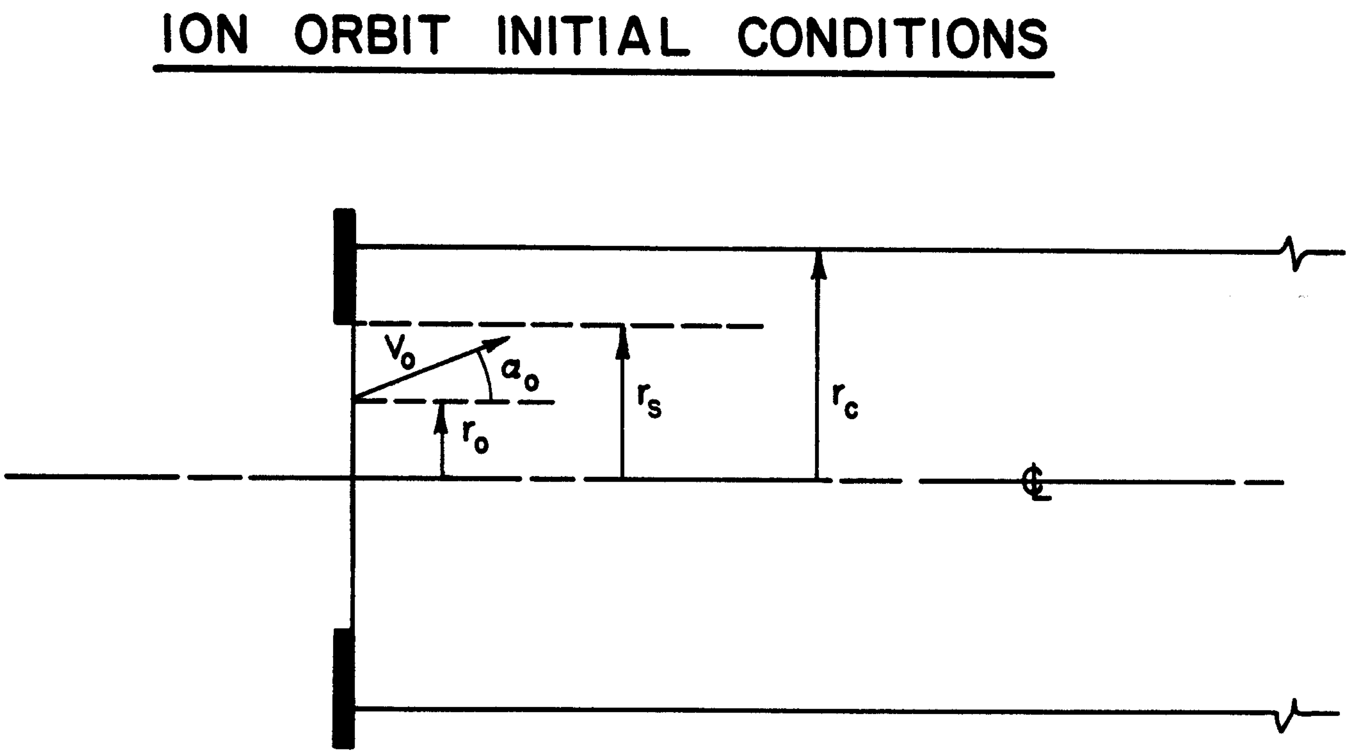

Fig. 2. Initial conditions for ion injection into a Z-Pinch plasma channel with no angular momentum.

$$z(t) = (V_o \cos \alpha_o - \frac{\omega_{cb}\bar{r}^2}{4r_c} \cos 2\phi)t$$

$$+ \frac{\bar{r}^2}{8r_c} (\frac{r_c \omega_b}{V_o \cos \alpha_o})^{1/2} [\sin 2(\omega_\beta t + \phi) - \sin 2\phi] + O(\epsilon^3)$$

$$r(t) = \bar{r} \cos (\omega_\beta t + \phi) + O(\epsilon^3)$$

where

$$\omega_\beta = \Omega(1 - \frac{\omega_{cb}\bar{r}^2}{16\, r_c V_o \cos \alpha_o} + \frac{\tan^2 \alpha_o}{4})$$

$$\tan \phi = -(\frac{r_c V_o \cos \alpha_o}{\omega_{cb} \cos \alpha_o})^{1/2} \tan \alpha_o$$

$$\bar{r} = (r_o^2 + \frac{r_c V_o \sin^2 \alpha_o}{\omega_{cb} \cos \alpha_o})^{1/2}$$

$\Omega = (\omega_{cb} V_o \cos \alpha_o / r_c)^{1/2}$ = betatron frequency

V_o = constant speed of ion

α_o = angle of injection into the channel

$r(0) = r_o$ $\qquad \dot{r}(0) = V_o \sin \alpha_o$

$z(0) = 0$ $\qquad \dot{z}(0)\ V_o \cos \alpha_o$.

These conditions demand that the channel current satisfy[7]

$$I_o > (1.57 \times 10^7 \frac{m_i/m_p}{Q} \alpha_m^2 V_o/c)(1 - r_s^2/r_c^2)^{-1} \text{ amps}$$

to confine ions with a maximum injection angle of α_m, where m_p is the proton mass.

2.2 Tapered Plasma Channel

If the channel radius is assumed to decrease linearly along its length such that

$$r_c(z) = r_c(1 - z/L) ,$$

then $$B_\theta = \frac{2I_o r}{c r_c^2 (1 - z/L)^2} \qquad r < r_c(1 - z/L)$$

where L is the taper length, r_c is the channel radius at z = 0 and this expression only applies for

$$(L - z)^2 \gg r^2 .$$

In this case the equations of motion for an ion take the form

$$\ddot{r} = -\frac{\omega_{cb}\dot{z}r}{r_c(1 - z/L)^2}$$

$$\ddot{z} = \frac{\omega_{cb}\dot{r}r}{r_c(1 - z/L)^2}$$

where $$\omega_{cb} = \frac{2\,QeI_o}{m_i c^2 r_c} .$$

Again, for $\dot{r}/\dot{z} \ll 1$, these equations may be solved approximately to give the ion orbit as(7)

$$z(t) = V_o t + O(\varepsilon^2)$$

$$r(t) = r_o(1 - t/T)^{1/2} \cos[-\Omega T \ln (1 - t/T)]$$

$$+ (V_o \sin \frac{\alpha_o}{\Omega})(1 - t/T)^{1/2} \sin[-\Omega T \ln (1 - t/T)] + O(\varepsilon^2)$$

where $T = L/V_o$

$$\Omega^2 = \omega_{cb} V_o / r_c$$

$$r(0) = r_o \qquad \dot{r}(0) = V_o \alpha_o$$

$$\dot{z}(0) = V_o$$

$$\alpha_o << 1 \quad \text{and} \quad (\Omega T)^{-1} << 1 .$$

2.3 Arbitrary $B_\theta(r,z)$ and Nonzero Angular Momentum

Rather than using either of the previous approximate analytic solutions to the ion orbit equations one can simply solve the differential equations that describe the ion motion. This is done for the general case of ions with nonzero angular momentum. This is shown schematically in Fig. 3. The previous solutions (with zero angular momentum) are solved for the "in-plane" motion of the ions because there was no component of velocity out of the r-z plane. For the purposes of simulation the r-z plane "in-plane" can always be chosen as the x-z plane. Then V_y will represent the out-of-plane velocity component.

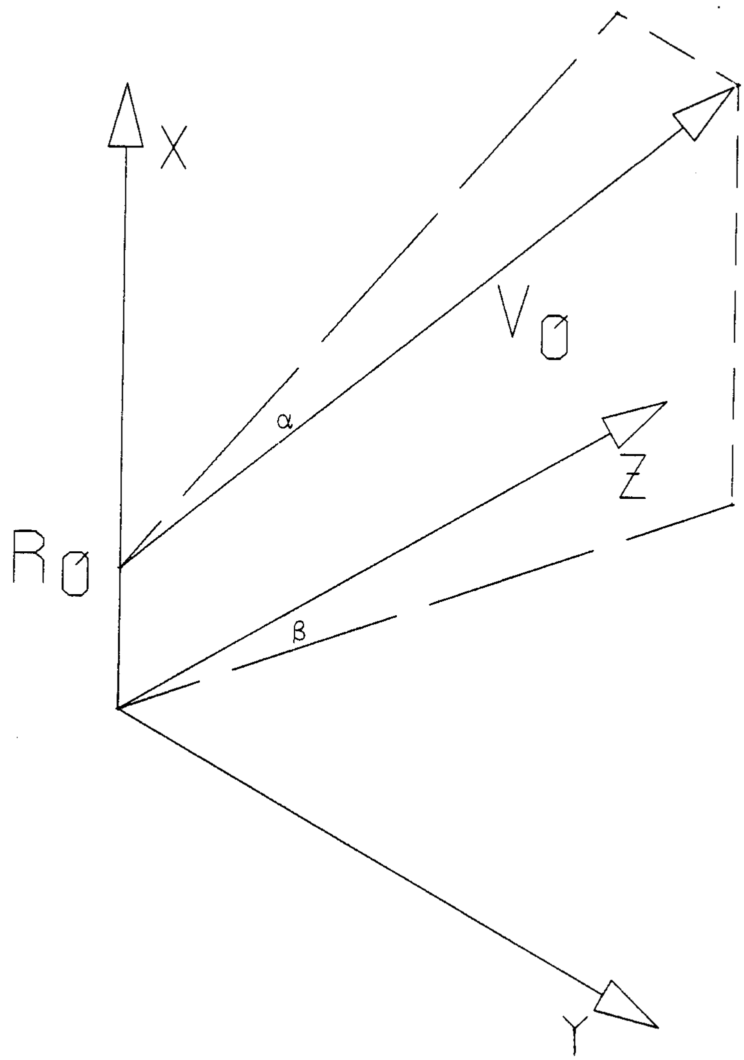

Fig. 3. Initial conditions for ion injection into a Z-Pinch plasma channel including angular momentum.

The equations of motion are

$$\ddot{x} = \frac{Qe}{mc}\left(\dot{y}B_z - \dot{z}B_y\right)$$

$$\ddot{y} = \frac{Qe}{mc}\left(\dot{x}B_z - \dot{z}B_x\right)$$

$$\ddot{z} = \frac{Qe}{mc}\left(\dot{x}B_y - \dot{y}B_x\right)$$

with

$x(0) = r_o$ $\qquad$ $\dot{x}(0) = V_o \sin \alpha_o$

$y(0) = 0$ $\qquad$ $\dot{y}(0) = V_o \cos \beta_o$

$z(0) = 0$ $\qquad$ $\dot{z}(0) = V_o \cos \alpha_o$

$r_o = f_{r_o}(x/r_s)$ $\qquad$ Gaussian distribution

$\alpha_o = f_{\alpha_m}$ $\qquad$ uniform distribution $[-\alpha_m, \alpha_m]$

$\beta_o = f_{\beta_m}$ $\qquad$ uniform distribution $[-\beta_m, \beta_m]$

These equations are posed as six first order O.D.E.'s

$\dot{x}_1 = x_2$ $\qquad$ $\dot{x}_2 = \frac{Qe}{mc}(x_4B_z - x_6B_y)$

$$\dot{x}_3 = x_4 \qquad \dot{x}_4 = \frac{Qe}{mc}(x_2 B_z - x_6 B_x)$$

$$\dot{x}_5 = x_6 \qquad \dot{x}_6 = \frac{Qe}{mc}(x_2 B_y - x_4 B_x)$$

and solved using a standard fifth order predictor-corrector solver. For $\beta_m = 0$ these equations default to the zero angular momentum equations. The subroutine BXYZ can be modified by the user to give arbitrary magnetic fields. For

$$B_\theta(r) = B_o(r/r_c)^n$$

the expressions for B_x and B_y are given below

$$B_x/B_\theta = \sin\theta \qquad B_x = B_\theta \sin\theta$$

$$B_y/B_\theta = \cos\theta \qquad B_y = B_\theta \cos\theta$$

$$B_x = B_\theta y/r \qquad r = \sqrt{x^2 + y^2}$$

$$B_y = B_\theta x/r \qquad B_\theta = B_o\left(\frac{x^2 + y^2}{x_c^2}\right)^{n/2}$$

hence

$$B_x = B_o\left(\frac{x^2 + y^2}{x_c^2}\right)^{n/2} \frac{y}{\sqrt{x^2 + y^2}} = B_o(x^2 + y^2)^{(n-1)/2} \frac{y}{x_c^n}$$

$$B_y = B_o \left(\frac{x^2 + y^2}{x_c^2}\right)^{n/2} \frac{x}{(x^2 + y^2)^{1/2}} = B_o \frac{(x^2 + y^2)^{(n-1)/2}}{x_c^n} x .$$

For the case of uniform current (hence linear B_θ) we get

$$B_x = B_o\, y/x_c ,$$

$$B_y = B_o\, x/x_c .$$

This is the default in the program.

3. The ION Code

The ION code is structured according to the three different ion orbit models discussed in Section 2. A structure chart of the code is given in Fig. 4. Each different model is completely self-contained within the code. New models can be easily added by changing the main program and adding a new subroutine. Variable names conform to this report and Ref. 7. The code calls two IMSL library routines, for random numbers (GGUBFS) and the differential equation solver (DGEAR). The code is designed to be run interactively with free format input for which the user is prompted, but can be run in batch mode as well. Output is mostly to files that may be post-processed for plotting. The format of the files is given in Table 1.

3.1 Features

There are three different output options for each of the three orbit models, thus providing nine different kinds of calculations. These three output options are:

Table 1. Format of Output Files to Use for Plotting

Output Option (ICALC)	Variable	Description	Format
1	NPTS	Number of (z,r) points	I10
	Z(I), R(I) I=1, NPTS	(z,r) points	1P12E10.2
2	NM1	Number of (z,r) points	I10
	Z(I), R(I) I=1, NM1	(z,r) points	1P12E10.2
3	NZFIX	Number of planes at which distributions are computed	I10
	RHIST(J,K) ((J=1,10),K=1,NZFIX)	Radii for bins where particles are tallied.	1P10E10.2
	PHIST(J,K) ((J=1,10),K=1,NZFIX)	Particles tallied in each bin.	1P10E10.2
	FHIST(J,K) ((J=1,10),K=1,NZFIX)	Fluence of particles tallied in each bin.	1P10E10.2

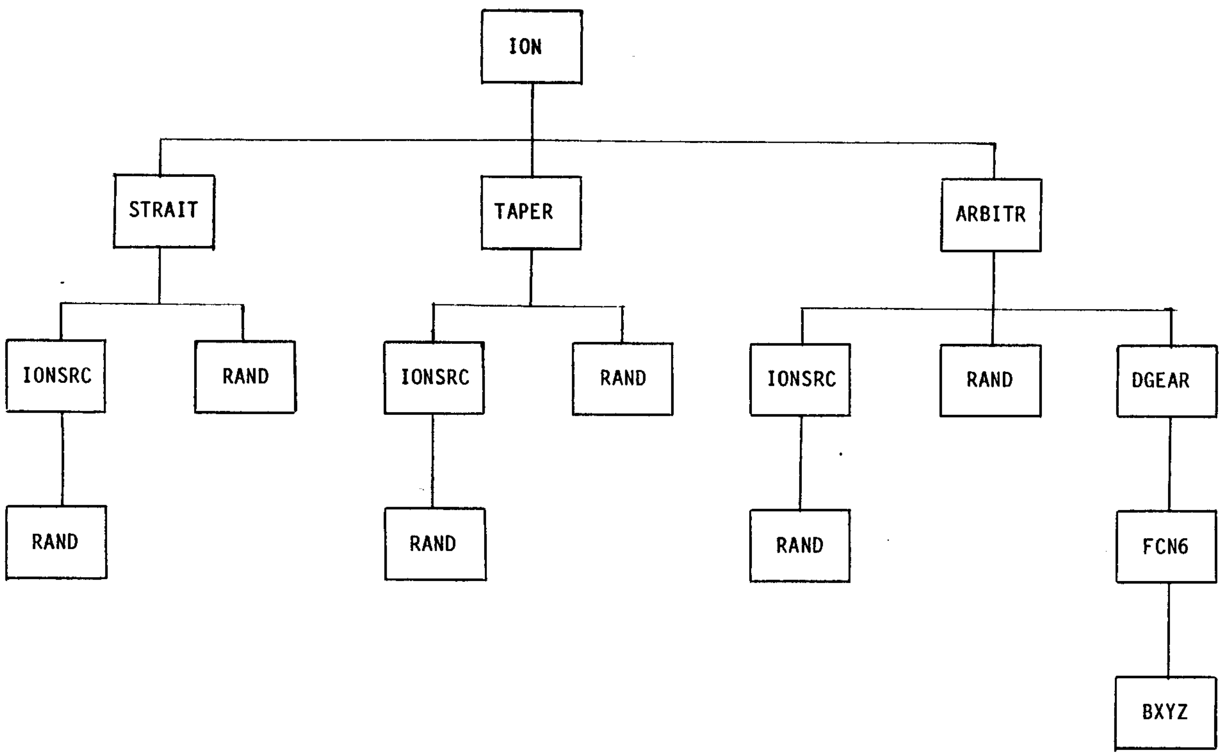

Fig. 4. Structure chart of the ION code.

1. trace the trajectory of ions for a specified length of time;
2. show the positions of all ions at a selected instant in time;
3. plot the radial profile of ions as they pass through a plane at a specified position in the channel.

Other output options can be easily added to each orbit model by including another ELSE IF statement in the appropriate subroutine.

In all cases the ions are randomly injected into the channel in a Gaussian distribution in radius and a uniform distribution in angle. These are shown in Fig. 5. The initial radius r_0 is selected by sampling the distribution

$$r_0 = r_s \sqrt{-2 \ln \xi}$$

and the angle of injection and the out of plane angle are selected by sampling

$$\alpha_0 = \alpha_m (1 - 2\xi)$$

$$\beta_0 = \beta_m (1 - 2\xi)$$

where ξ is a random number uniformly distributed in the interval [0,1]. The code uses pseudo-random numbers since the same initial seed number will produce the same sequence of random numbers.

Acknowledgment

This work was supported in part by contracts from Kernforschungszentrum Karlsruhe and Sandia National Laboratory, Albuquerque.

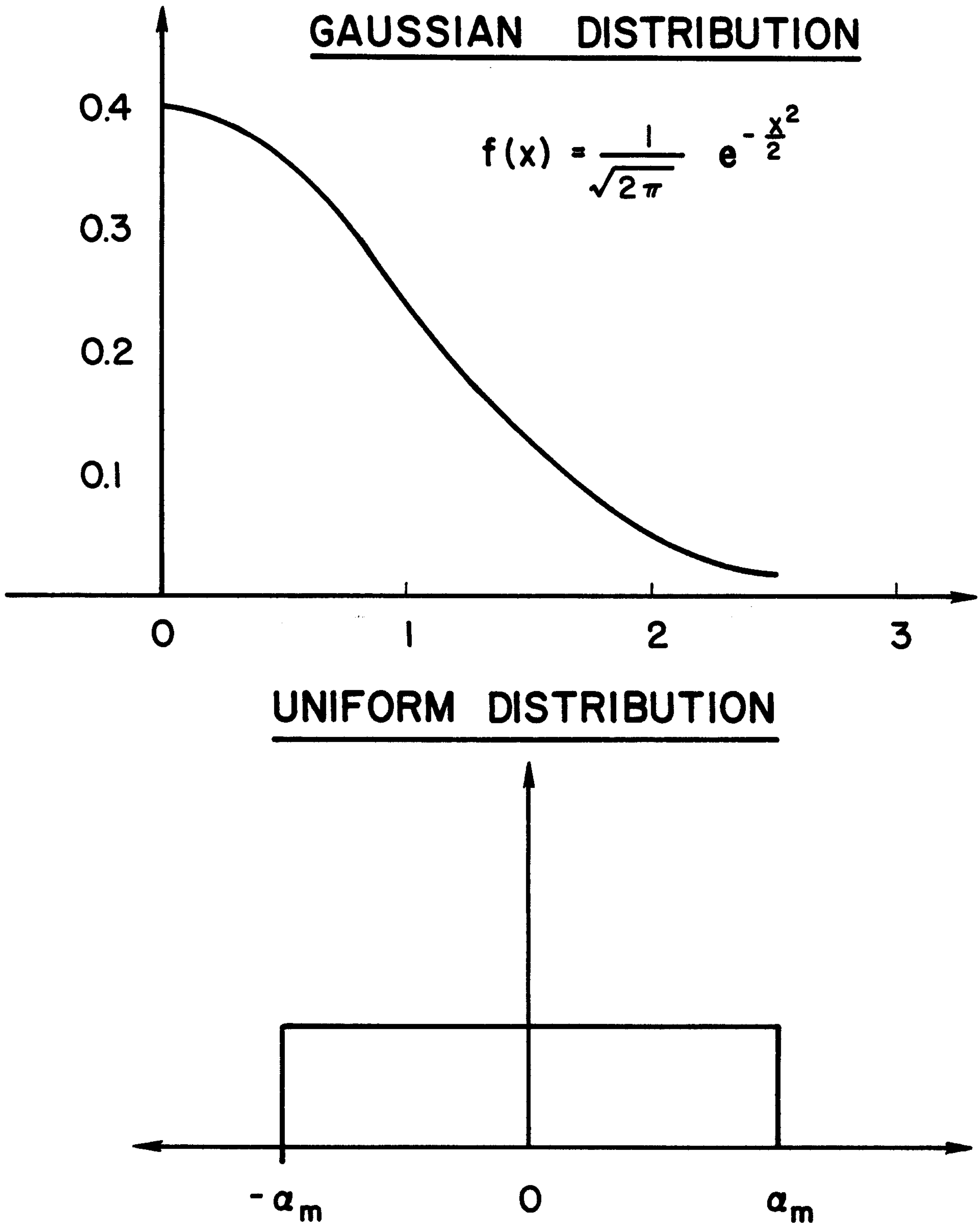

Fig. 5. Radial and angular distributions used for ion injection into z-pinch plasma channel.

References

1. G. Yonas, in Plasma Physics and Controlled Nuclear Fusion Research 1982 (Proc. 9th Int. Conf. Baltimore, 1982) Vol. 2, IAEA, Vienna (1983), p. 353.

2. J.R. Freeman, L. Baker, and D.L. Cook, "Plasma Channels for Intense Light Ion Beam Reactors," Nucl. Fus. 22, 383 (1982).

3. P.F. Ottinger, D. Mosher, and S.A. Goldstein, "Microstability of a Focused Ion Beam Propagating Through a Z-Pinch Plasma," Phys. Fluids 22, 332 (1979).

4. P.F. Ottinger, D. Mosher, and S.A. Goldstein, "Electromagnetic Instabilities in a Focused Ion Beam Propagating Through a Z-Discharge Plasma," Phys. Fluids 24, 164 (1981).

5. P.F. Ottinger, S.A. Goldstein, and D. Mosher, "Constraints on Transportable Ion Beam Power," NRL Memorandum Report-4948, November 1982.

6. R.R. Peterson, "WINDOW - A Code to Compute Ion Beam Power Constraints," Fusion Power Associates Report FPA-84-6 (1984).

7. P.F. Ottinger, D. Mosher, and S.A. Goldstein, "Propagation of Intense Ion Beams in Straight and Tapered Z-Discharge Plasma Channels," Phys. Fluids 23, 909 (1980).

APPENDIX. ION CODE LISTING AND IMSL ROUTINE DESCRIPTIONS

```
      cliche seed
c  use double precision random for imsl random number generator
c  and integer * 2 random for ibm-pc random number generator
      double precision random
c     integer * 2 random
      common/seed/ random
      endcliche
      cliche all
      common/all/
     1 model,    qion,     xmion,    rchanl,   zchanl,
     2 currnt,   b0,       nion,     rsourc,   alpham,
     3 betam,    e0,       wcb,      beta0,    qimic,
     4 timmax,   ntime,    dedt,     icalc,    tsnap,
     5 v0min,    dvdt,     deltat,   trap,     rchsq,
     5 v0,       r0,       alpha0,   i6,
     6 i5,       i10,      c,        amps,     nzfix,
     7 rfixed,   pi,       atomwt,   zfixed(10),
     7 rhist(10,10),       phist(10,10),       fhist(10,10)
      endcliche
      cliche save
      common/save/
     1 xsave(1000),        vxsave(1000),
     2 ysave(1000),        vysave(1000),
     3 zsave(1000),        vzsave(1000),
     4 r(5000),            z(5000)
      endcliche
      cliche bugs
      logical debug
      common/bugs/ idebug, debug(20)
      endcliche
      cliche solve
      integer * 4 iwk, numeqn, meth, miter, index, ier
      common/solve/
     1 wk(103),  iwk(6),   x,        y(6),     numeqn,
     2 dt0,      xend,     tol,      meth,     miter,
     3 index,    ier
      endcliche
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  this simple program computes ion trajectories in z-discharge plasma
c  channels using the analytic methods of ottinger, mosher, and goldstein
c  in "propagation of intense ion beams in straight and tapered
c  z-discharge plasma channels", physics of fluids 23, 909(1980).
c
c  the program also computes the ion orbits in channels with
c  arbitrary magnetic fields varying with z and r, and for beams with
c  non-zero angular momentum.  it uses a predictor-corrector solver
c  from the imsl library to solve the equations of motion.
c
c  units used in the calculation are cgs and gaussian.  some input
c  quantities are requested in other units, such as amperes, but they
c  are converted in the program before use.
c
c  the user is prompted for only a few input parameters and the output
c  is best viewed in graphical form.
c
c  the program is written in fortran 77 and can be run on an ibm/pc
c  computer using the profort compiler and the corresponding imsl
c  mathematical subroutine library.  the program writes files that can
c  be post-processed by a graphics utility for plotting.
c
```

```
c  the program is written in a highly structured form and completely
c  documented so that it can be easily modified to suit the users needs.
c
c  documentation for the program is provided in:
c
c       g.a. moses, "ion - a code to compute ion trajectories
c       in z-pinch plasma channels", fusion power associates
c       report fpa-84-5 (rev. january 1986).
c
c                             and
c
c       g.a. moses, "ion - a code to compute ion trajectories
c       in z-pinch plasma channels", university of wisconsin
c       fusion technology institute report uwfdm-xxx, jan. 1986.
c
c
c  written by:                        gregory a. moses
c
c               fusion power associates        university of wisconsin
c               6515 grand teton plaza         1500 johnson dr.
c               madison, wisconsin 53719       madison, wisconsin 53706
c               (608)833-3388                  (608)263-3368
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c
      use all
      use seed
      use bugs
c
      character cmodel(3)*12
      character copt(3)*12
      character outfil(10)*8
c
c  set up dropfile for ctss use
      call link("
     1          unit5=(terminal),
     2          unit6=(terminal)//")
c
c  set some input/output device numbers and other constants
      outfil(1) = 'out10   '
      outfil(2) = 'out11   '
      outfil(3) = 'out12   '
      outfil(4) = 'out13   '
      outfil(5) = 'out14   '
      outfil(6) = 'out15   '
      outfil(7) = 'out16   '
      outfil(8) = 'out17   '
      outfil(9) = 'out18   '
      outfil(10)= 'out19   '
      i5 = 5
      i6 = 6
      idebug = 6
      i10 = 9
      do 5 i = 1,20
      debug(i) = .false.
    5 continue
c
c     ******start loop around ion main program here******
    1                      continue
c
```

```
c  each time through the output unit number is incremented by one
      il0 = il0 + 1
      if( il0 .eq. 20 ) then
         write(i6,*) ' max number of output files exceeded'
         stop
      endif
      call create( il0, outfil(il0-9), 2, -1)
c
c  initialize the random number generator seed
      random = 1
c     random = 123456.d0
c
      pi = 4. * atan(1.)
      beta0 = 0.0
      alpha0 = 0.0
      dedt = 0.0
      amps = 0.0
      b0 = 0.0
      c = 2.9979e10
      cmodel(1) = '     uniform'
      cmodel(2) = '     tapered'
      cmodel(3) = '   arbitrary'
      copt(1) = 'trajectories'
      copt(2) = ' phase space'
      copt(3) = 'radial profl'
c
c  input the type of model ( straight  or tapered channel or
c  arbitrary magnetic field with possible non-zero angular momentum)
   10 write(i6,1001)
 1001 format(
     1 '0 what type of channel calculation do you want?'/
     1 '  0 = stop the calculation'/
     2 '  1 = straight channel - analytic solution'/
     3 '  2 = tapered channel  - analytic solution'/
     4 '  3 = arbitrary channel - numerical solution  ')
      read(i5,*) xmodel
      model = xmodel
c
c  is it time to stop all of this?
      if( model .eq. 0 ) then
         write(i6,*) '  all done'
         stop
      endif
c
c  type of calculation - either plot trajectories from an instantaneous
c  ion source or plot the instantaneous positions of ions from a time
c  dependent source or plot histogram of ion radial position
      write(i6,1007)
 1007 format(
     1 '0 choose the type of calculation to be done'/
     2 '  1 = trajectories from an instantaneous source'/
     3 '  2 = positions of ions from a time dependent source'/
     4 '  3 = radial distribution of ions at fixed z positions  ')
      read(i5,*) xicalc
      icalc = xicalc
      if( icalc .eq. 2 ) then
      write(i6,*) ' time at which to plot the ion positions  '
      read(i5,*) tsnap
      endif
      if( icalc .eq. 3 ) then
c
```

```
c  number of z positions to record radial distribution
      write(i6,*)
     1 '  number of z positions to record radial distribution  '
      read(i5,*) nzfix
c
c  z positions themselves
      write(i6,1011)
 1011 format(
     1 '  z positions at which to plot radial distribution(cm)'/)
      read(i5,*) (zfixed(k), k=1,nzfix)
      endif
c
c  initial energy and de / dt
      write(i6,*) ' initial ion energy(mev)  '
      read(i5,*) e0
      if( icalc .eq. 2 ) then
         write(i6,*) ' de / dt(mev/sec) if a ramp is desired  '
         read(i5,*) dedt
      endif
c
c  ion charge and mass
      write(i6,*) ' ion charge state  '
      read(i5,*) qion
      write(i6,*) ' atomic weight(amu)  '
      read(i5,*) atomwt
      xmion = atomwt * 1.6726e-24
c
c  channel radius, length, current or maximum b field
      write(i6,*) ' channel radius(cm)  '
      read(i5,*) rchanl
      write(i6,*)  ' length(cm)  '
      read(i5,*) zchanl
      write(i6,*) ' current(amps)  '
      read(i5,*) amps
      write(i6,*) ' or b-zero(gauss)  '
      read(i5,*) b0
c
c  number of ions, spot radius, maximum divergence angles
      write(i6,*) ' number of ions in the source distribution  '
      read(i5,*) xnion
      nion = xnion
      write(i6,*) ' focal spot radius(cm)  '
      read(i5,*) rsourc
      write(i6,*) ' maximum divergence angle(rad) in plane  '
      read(i5,*) alpham
      if( model .eq. 3 ) then
      write(i6,*) ' maximum divergence angle(rad) out of plane  '
      read(i5,*) betam
      endif
c
c  maximum time and number of times to evaluate
      if( icalc .eq. 1 ) then
      write(i6,1005)
 1005 format(
     1 '0 maximum time to run the problem ',
     2 'and number of times to evaluate'/)
      else if( icalc .eq. 2 ) then
      write(i6,1012)
 1012 format(
     1 '0 maximum time that source is turned on ',
     2 ' and number of times to evaluate'/)
```

```fortran
      else
      endif
c
      if( icalc .eq. 1 .or. icalc .eq. 2 ) then
      read(i5,*) timmax, xntime
      ntime = xntime
      endif
c
c  compute terms used for all models
      v0min = sqrt(2. * 1.6022e-6 * e0 / xmion)
      if( dedt .eq. 0.0 ) then
         emax = e0
         vmax = v0min
         dvdt = 0.0
      else
         emax = e0 + dedt * timmax
         vmax = sqrt( 2. * 1.6022e-6 * emax / xmion )
         dvdt = (vmax - v0min) / timmax
      endif
      if( amps .ne. 0.0 ) then
         currnt = 3.e9 * amps
         b0 = 2. * currnt / (c * rchanl)
      else
         currnt = 0.5 * b0 * c * rchanl
      endif
      wcb = 4.8032e-10 * qion * b0 / (xmion * c)
      wbeta = sqrt(wcb * v0min / rchanl)
      if( ntime .ne. 0 ) deltat = timmax / ntime
      trap = sqrt(currnt * qion * c / (4.71e16 * atomwt * v0min))
      rchsq = rchanl ** 2
c
c  summarize the input
      if( icalc .eq. 1 ) then
      write(i6,1020)
     1 cmodel(model),copt(icalc),qion,atomwt,e0,
     2 v0min,nion,rsourc,alpham
 1020 format(
     1 '0 type of channel....................',a/
     2 '  type of calculation................',a/
     3 '  ion charge.........................',f12.3/
     4 '  ion atomic weight..................',f12.3/
     5 '  ion energy(mev)....................',f12.3/
     6 '  ion velocity(cm/sec)...............',1p1e12.4/
     7 '  number of ion trajectories.........',i12/
     8 '  ion source focal radius(cm)........',e12.4/
     9 '  ion source maximum divergence(rad)..',e12.4)
      write(i6,1021)
     1 rchanl,zchanl,amps,b0,timmax,ntime,deltat
 1021 format(
     1 '  channel radius(cm)..................',0p1f12.3/
     2 '  channel length(cm)..................',f12.3/
     3 '  channel current(amperes)............',1p1e12.4/
     4 '  maximum magnetic field(gauss).......',e12.4/
     5 '  maximum time to plot trajectory(sec)',e12.4/
     6 '  number of times to evaluate traj....',i12/
     7 '  timestep for trajectory calc........',e12.4)
c
      else if( icalc .eq. 2 ) then
      write(i6,1022)
     1 cmodel(model),copt(icalc),qion,atomwt,e0,dedt,
     2 v0min,dvdt,nion,timmax,ntime,deltat,rsourc,alpham,tsnap
```

```
 1022 format(
     1 '0 type of channel....................',a/
     2 '  type of calculation.................',a/
     3 '  ion charge..........................',f12.3/
     4 '  ion atomic weight...................',f12.3/
     5 '  ion energy(mev).....................',f12.3/
     5 '  ion energy ramp slope(mev/sec)......',1p1e12.4/
     6 '  ion velocity(cm/sec)................',1p1e12.4/
     6 '  ion velocity ramp slope(cm/sec**2)..',e12.4/
     7 '  number of ion trajectories..........',i12/
     5 '  ion source pulse time(sec)..........',e12.4/
     6 '  number of times to evaluate source..',i12/
     7 '  timestep for source calc(sec).......',e12.4/
     8 '  ion source focal radius(cm).........',e12.4/
     9 '  ion source maximum divergence(rad)..',e12.4/
     1 '  time to plot ion positions(sec).....',e12.4)
      write(i6,1023)
     1 rchanl,zchanl,amps,b0
 1023 format(
     1 '  channel radius(cm)..................',f12.3/
     2 '  channel length(cm)..................',f12.3/
     3 '  channel current(amperes)............',1p1e12.4/
     4 '  maximum magnetic field(gauss).......',e12.4)
c
      else if( icalc .eq. 3 ) then
      write(i6,1024) (k,zfixed(k), k=1,nzfix)
 1024 format(
     1 ' zfixed(cm)....(',i2,')............', 1p1e12.4)
      write(i6,1025) cmodel(model), copt(icalc),
     1 e0, qion, atomwt, rchanl, zchanl,
     1 amps, b0, nion, rsourc, alpham, betam
 1025 format(
     1 '0 type of channel....................',a/
     2 '  type of calculation.................',a/
     2 ' ion energy(mev).......................', 1p1e12.4/
     3 ' ion charge............................', e12.4/
     4 ' atomic weight(amu)....................', e12.4/
     5 ' channel radius(cm)....................', e12.4/
     6 ' channel length(cm)....................', e12.4/
     7 ' current(amps).........................', e12.4/
     8 ' magnetic field(gauss).................', e12.4/
     9 ' number of ions in source..............', i12/
     z ' focal spot radius(cm).................', e12.4/
     1 ' max. divergence angle in plane(rad)..', e12.4/
     2 ' max. divergence angle out of plane...', e12.4)
      write(i6,1026) xmion, v0min, currnt, wcb, wbeta
 1026 format(' '/
     1 ' ion mass(g)...........................', 1p1e12.4/
     2 ' ion velocity(cm/sec)..................', e12.4/
     3 ' current(statamps).....................', e12.4/
     5 ' wcb(1/sec)............................', e12.4/
     6 ' betatron freq(1/sec)..................', e12.4)
      else
      endif
c
c  choose the channel model
      if( model .eq. 1 ) then
         call strait
      else if( model .eq. 2 ) then
         call taper
      else if( model .eq. 3 ) then
```

```
         call arbitr
      else
      write(i6,*) ' error in choice of model, model =', model
      endif
c
c  loop back to the beginning for another calculation
      goto 1
c
  999 stop
      end
      subroutine fcnj(n,x,y,pd)
      integer n
      real y(n), pd(n,n), x
      return
      end
      subroutine ionsrc(
     1 rsourc,alpham,betam,
     3 x0,alpha0,beta0)
c
c  this subroutine chooses an ion x position, x0, and an angle of
c  injection in the x-z plane, alpha0, and an angle of injection out of
c  the x-z plane, beta0, into the channel by sampling from a gaussian
c  distribution in position and uniform distributions in angle.
c
      use seed
      use bugs
c
c  sample the distribution for the position
      x0 = rsourc * sqrt(-2. * (alog(rand(random) + 1.e-10)))
c
c  select the angles
      alpha0 = alpham * (1. - 2. * rand(random))
      if( betam .ne. 0.0 )
     1      beta0 = betam * (1. - 2. * rand(random))
c
c  debug output
      if( debug(5) ) then
      write(idebug,*) ' ionsrc - rsourc,alpham,betam,x0,alpha0,beta0'
      write(idebug,*) rsourc, alpham, betam, x0, alpha0, beta0
      endif
c
c  all done
      return
      end
      subroutine arbitr
c
c  this subroutine computes ion trajectories or positions using
c  an arbitrary magnetic field profile b(r,z) or b(x,y,z) and
c  ion beams with non-zero angular momentum
c
      use all
      use seed
      use solve
      use save
      use bugs
c
      parameter ( nsmax = 500 )
      external fcn6, fcnj
c
c  set up predictor-corrector solver parameters
      numeqn = 6
```

```
      tol = 5.e-4
      meth = 1
      miter = 0
c
c  get the floating point form of the number of time steps
      fnsmax = nsmax
c
c  first get the numeric constant in the differential equations
      qimic = 4.8032e-10 * qion / (xmion * c)
c
c  set velocity scale factor for no out of plane vdlocity
      sfactr = 1.0
c
c  choose the type of calculation to be done
      if( icalc .eq. 1 ) then
c
c  we are computing the trajectories of ions from an
c  instantaneous source
c
c  compute some terms that do not depend on r0 and alpha0
      v0 = v0min
      lost = 0
c
c  loop over "nion" ions from the source
      do 150 ni = 1,nion
c
c  get the radius and angle of injection of an ion
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  reset the velocity scale factor if there is out of plane velocity
      if( betam .ne. 0.0 )
     1    sfactr = 1. / sqrt(1. + sin(beta0)**2)
c
c  loop over "ntime" times to get r(t) and z(t) positions to plot
      t = 0.0
      z(1) = 0.0
      r(1) = r0
c
c  get the initial conditions for the dgear solver
c  y(1)=x, y(2)=vx, y(3)=y, y(4)=vy, y(5)=z, y(6)=vz, x=time
      y(1) = r0
      y(2) = v0 * sin(alpha0) * sfactr
      y(3) = 0.
      y(4) = v0 * sin(beta0) * sfactr
      y(5) = 0.
      y(6) = v0 * cos(alpha0) * sfactr
      x = 0.
      index = 1
      dt0 = deltat * 0.1
      do 100 nt = 2,ntime+1
      t = t + deltat
      xend = t
      call dgear(
     a numeqn,fcn6,fcnj,x,dt0,y,xend,tol,meth,miter,index,
     b iwk,wk,ier)
      if( ier .gt. 67 ) then
         write(i6,*) ' error in dgear ier=',ier
         stop
      endif
```

```
      if( betam .eq. 0.0 ) then
         r(nt) = y(1)
      else
         r(nt) = sqrt( y(1)**2 + y(3)**2 )
      endif
      z(nt) = y(5)
      if( abs(r(nt)) .gt. rchanl ) then
         npts = nt
         lost = lost + 1
         goto 125
      endif
  100 continue
      npts = ntime + 1
  125 continue
c
c  debug output
      if( debug(4) ) then
      write(idebug,1000) npts,lost,(z(i),r(i), i=1,npts)
 1000 format(2i12,(1p2e12.4))
      endif
c
c  save trajectory for plotting
      write(i10,1100) npts
      write(i10,1101) (z(i),r(i), i=1,npts)
 1100 format(i10)
 1101 format(1p12e10.2)
  150 continue
c
c  write summary of calculation
      frac = float(nion - lost) / float(nion)
      write(i6,1102) nion, lost, frac
 1102 format('0 ions injected....................', i8/
     1       '  ions lost from channel...........', i8/
     2       '  fraction successfully propagated..', 1p1e12.4)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else if( icalc .eq. 2 ) then
c
c  we are computing the z and r positions of ions from a
c  time dependent source
c
c  loop over "ntime" source times
      sfactr = 1.0
      n = 1
      tprime = -deltat
      do 250 ntp = 1,ntime
      tprime = tprime + deltat
c
c  loop over "nion" source ions at this time
      do 225 ni = 1,nion
c
c  get the radius and angle of injection of an ion
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  reset velocity scale factor if there is out of plane velocity
      if( betam .ne. 0.0 )
     1    sfactr = 1. / sqrt( 1. + sin(beta0)**2 )
c
```

```
c  distribute the injection time uniformly from tprime to
c  tprime + deltat
      tpr = tprime + rand(random) * deltat
c
c  get the ion velocity -- it may be ramped
      v0 = v0min + dvdt * tpr
c
c  compute the position
c
c  get the initial conditions for the dgear solver
c  y(1)=x, y(2)=vx, y(3)=y, y(4)=vy, y(5)=z, y(6)=vz, x=time
      x = 0.0
      y(1) = r0
      y(2) = v0 * sin(alpha0) * sfactr
      y(3) = 0.
      y(4) = v0 * sin(beta0) * sfactr
      y(5) = 0.
      y(6) = v0 * cos(alpha0) * sfactr
c
c  integrate the equations of motion up to the desired time
      xend = tsnap - tpr
      dt0 = xend * 2.e-4
      index = 1
      call dgear(
     a numeqn,fcn6,fcnj,x,dt0,y,xend,tol,meth,miter,index,
     b iwk,wk,ier)
      if( ier .gt. 67 ) then
         write(i6,*) ' error in dgear ier=',ier
         stop
      endif
  200 continue
      if( betam .eq. 0.0 ) then
         r(n) = y(1)
      else
         r(n) = sqrt( y(1)**2 + y(3)**2 )
      endif
      z(n) = y(5)
      n = n + 1
  225 continue
  250 continue
c
c  debug output
      if( debug(4) ) then
      write(idebug,1000) (z(i),r(i), i=1,n-1)
      endif
c
c  save the positions for plotting
      nml = n - 1
      write(i10,1200) nml
      write(i10,1201) (z(i), r(i), i=1,nml)
 1200 format(i10)
 1201 format(1p12e10.2)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else if( icalc .eq. 3 ) then
c
c  we are computing a histogram of ion radial position at fixed
c  z positions.
c
c  get terms that do not depend on initial conditions
```

```
      v0 = v0min
      sfactr = 1.0
      do 400 k = 1,nzfix
      do 400 j = 1,10
      rhist(j,k) = rchanl * (j / 10.)
 400  continue
c
c  loop over "nion" ions from the source
      do 450 n = 1,nion
c
c  get the initial positions and angles of injection of an ion
c  it is assumed to be injected in the x-z plane so y0 = 0,
c  but it can have an out of plane velocity vy at an angle beta0
      call ionsrc(
     1 rsourc, alpham, betam,
     3 r0, alpha0, beta0)
c
c  get the scale factor so that the energy of every ion is the
c  same when there is an out of plane component of the velocity
      if( betam .ne. 0.0 )
     1    sfactr = 1. / sqrt(1. + sin(beta0)**2)
c
c  get the initial conditions for the dgear solver
c  y(1)=x, y(2)=vx, y(3)=y, y(4)=vy, y(5)=z, y(6)=vz, x=time
      y(1) = r0
      y(2) = v0 * sin(alpha0) * sfactr
      y(3) = 0.
      y(4) = v0 * sin(beta0) * sfactr
      y(5) = 0.
      y(6) = v0 * cos(alpha0) * sfactr
      vz0 = y(6)
      x = 0.
c
c  get the initial angular momentum
      amom0 = y(1) * y(4)
c
c  set dgear parameters for the first time step
      dt0 = zfixed(1) / (5000. * vz0)
      index = 1
c
c  for each plane at zfixed(k)
      do 425 k = 1,nzfix
c
c  compute the time to reach zfixed(k)
      xend = zfixed(k) / vz0
c
c  integrate the equations of motion up to the desired position
      call dgear(
     a numeqn,fcn6,fcnj,x,dt0,y,xend,tol,meth,miter,index,
     b iwk,wk,ier)
c
c  check for problems
      if( ier .gt. 67 ) then
         write(i6,*) 'error in dgear -- ier=', ier
         stop
      endif
c
c  compute the ion radial position
      if( betam .eq. 0.0 ) then
         rfixed = y(1)
      else
```

```
         rfixed = sqrt( y(1)**2 + y(3)**2 )
      endif
c
c  compute the ion speed to compare to the initial speed
c       v0fix = sqrt( y(2)**2 + y(4)**2 + y(6)**2 )
c       vfrac = (v0 - v0fix) / v0
c       if( abs(vfrac) .gt. 0.2 )
c     1          write(i6,*) 'v0=',v0,' v0fix=',v0fix
c
c  compute the angular momentum to compare to the initial
c       amom = y(1) * y(4) - y(3) * y(2)
c       amfrac = (amom0 - amom) / amom0
c       if( abs(amfrac) .gt. 0.2 )
c     1          write(i6,*) 'amom0=',amom0,' amom=',amom
c
c  put the ion into the radial particle distribution
      do 420 j = 1,10
      if( rfixed .lt. rhist(j,k) ) then
         phist(j,k) = phist(j,k) + 1.
         goto 425
      endif
  420 continue
c
c  end loop over zfixed(k)
  425 continue
c
c  end loop over ions from source
  450 continue
c
c  compute the fluence distribution from the particle distribution
      do 460 k = 1,nzfix
      fhist(1,k) = phist(1,k) / (pi * rhist(1,k)**2)
      do 460 j = 2,10
      fhist(j,k) = phist(j,k) /
     1             (pi * (rhist(j,k)**2 - rhist(j-1,k)**2))
  460 continue
c
c  save the distributions for plotting
      write(i10,1300) nzfix
      write(i10,1301) ((rhist(j,k), j=1,10), k=1,nzfix)
      write(i10,1301) ((phist(j,k), j=1,10), k=1,nzfix)
      write(i10,1301) ((fhist(j,k), j=1,10), k=1,nzfix)
 1300 format(i10)
 1301 format(1p10e10.2)
c
c  write a summary of the calculation
      do 500 k = 1,nzfix
      sum = 0.0
      do 490 j = 1,10
      sum = sum + phist(j,k)
  490 continue
      frac = sum / float(nion)
      write(i6,1400) nion, sum, frac
      write(i6,1401) (rhist(j,k), j=1,10), (phist(j,k), j=1,10)
      write(i6,1402) (rhist(j,k), j=1,10), (fhist(j,k), j=1,10)
  500 continue
 1400 format('0 number of source ions..........', i8/
     1       '  number of ions in distribution.', f8.0/
     2       '  fraction of ions reaching plane', 1p1e12.4)
 1401 format('0 particle distribution..........'/(1p10e10.2))
 1402 format('0 fluence distribution...........'/(1p10e10.2))
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else
      endif
      end
      subroutine taper
c
c  this subroutine computes ion trajectories or positions using
c  the tapered channel model
c
      use all
      use seed
      use save
      use bugs
c
c  choose the type of calculation to be done
      if( icalc .eq. 1 ) then
c
c  we are computing the trajectories of ions from an
c  instantaneous source
c
c  compute some terms that do not depend on r0 and alpha0
      v0 = v0min
      tmax = zchanl / v0
      lost = 0
c
c  loop over "nion" ions from the source
      do 150 ni = 1,nion
c
c  get the radius and angle of injection of an ion
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  get some more terms that depend on r0 and alpha0
      bfreq = sqrt(wcb * v0 / rchanl)
      sina0 = sin(alpha0)
c
c  loop over "ntime" times to get r(t) and z(t) positions to plot
      t = 0.0
      z(1) = 0.0
      r(1) = r0
      do 100 nt = 2,ntime+1
      t = t + deltat
      z(nt) = v0 * t
      sqrttt = sqrt(1. - t / tmax)
      argu = -bfreq * tmax * alog(1. - t / tmax)
      r(nt) = r0 * sqrttt * cos(argu) +
     1        v0 * sina0 / bfreq * sqrttt * sin(argu)
      if( abs(r(nt)) .gt. rchanl ) then
         npts = nt
         lost = lost + 1
         goto 125
      endif
  100 continue
      npts = ntime + 1
  125 continue
c
c  debug output
      if( debug(3) ) then
      write(idebug,1000) npts,lost,(z(i),r(i), i=1,npts)
 1000 format(2i12,(1p2e12.4))
```

```
      endif
c
c  save trajectory for plotting
      write(i10,1100) npts
      write(i10,1101) (z(i),r(i), i=1,npts)
 1100 format(i10)
 1101 format(1p12e10.2)
  150 continue
c
c  write summary of calculation
      frac = float(nion - lost) / float(nion)
      write(i6,1102) nion, lost, frac
 1102 format('0 ions injected.....................', i8/
     1       '  ions lost from channel............', i8/
     2       '  fraction successfully propagated..', 1p1e12.4)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else if( icalc .eq. 2 ) then
c
c  we are computing the z and r positions of ions from a
c  time dependent source
c
c  loop over "ntime" source times
      n = 1
      tprime = -deltat
      do 250 ntp = 1,ntime
      tprime = tprime + deltat
c
c  loop over "nion" source ions at this time
      do 200 ni = 1,nion
c
c  get the radius and angle of injection of an ion
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  distribute the injection time uniformly from tprime to
c  tprime + deltat
      tpr = tprime + rand(random) * deltat
c
c  get the ion velocity -- it may be ramped
      v0 = v0min + dvdt * tpr
      tmax = zchanl / v0
c
c  get some more terms that depend on r0 and alpha0
      bfreq = sqrt(wcb * v0 / rchanl)
      sina0 = sin(alpha0)
c
c  compute the position
      z(n) = v0 * (tsnap - tpr)
      sqrttt = sqrt(1. - (tsnap - tpr) / tmax)
      argu = -bfreq * tmax * alog(1. - (tsnap - tpr) / tmax)
      r(n) = r0 * sqrttt * cos(argu) +
     1        v0 * sina0 / bfreq * sqrttt * sin(argu)
      n = n + 1
  200 continue
  250 continue
c
c  debug output
      if( debug(3) ) then
```

```
      write(idebug,1000) (z(i),r(i), i=1,n-1)
      endif
c
c  save the positions for plotting
      nml = n - 1
      write(i10,1200) nml
      write(i10,1201) (z(i), r(i), i=1,nml)
 1200 format(i10)
 1201 format(1p12e10.2)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else if( icalc .eq. 3 ) then
c
c  we are computing a histogram of ion radial position at fixed
c  z positions.
c
c  get terms that do not depend on r0 and alpha0
      do 400 k = 1,nzfix
      do 400 j = 1,10
      rhist(j,k) = rchanl * (j / 10.)
  400 continue
      v0 = v0min
      tmax = zchanl / v0

      bfreq = sqrt(wcb * v0 / rchanl)

c
c  loop over nion ions from the source
      do 450 n = 1,nion
c
c  get the radius and angle of injection
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  get some more terms that depend on r0 and alpha0
      sina0 = sin(alpha0)
c
c  loop over the number of planes where distributions are desired
      do 445 k = 1,nzfix
      t = zfixed(k) / v0
      sqrttt = sqrt(1. - t / tmax)
      argu = -bfreq * tmax * alog(1. - t / tmax)
c
c  compute the radial position
      rfixed = abs(r0 * sqrttt * cos(argu) +
     1        v0 * sina0 / bfreq * sqrttt * sin(argu))
c
c  put the ion into the radial particle distribution
      do 440 j = 1,10
      if( rfixed .lt. rhist(j,k) ) then
      phist(j,k) = phist(j,k) + 1.
      goto 445
      endif
  440 continue
  445 continue
  450 continue
c
c  compute the fluence distribution from the particle distribution
      do 460 k = 1,nzfix
```

```
      fhist(1,k) = phist(1,k) / (pi * rhist(1,k)**2)
      do 460 j = 2,10
      fhist(j,k) = phist(j,k) /
     1             (pi * (rhist(j,k)**2 - rhist(j-1,k)**2))
  460 continue
c
c  save the distributions for plotting
      write(i10,1300) nzfix
      write(i10,1301) ((rhist(j,k), j=1,10), k=1,nzfix)
      write(i10,1301) ((phist(j,k), j=1,10), k=1,nzfix)
      write(i10,1301) ((fhist(j,k), j=1,10), k=1,nzfix)
 1300 format(i10)
 1301 format(1p10e10.2)
c
c  write a summary of the calculation
      do 500 k = 1,nzfix
      sum = 0.0
      do 490 j = 1,10
      sum = sum + phist(j,k)
  490 continue
      frac = (sum / float(nion))
      write(i6,1400) nion, sum, frac
      write(i6,1401) (rhist(j,k), j=1,10), (phist(j,k), j=1,10)
      write(i6,1402) (rhist(j,k), j=1,10), (fhist(j,k), j=1,10)
  500 continue
 1400 format('0 number of source ions..........', i8/
     1       '  number of ions in distribution.', f8.0/
     2       '  fraction of ions reaching plane', 1p1e12.4)
 1401 format('0 particle distribution..........'/(1p10e10.2))
 1402 format('0 fluence distribution...........'/(1p10e10.2))
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else
c
c  a place for other options
      endif
      return
      end
      function rand(iy)
c
c  there are two versions of this random number generator.  the
c  first is a ibm/pc version and the second uses the imsl library
c  routine found on most mainframe computers.
c
c          ************ ibm/pc version ***************
c  rand is a uniform random number generator based on theory and
c  suggestions given in d.e. knuth (1969), vol 2.  the integer iy
c  should be initialized to an arbitrary integer prior to the first call
c  to rand.  the calling program should not alter the value of iy
c  between subsequent calls to rand.  values of rand will be returned
c  in the interval (0,1).
c
c     integer*2 iy
c     integer*2 ia, ic, itwo, m2, m, mic
c     double precision halfm
c     real s
c     double precision datan, dsqrt
c
c     data m2/0/
c     data itwo/2/
c
c     decide if this is the first entry
```

```
c     if( m2 .eq. 0 ) then
c
c        compute machine integer word length
c        m = 1
c  10    m2 = m
c        m = itwo * m2
c        if( m .gt. m2 ) goto 10
c        halfm = m2
c
c        compute multiplier and increment for linear congruential method
c        ia = 8 * idint(halfm * datan(1.d0)/8.d0) + 5
c        ic = 2 * idint(halfm * (0.5d0 - dsqrt(3.d0)/6.d0)) + 1
c        mic = (m2 - ic) + m2
c
c        s is the scale factor for converting to floating point
c        s = 0.5 / halfm
c     endif
c
c     compute next random number
c     iy = iy * ia
c
c     the following statement is for computers which do not allow
c     integer overflow on addition
c      if( iy .gt. mic ) iy = (iy - m2) - m2
c
c     iy = iy + ic
c
c     the following statement is for computers where the
c     word length for addition is greater than for multiplication
c      if( iy / 2 .gt. m2 ) iy = (iy - m2) - m2
c
c     the following statement is for computers where integer
c     overflow affects the sign bit
c     if( iy .lt. 0 ) iy = (iy + m2) + m2
c     rand = float(iy) * s
c     return
c
c
c        ************ imsl version ****************
c  this routine calls the system dependent random number generator
c  we are using the imsl library routine ggubfs.
c
      double precision iy
c
c  get the random number
      rand = ggubfs(iy)
      return
      end
      subroutine strait
c
c  this subroutine computes ion trajectories or positions using
c  the straight channel model
c
      use all
      use seed
      use save
      use bugs
c
c  choose the type of calculation to be done
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
      if( icalc .eq. 1 ) then
c
c  we are computing the trajectories of ions from an instantaneous
c  source
c
c  compute some more terms that do not depend on r0 and alpha0
      v0 = v0min
      lost = 0
c
c  loop over "nion" ions from the source
      do 150 ni = 1,nion
c
c  get the radius and angle of injection of an ion
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  determine whether the ion is trapped or whether it escapes the
c  channel b - field before any trajectory is computed
      trapal = trap * sqrt( 1.0 - min((r0/rchanl)**2, 1.0) )
      if( alpha0 .gt. trapal ) then
         lost = lost + 1
         goto 150
      endif
c
c  get some more terms that depend on r0 and alpha0
      cosa0 = cos(alpha0)
      sina0 = sin(alpha0)
      tana0 = sina0 / cosa0
      bfreq = sqrt(wcb * v0 * cosa0 / rchanl)
      rbar = sqrt(r0**2 + rchanl * v0 * sina0**2 / (wcb * cosa0))
      phi = atan(-sqrt(rchanl * v0 * cosa0 / (wcb * r0**2))
     1    * tana0)
      wb = bfreq * (1. - wcb * rbar**2 / (16. * rchanl * v0 * cosa0)
     1   + 0.25 * tana0**2)
c
      z1 = v0 * cosa0 - wcb / (4. * rchanl) * rbar**2 * cos(2. * phi)
      z2 = rbar**2 / (8. * rchanl) * sqrt(rchanl * wcb / (v0 * cosa0))
      z3 = sin(2. * phi)
c
c  loop over ntime times to get z(t) and r(t) positions to plot
      t = 0.0
      tprime = 0.0
      z(1) = 0.0
      r(1) = r0
      do 100 nt = 2,ntime+1
      t = t + deltat
      z(nt) = z1 * t + z2 * (sin(2. * (wb * t + phi)) - z3)
      r(nt) = rbar * cos(wb * t + phi)
      if( abs(r(nt)) .gt. rchanl ) then
         npts = nt
         lost = lost + 1
         goto 125
      endif
  100 continue
      npts = ntime+1
  125 continue
c
c  debug output
      if( debug(2) ) then
      write(idebug,1000) npts,lost,(z(i),r(i), i=1,npts)
```

```
 1000 format(2i12,(1p2e12.4))
      endif
c
c  save trajectory for plotting
      write(i10,1100) npts
      write(i10,1101) (z(i),r(i), i=1,npts)
 1100 format(i10)
 1101 format(1p12e10.2)
  150 continue
c
c  write summary of calculation
      frac = float(nion - lost) / float(nion)
      write(i6,1102) nion, lost, frac
 1102 format('0 ions injected....................', i8/
     1       '  ions lost from channel............', i8/
     2       '  fraction successfully propagated..', 1p1e12.4)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else if( icalc .eq. 2 ) then
c
c  we are computing the z and r positions of ions from
c  a time dependent source
c
c  loop over ntime source times
      n = 1
      tprime = -deltat
      do 250 ntp = 1,ntime
      tprime = tprime + deltat
c
c  loop over nion source ions at this time
      do 200 ni = 1,nion
c
c  get the radius and angle of injection of an ion
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  distribute the injection time uniformly from tprime to
c  tprime + deltat
      tpr = tprime + rand(random) * deltat
c
c  get ion velocity - it may be ramped
      v0 = v0min + dvdt * tpr
c
c  get some more terms that depend on r0 and a0
      cosa0 = cos(alpha0)
      sina0 = sin(alpha0)
      tana0 = sina0 / cosa0
      bfreq = sqrt(wcb * v0 * cosa0 / rchanl)
      rbar = sqrt(r0**2 + rchanl * v0 * sina0**2 / (wcb * cosa0))
      phi = atan(-sqrt(rchanl * v0 * cosa0 / (wcb * r0**2))
     1    * tana0)
      wb = bfreq * (1. - wcb * rbar**2 / (16. * rchanl * v0 * cosa0)
     1   + 0.25 * tana0**2)
c
      z1 = v0 * cosa0 - wcb / (4. * rchanl) * rbar**2 * cos(2. * phi)
      z2 = rbar**2 / (8. * rchanl) * sqrt(rchanl * wcb / (v0 * cosa0))
      z3 = sin(2. * phi)
c
c  compute the position
```

```
      z(n) = zl * (tsnap - tpr) +
     1       z2 * (sin(2. * (wb * (tsnap -tpr) + phi)) - z3)
      r(n) = rbar * cos(wb * (tsnap - tpr) + phi)
      n = n + 1
  200 continue
  250 continue
c
c  debug output
      if( debug(2) ) then
      write(idebug,1000) (z(i),r(i), i=1,n-1)
      endif
c
c  save the positions for plotting
      nml = n - 1
      write(i10,1200) nml
      write(i10,1201) (z(i), r(i), i=1,nml)
 1200 format(i10)
 1201 format(1p12e10.2)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else if( icalc .eq. 3 ) then
c
c  we are computing a histogram of ion radial position at a fixed
c  z position.
c
c  get terms that do not depend on r0 and alpha0
      do 400 k = 1, nzfix
      do 400 j = 1,10
      rhist(j,k) = rchanl * (j / 10.)
  400 continue
      v0 = v0min
c
c  loop over nion ions from the source
      do 450 n = 1,nion
c
c  get the radius and angle of injection
      call ionsrc(
     1 rsourc,alpham,betam,
     3 r0,alpha0,beta0)
c
c  get some more terms that depend on r0 and a0
c
      cosa0 = cos(alpha0)
      sina0 = sin(alpha0)
      tana0 = sina0 / cosa0
      bfreq = sqrt(wcb * v0 * cosa0 / rchanl)
      rbar = sqrt(r0**2 + rchanl * v0 * sina0**2 / (wcb * cosa0))
      phi = atan(-sqrt(rchanl * v0 * cosa0 / (wcb * r0**2))
     1    * tana0)
      wb = bfreq * (1. - wcb * rbar**2 / (16. * rchanl * v0 * cosa0)
     1   + 0.25 * tana0**2)
c
      zl = v0 * cosa0 - wcb / (4. * rchanl) * rbar**2 * cos(2. * phi)
      z2 = rbar**2 / (8. * rchanl) * sqrt(rchanl * wcb / (v0 * cosa0))
      z3 = sin(2. * phi)
c
c  for each plane where a distribution is desired
      do 445 k = 1, nzfix
c
c  estimate the time that an ion takes to reach z-fixed.  this will
```

```
c  always be less than the actual time
      t0 = zfixed(k) / v0
c
c  get the computed z position at this estimated time
      z0 = z1 * t0 + z2 * (sin(2. * (wb * t0 + phi)) - z3)
c
c  estimate a time step to allow us to approach the correct z-fixed
      t1 = t0 * (zfixed(k) / z0)
      deltat = (t1 - t0) / 100.
c
c  find the correct time for an ion to reach zfixed
      do 425 i = 1,1000
      t = t0 + i * deltat
      ztest = z1 * t + z2 * (sin(2. * (wb * t + phi)) - z3)
      if( ztest .ge. zfixed(k) ) goto 430
  425 continue
c
c  if we reach here then we did not find t corresponding to zfixed
      write(i6,*) 'did not find time for z-fixed'
      stop
c
c  we have found the time, now evaluate the radial position
  430 continue
      rfixed = abs(rbar * cos(wb * t + phi))
c
c  put the ion into the radial particle distribution
      do 440 j = 1,10
      if( rfixed .lt. rhist(j,k) ) then
      phist(j,k) = phist(j,k) + 1.0
      goto 445
      endif
  440 continue
  445 continue
  450 continue
c
c  compute the fluence distribution from the particle distribution
      do 480 k = 1,nzfix
      fhist(1,k) = phist(1,k) / (pi * rhist(1,k)**2)
      do 470 j = 2,10
      fhist(j,k) = phist(j,k) /
     1             (pi * (rhist(j,k)**2 - rhist(j-1,k)**2))
  470 continue
  480 continue
c
c  save the distributions for plotting
      write(i10,1300) nzfix
      write(i10,1301) ((rhist(j,k), j=1,10), k=1,nzfix)
      write(i10,1301) ((phist(j,k), j=1,10), k=1,nzfix)
      write(i10,1301) ((fhist(j,k), j=1,10), k=1,nzfix)
 1300 format(i10)
 1301 format(1p10e10.2)
c
c  write a summary of the calculation
      do 500 k = 1,nzfix
      sum = 0.0
      do 490 j = 1,10
      sum = sum + phist(j,k)
  490 continue
      frac = (sum / float(nion))
      write(i6,1400) nion, sum, frac
      write(i6,1401) (rhist(j,k), j=1,10), (phist(j,k), j=1,10)
```

```
      write(i6,1402) (rhist(j,k), j=1,10), (fhist(j,k), j=1,10)
  500 continue
 1400 format('0 number of source ions..........', i8/
     1      '  number of ions in distribution.', f8.0/
     2      '  fraction of ions reaching plane', 1p1e12.4)
 1401 format('0 particle distribution..........'/(1p10e10.2))
 1402 format('0 fluence distribution...........'/(1p10e10.2))
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      else
c
c  a place to put other options
      endif
      return
      end
      subroutine bxyz(
     1 targ,xarg,yarg,zarg,b0,rchanl,rchsq,
     2
     3 bx,by,bz)
c
c  this subroutine computes b-x,y,z and must be supplied
c  by the user.
c
      use bugs
c
c  see whether we are inside or outside the channel
      xysq = xarg**2 + yarg**2
      if( xysq .lt. rchsq ) then
c  we are inside
c        bx = b0 * yarg / rchanl
c        by = b0 * xarg / rchanl
         bx = b0 * (yarg / rchanl) ** 5
         by = b0 * (xarg / rchanl) ** 5
         bz = 0.0
      else
c  we are outside
         if( yarg .ne. 0. ) then
            bx = b0 * rchanl / yarg
         else
            bx = 0.
         endif
         if( xarg .ne. 0. ) then
            by = b0 * rchanl / xarg
         else
            by = 0.
         endif
         bz = 0.0
      endif
c
c  debug output
      if( debug(7) ) then
      write(idebug,*)
     1     ' bxyz - targ,xarg,yarg,zarg,b0,rchanl,bx,by,bz'
      write(idebug,*) targ,xarg,yarg,zarg,b0,rchanl,bx,by,bz
      endif
c
      return
      end
      subroutine fcn6(
     1 n,x,y,
```

```
     2
     3 yprime)
c
c  this subroutine provides the imsl library gear routine
c  (dgear) with the right hand sides of the six o.d.e.'s to
c  be solved
c
      real y(n), yprime(n)
c
      use all
      use bugs
c
c  first call the bxyz routine to compute the field at position
c  x = y(1), y = y(3), and z = y(5) at time = x.
      call bxyz(
     1 x,y(1),y(3),y(5),b0,rchanl,rchsq,
     2
     3 bx,by,bz)
c
c  now compute the right hand sides
      yprime(1) = y(2)
      yprime(2) = qimic * (y(4) * bz - y(6) * by)
      yprime(3) = y(4)
      yprime(4) = qimic * (y(2) * bz - y(6) * bx)
      yprime(5) = y(6)
      yprime(6) = qimic * (y(2) * by - y(4) * bx)
c
c  do we want debug output
      if( debug(6) ) then
         write(idebug,1000) (y(i), i=1,6)
 1000    format(' y(i)=',1p6e12.4)
         write(idebug,1001) (yprime(i), i=1,6)
 1001    format(' yprime(i)=',1p6e12.4)
         write(idebug,1002) x,bx,by,bz,qimic
 1002    format(' x=',1p1e12.4,'  bx=',e12.4,'  by=',e12.4,
     1          '  bz=',e12.4,'  qimic=',e12.4)
      endif
c
c  all done
      return
      end
```