

1 of 2

9/93 Jan 1

UCRL-ID-110499

The Best of GYMNOS: A Users Guide

D. W. Hewett and D. J. Larson

August 1993



Lawrence
Livermore
National
Laboratory

This is an informal report intended primarily for internal or limited external distribution. The opinions and conclusions stated are those of the author and may or may not be those of the Laboratory.

Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

**This report has been reproduced
directly from the best available copy.**

**Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (615) 576-8401, FTS 626-8401**

**Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161**

The Best of GYMNOs: A Users Guide

D.W. Hewett and D.J. Larson

University of California

Lawrence Livermore National Laboratory

Livermore, California 94550

I. Introduction, History, & Acknowledgement.....	3
II. Overall Philosophy and Verbal Flowchart.....	5
III. Initialization.....	8
III.A) INIT.....	9
III.A.1) GEOSSET called by INIT (See V. BCSET).....	9
III.A.2) Species Input; Control Arrays NSTPROP, BEAM.....	10
III.A.3) Macroscopic Profile Generation using INPROF.....	11
III.A.4) Particle initialization.....	12
III.A.4.a) Particle Parameter RPPSP Computed and Buffering Initialized with SETB.....	12
III.A.4.b) Particle Initialization using Particles/Cell NPARTK.....	12
IV. The Central Time Integration Loop.....	16
IV.A) TRANSPORT of the particles from TIME to Time+DT.....	17
IV.A.1) Start New Species, AUTOVLM, DPARTS, DPART Call PBCSET (See V. BCSET), Preset E & B.....	18
IV.A.2) Call PARMOVE-The Particle Time Integration.....	19
IV.A.2.a) Main Boris Push, Call TRAP, First Order Advance for Advanced Velocity Moments.....	19
IV.A.2.b) TRAP of the Particles Outside Plasma Region.....	23
IV.A.2.b.1) Processing LOST PARTICLES.....	23
IV.A.2.b.1.a) Determining Closest Corner.....	23
IV.A.2.b.1.b) Perfect Reflection or Delete.....	24
IV.A.2.b.2) Thermal Reemission.....	27
IV.A.2.b.2.a) Controls from PBCSET.....	27
IV.A.2.b.2.b) Mechanics.....	28
IV.A.2.b.3) Injection and Field Emission.....	29
IV.A.2.b.3.a) Controls from PBCSET.....	29
IV.A.2.b.3.b) Mechanics.....	30
IV.A.3) Finish a Species, AUTOVLM, Restore E & B, DAPART Convert Raw Counts to Densities, Save Histories.....	32
IV.B) Call DSOURCE (See VI. Diagnos).....	32
IV.C) Call FIELD-The Director of the Field algorithm, using FLDSET (See V. BCSET).....	32
IV.D) Call DFIELD (See VI. DIAGNOS).....	35

MASTER

eb

V) User Specified Physics Routines	
V.A) Macroscopic Initial Profile Generation by INPROF.....	36
V.B) User Specified Boundary/Structure Physics in BCSET	
V.B.1) GEOSET.....	37
V.B.2) PBCSET.....	41
V.B.3) FLDSET.....	44
VI) User Specified Diagnostics in DIAGNOS	
VI.A) DPART Generates Particle Phase-Space Plots.....	49
VI.B) DAPART Generates Particle Position-Space Plots.....	49
VI.C) DASPECS Plots Macroscopic Source Terms from Each Species.....	50
VI.D) DSOURCE Plots Species-Summed Macroscopic Source Terms.....	50
VI.E) DFIELD Plots Potentials and Fields A, Phi, E, and B.....	50
VI.F) HISTORY Plots Histories After "Unraveling" Pointer.....	50
VII) Running GYMNOS	
VII.A) Subroutines in GYM30.....	52
VII.B) Subroutines in GYMS22.....	54
VII.C) A Test Case: The LBL Source.....	55
VIII) Summary and Conclusions.....	57

APPENDICES

Appendix 1. geoMset.....	59
Appendix 2. GRAPHICS.....	63
Appendix 3. BIRTH.....	70
Appendix 4. TYPICAL INPUT "DECKS"	72
Appendix 5. WRSTART.....	75
Appendix 6. EFFVOL.....	76
Appendix 7. AUTOVLM.....	78
Appendix 8. PARTICLE BUFFERING ROUTINES: SETB, GETB, RESETB.....	79
Appendix 9. VDKR.....	83
Appendix 10. VTRIK.....	89
Appendix 11. VPARMOV/VTRAP/VBIRTH/DEATH.....	90
Appendix 12. PARMOVR.....	96
Appendix 13. Santa Fe Numerical Simulation Abstract.....	98
Appendix 14. GYM28 SUMMARY SHEETS.....	102

I. Introduction

The code GYMNOS is a 2.5D magnetostatic PIC code that is distinguished from the multitude of similar codes by its ability to handle boundary conditions and internal structures. This code was started (see next section) during the period starting from December 1987 and the major sections were completed and in routine use by the end of 1990. The code is written in R-Z geometry (axisymmetric $m=0$ only) but could easily be modified to handle an X-Z configuration. The several competitors of this code, having similar (but we believe less general) capabilities, distinguish themselves in a different way: They have very formal, thick instruction manuals and usually command an army of people to maintain both the code and said manual. We are here attempting to convince the reader that GYMNOS is better in these two regards. Offered here is a description of a code with a great deal of physics capability coupled with no interactive graphics capability and very little system capability. These latter two subjects contribute most of the complexity in both the code and manual in the competition! Expressed another way, the user will almost never be thwarted by unexpected system changes on Friday afternoon. The price extracted is that the user is required to monitor/babysit runs so that output is "hand-delivered" to the output medium and properly archived to prevent loss. A further spinoff of this simplicity is that "single pilot" operation is easy. And, the only manual is this modest document.

History and Acknowledgements

During the Christmas holidays in 1987, I found myself with an unusual opportunity. My employer, LLNL, found it necessary to allow its staff to take required vacation between Christmas and New Year and, with my second child about to be born (1/12/88), I found the time to try some new ideas concerning techniques to impose boundary conditions on internal structures in my favorite elliptic solving algorithms using ADI. As you will discover reading this, I was able to incorporate almost complete generality for each mesh point into an algorithm that could be completely vectorized using key or template arrays.

The success of this venture left me with a very versatile algorithm that could quickly solve not only the Poisson equation but also the curl curl operator. After a few test cases around $r=0$ that suitably humbled me, I embarked on the construction of a PIC code that could serve as a vehicle for a renewed Darwin effort and a host of fluid electron, PIC kinetic ion models. After a series of lighthearted names, the name GYMNOS looks like it will stand the proverbial test of time for the basic code. The best justification is that this bare-bones vehicle for all the more sophisticated models to come is named the Greek word for naked.

As it turned out however, this "basic" code now has developed many capabilities. There is much to be done even with simple internal structures. GYMNOS now has the ability to generate its own internal structures given the essential information that it have a "bar" or "pie" cross section in the R-Z plane with a starting and ending point plus a width for the bar and a center, radius, and starting and stopping azimuth for the pie. Particles (perhaps multiple species) may be perfectly reflected, fully absorbed, absorbed with a fraction thermally reemitted, field-emitted using the self-consistent electric field, or injected with prescribed density, normal velocity, temperature, and temperature anisotropy

from any of these structures. The particle pusher itself is a Boris pusher that is fully vectorized.

The code accumulates particle and stores both particle and field data on cell corners. Internal structures are defined only on cell corners or lines that join nearest neighbor corners. The above mentioned particle absorption, injection, field emission, and reflection also take place at such corners (but thermal reemission comes from the "surfaces" between cell corners on structures surfaces). "Raw counts" from particle accumulation are converted to true densities in a manner that automatically accounts for the reduced volume of some of these cells adjacent to internal boundaries.

I experienced the joy of having a graduate student "come up to speed" during this period. David J. Larson (UC Davis) suffered through most of the development of the particle/structure interaction. His participation ranged from "sounding board" in the early stages to bug finder/fixer as he applied GYMNOS to the MIRRORTRON concept. David is now quite capable of conceiving and building enhancements in his own right: the relativistic particle pusher is wholly his implementation of the relativistic ideas in Birdsall and Langdon applied to our RZ Boris pusher. David is now implementing the "Streamlined Darwin Field" model using the new coupled equation solver he helped to invent. He is building this new capability into a variant of GYMNOS that he calls BEAGLE (the name of the ship that carried THE Charles Darwin on his origin-of-the-species voyage!). I suspect we will all hear much more from this young physicist in the years ahead.

Dennis W. Hewett 10/21/91

II. Overall Philosophy and Verbal Flowchart

What follows is a description of the software of GYMNOS. The description is necessarily given on several levels, starting with the most cursory overview.

```
c      GYMNOS (RZ PIC with arbitrary boundaries)
c          date: GYMNOS 3/28/91  started July 14, 1988

      common/param/  jstgnsw
      ...
      call first(idrun,idcode,'GYMNOS',memoly,imsg)
c      Opens input file, initializes plots, cr8s output files
      ...
c      Check dimensions, create "constants" pi,c/pi4, etc.
c      Input geometry and BC physics.
c
c      Reading a restart?  Check NREST
if (nrest.lt.0) then
  call rstart(...)
else
  it=0           time=0.
  call init
endif

c
c      TIME ADVANCE
100 it=it+1      time=time+dt
      if (time.gt.(tmax+dth)) go to 200
c      Set RESTART flag
      call trans          call dsouce
      call field          call dfield
      call history
c      Check RESTART flag, write RESTART if needed.
      call wstart(....)
c      Check for TTY interrupts  N gives particle numbers
c          IOTTY gives TTY print. ABORT stop 777
      if (imsg.eq.1) call msproc(imsg,...,iotty,...)
      go to 100
200 call endplt
      stop 777
end
```

- 1) In the beginning, the code sets physical constants, creates files, initializes graphics, and opens the input file. This is done by a call to subroutine FIRST.
- 2) Namelist NDATA is read next. Most importantly, the namelist contains a time limit

TMAX, time step size DT, plot intervals and a switch NREST that determines whether the code starts from an existing restart file or from a new initialization. If this is a new initialization the data in NDATA that defines the simulation region (including provisions for internal structures other than the default configuration) will be used. A parameter JSTGNSW, tells the code either to use the default structure configuration, to expect to find more information in the input file, or to query the tty immediately after start up for additional structure information. Included here is also a switch controlling the initial plasma configuration to be used- often signaled by ISTATE-and various voltages, time delays, and turnon and turnoff times.

3) The run is actually initialized by a call to routine INIT. INIT first causes the code geometry to be initialized via a call to GEOSET. GEOSET sets up at least 4 structures: typically, structure number NSTR=1 is a structure at RMIN containing all the points I=1 for all J. Similarly, NSTR=2 contains all the points at r=RMAX, or I=NR1, $1 \leq J \leq NZ1$. NSTR=3,4 are the boundaries at ZMIN,ZMAX respectively. Structures 3,4 will not exist if these boundaries are periodic. Just which structure the corner points go into depends on the desired physics. STRGEN is the array which contains the controlling information and GEOSET, along with possible changes from input, sets this array. It then calls GEOMSET which set more control arrays such as SKEY, NSTRUCT, KORNT, KCIJ, KCORR. More details are available in the discussion of the entries in BCSET.

INIT reads in a namelist NECHSP for each species that includes a species name NAMESP, the initial number of particles NP, the maximum number NP MAX, the real charge per particle, the real mass per particle, the minimum and maximum density, the temperature (with possible anisotropy), and drift speeds.

Equally important, these species namelists bring information that set the physical properties of the structure surfaces. Properties such as particle reflection, absorption and thermal reemission, field emission, and injection are carried in the arrays NSTPROP and BEAM. NSTPROP(NSTR,N) determines which species are active on a given structure. BEAM(NSTR,N,8) carries more essential injection/thermal reemission/field emission parameters such as density, normal velocity, anisotropy, and turnon/turnoff times.

For each species N, INIT calls subroutine INPROF for each grid point. INPROF return a cell center density, vector drift velocity, temperature, vector electric E and magnetic B fields. Using the the curl of B as the source, INIT call a Poisson solver that returns the initial vector potential A.

Particle initialization is the next task. INIT calls SETB to initialize the particle storage and buffering required. Particles can now be created and stored in buffers that are accessed using calls to GETB. After determining several necessary parameters, most importantly the parameter Real Particles Per Simulation Particle RPPSP for each species, the the NP initial particles for each N are allocated to the various cells so that when we sum the particles and scatter them to the mesh, we recover roughly the same macroscopic moments we have defined in INPROF. This check is precisely what is accomplished in the bottom of INIT when a time step with DT=0 is performed.

4) The main time integration loop is in the MAIN routine, starting with statement label 100. First subroutine TRANS is called. This accomplishes a time step for all species and the accumulation of plasma source terms for each species as well as the net plasma

charge and current densities. TRANS calls various entries in DIAGNOS as needed to get diagnostics during and after each species is integrated. MAIN calls the DIAGNOS entry DSOURCE to diagnos the net plasma sources and then calls FIELD which orchastrats the construction of new field components given the newly updated plasma sources. An entry DFIELD in DIAGNOS then diagnoses these new fields. A HISTORY entry in DIAGNOS is next called as needed as is a procedure NEEDF3 to close a plot file and open a new one. Finally, a restart file is written periodically and, almost always, when TIME exceeds TMAX.

5) MAIN closes all files with a call ENDPLT for graphics and a STOP 777, the usual signal for normal termination.

The code encompasses roughly 10,000 lines of FORTRAN, including the routines directly called by the five steps outlined above, a support binary package containing the Poisson solvers and related routines, and a plot library. There is a wealth of essential detail in these support routines. Many are essential parts of this "toolkit" that has been growing and evolving since the beginning of computers. The biggest since source was probably Claire Nielson and his colleagues at LANL circa 1973—they probably won't recognize much other than some issues of style now and there is certainly no reason to hold anyone other than DWH responsible for accuracy and lingering errors. Countless incidents have convinced us that such errors will always exist and will materialize when new parameter regimes are attempted. However, we also believe that these bugs have little effect on the results of the test cases presented here.

A final comment, this code is written entirely in CGS units with temperatures in eV and most all mesh quantities are defined at cell corner. Generally formulas can be found in the NRL FORMULARY. We now discuss the various sections of the code in greater detail.

III. Initialization

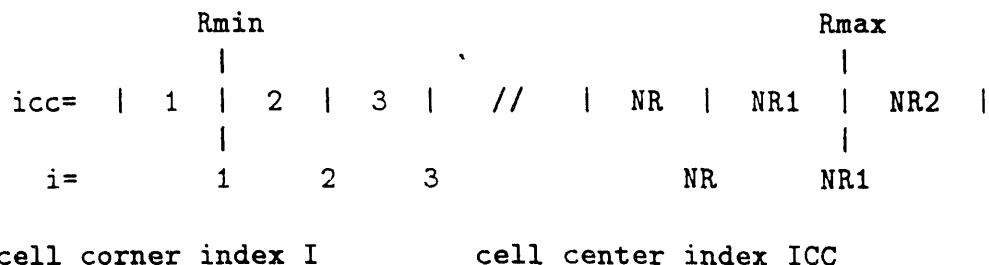
Having already seen the main time loop in the previous section, we now discuss bits and pieces of the code that require extra discussion to flesh out the picture that is always contained in the Fortran. The Fortran source is the ultimate reference. Right or wrong the code does only what it is told; the words that we add here only convey the concept of the algorithm that we intend for the code to accomplish.

DWH almost never uses explicit type declarations. Dennis makes fewer mistakes if he adheres to time honored integer and floating point variable names. DJL has been forced into this mode as well! Dennis has finally succumbed to the use of JR/Q/Z as the name of real current arrays.

real jr, jq, jz

I also use “Q” or “q” to represent θ throughout the code.

GYMNOS is built on a uniform rectangular mesh with the I index associated with the R or radial variable. J is associated with the Z or axial variable. The mesh points run from I=1 at RMIN to I=NR1 at RMAX where NR1=NR+1 and each mesh point is at a cell corner of the NR real cells in the problem. Similarly, J runs from 1 to NZ1 and spans NZ real cells in the axial direction. The code actually runs on a mesh that is dimensioned NR2 X NZ2 where NR2=NR+2 and NZ2=NZ+2. All two dimensional quantities are defined on such meshes and dimensioned accordingly. The purpose is that I occasionally use a *cell centered* indexing scheme with a ghost cell on each end of the set of real cells already defined. A diagram of the r-axis makes things a little clearer.



A parameter statement defines NR2 and NZ2 in the top of the main code (these are problem dependent).

```
parameter (nr2=73,nz2=19,mxg1d=73,nstmax=8,ithmx=2000,kcormx=700)
```

Soon after another parameter statement defines other useful grid parameters.

```

parameter (nr1=nr2-1,nr=nr2-2,nr3=nr2+1,nz1=nz2-1,nz=nz2-2,
. nrm1=nr-1,nzm1=nz-1,nr2nz2=nr2*nz2,nr2nz=nr2*nz,
. nr2nz1=nr2*nz1)

```

The physical dimensions that go with the counting indices can now be computed. If we are beginning a new run, INIT will be called here and these mesh parameters will be computed there. (If we are starting from a restart file, we need only compare a few things we have just input with what is already in the restart file to insure compatibility.)

III.A) INIT

At the beginning of a new run, many things need to be set up and INIT is the subroutine that does most of this—or at least directs it. To begin, we first compute those mesh lengths that go with the mesh discussed in the previous section. The dimensions of the mesh cells are straightforward and they and several related quantities are computed are computed and stored to ease the workload of constructing finite difference schemes throughout the code. (Caution! We put multiple Fortran statements on the same line to save paper—read left to right then down.)

```

dr=(rmax-rmin)/float(nr)      dz=(zmax-zmin)/float(nz)
drz=sqrt(dr**2+dz**2)
dri=1./dr                      dzi=1./dz
dris=dri*dri                   dzis=dzi*dzi
drih=.5*dri                     dzih=.5*dzi
drh=.5*dr                       dzh=.5*dz

c
do 20 j=1,nz2                  zgcc(j)=zmin+(j-1.5)*dz
20 zg(j)=zmin+(j-1)*dz
do 30 i=1,nr2                  rgcc(i)=rmin+(i-1.5)*dr
      rg(i)=rmin+(i-1)*dr       volcc(i)=pi2*dr*dz*rgcc(i)
30  vol(i)=pi2*dr*dz*rg(i)
      vol(1)=pi*drh*(rgcc(2)+rg(1))*dz
      vol(nr1)=pi*drh*(rg(nr1)+rgcc(nr1))*dz

```

Note that RG, Radial Grid, is a cell corner quantity. RGCC is a Cell Center quantity. Similarly, the suffix CC has the same meaning for the z-axis and for the cell volume definitions. Notice that VOL needs to be adjusted at each end to account for the “volume” outside of the simulation region.

Storage location	
CORNERS	rg,zg,vol, ipbc, kornt, key, kn, bv, bvr,bvz
CENTERS	skey, rgcc, zgcc, volc

III.A.1) GEOSET called by INIT (See V. BCSET)

INIT has just set most of the elementary grid parameters. It now calls GEOSET to initialize the arrays that determine structures. To accomplish this GEOSET (a user/problem specific routine) will fill arrays NSTRUCT, NSTPROP, and STRGEN. Next GEOSET calls the routine GEOMSET that interprets STRGEN to build controlling arrays NSTRUCT,

KCIJ, and KCORR that will be used almost everywhere else in GYMNOS. Details can be found in Section V. and Appendix 1.

```

c      NSTRUCT(NSTR,1)--name of structure numbered NSTR
c      NSTRUCT(NSTR,2/3)--KCOR start/stop pt in KCIJ.
c      IorJCOR = KCIJ(KCOR,1or2) for structure point KCOR
c      ex: do 7 kcor=nstruct(nstr,2),nstruct(nstr,3)
c            7 key(kcij(kcor,1),kcij(kcor,2))=-1
c      KCORR(icor,jcor)=KCOR (cross reference array)
c      NSTPROP(NSTR,Nspecies)--n00-n50, n51-%ThRemt,PRFLCT
c      (n=1/2/0 Femt/Njct/Neither), negative=no dithr

call geoset(Jphi)

```

III.A.2) Species Input; Control Arrays NSTPROP, BEAM

In this section of INIT, we load the particle parameters species by species. In addition, species dependent beam and structure parameters are also input.

```

namelist /nechsp/ np,npmax,pchrg,pmass,sdenmx,sdenmn,stemin,
.   sdrftr,sdrftq,sdrftz,sbeam,nstppr, sdithrl, sdithzl, jestatic,
.   svrvrt,svqvt,svzvt,namesp,svrlim,svqlim,svzlim

c          Set fields and macroscopic sources
do 200 n=1,nsp
      ...          zero and set defaults
read (6,nechsp)
if (npmax.le.0) npmax=np
write (7,nechsp)
namsp(n)=namesp    mspec(n)=npmax  nspec(n)=np  npinit(n)=np
sqsgnl(n)=pchrg    smsngl(n)=pmass
dithrl(1,n)=sdithrl  dithrl(2,n)=sdithzl
denmx(n)=sdenmx    denmn(n)=sdenmn    temin(n)=stemin
udrftr(1,n)=sdrftr  udrft(2,n)=sdrftq    udrft(3,n)=sdrftz
vrvt(n)=svrvrt     vqvt(n)=svqvt      vzvt(n)=svzvt
vrlim(1,n)=svrlim(1)  vqlim(1,n)=svqlim(1)  vzlim(1,n)=svzlim(1)
vrlim(2,n)=svrlim(2)  vqlim(2,n)=svqlim(2)  vzlim(2,n)=svzlim(2)
iestatic(n)=jestatic
do 135 nstr=1,nstrmx
do 135 i=1,8
beam(nstr,n,i)=sbeam(nstr,i)
135 nstprop(nstr,n)=nstppr(nstr)
wpn=sqrt(pi4*pchrg*pchrg*sdenmx/pmass)
vthn=sqrt(boltz*k*temin(n)/smsngl(n))

```

```

        write (59/7/100,143) n,namesp,wpn,wpn*dt,vthn,vthn*dt
143 format (8x,'n      namesp',7x,'wpn      wpn*dt',6x,
           'vthn      vthn*dt'/i9,2x,a8,4(1pe10.2))
        write (100,nechsp)
200 write (59,nechsp)
c          ALL species information input now complete.

```

III.A.3) Macroscopic Profile Generation using INPROF

INIT calls INPROF for every cell center to get the ideal macroscopic moments for the PIC represented species. These cell centered moments serve as weight functions for the soon-to-initialized particle distributions. DWH first learned about this method from A.G. Sgro who had implemented these ideas in 1D. Initial field configurations could also be initialized this way but for elliptic field models return self consistent fields without need of any previous information given the instantaneous plasma source configuration. Other models DWH has written (such as zero- and finite-electron-mass hybrid models) used this procedure.

```

c      for initialization,
c      istate=1      Homogeneous Load
c              =2      Exponential z Profile
c              =3      Realistic LBL Source

do 300 n=1,nsp
do 210 j=1,nz1    do 210 i=1,nr1
call inprof(i,j,n,rg(i),zg(j),deno,tpo,uro,uqo,uzo)
ur(i,j,n)=uro      uq(i,j,n)=uqo      uz(i,j,n)=uzo
den(i,j,n)=deno    tem(i,j,n)=tpo
210 tpmmin=amin1(tpo,tpmin)
tpmin=amax1(0.,-tpmin)          (A procedure to guarantee
do 220 j=1,nz1    do 220 i=1,nr1    positive TEMperatures.)
220 tem(i,j,n)=tem(i,j,n)+tpmin
do 230 i=1,nr1    do 230 j=1,nz1
230 tem(i,j,n)=cvmgp(tem(i,j,n),0.,den(i,j,n)-dmin).
c      END of macroscopic equilibrium initializer
c      set initialize velocity plot limits
call maxv(tem(1,1,n),1,nr2nz2,mloc,temmx)
call aminmx(ur(1,1,n),1,nr2nz2,1,urmn,urmx)
call aminmx(uq(1,1,n),1,nr2nz2,1,uqmn,uqmx)
call aminmx(uz(1,1,n),1,nr2nz2,1,uzmn,uzmx)
vthrm=sqrt(boltzk*temmx/pmass)
if (vrlim(1,n).eq.0.) vrlim(1,n)=amin1(0.,urmn)-3.*vthrm
if (vrlim(2,n).eq.0.) vrlim(2,n)=amax1(0.,urmx)+3.*vthrm
if (vqlim(1,n).....

```

```

if (vzlim(2,n)...
call aplotx(den(1,1,n),nr2,ravg,zavg,rsc,zsc,
-2,-2,-3,-2,nr1,nz1,1,nr1,namsp(n),8,'aplt den',8,1,1,1)
call cplot(-2,-3,nr1,nz1,rsc,zsc,-1,-1,nr2,den(1,1,n),
x           -mcont,zc,'Den',3,'r',1,'z',1)
300 call vplot(-3,-3,nr1,nz1,rsc,zsc,-ncr,-ncz,nr2,
ur(1,1,n),uz(1,1,n),'Ur,z',-4,'r',1,'z',1)

```

III.A.4) Particle initialization

III.A.4.a) Particle Parameter RPPSP Computed and Buffering Initialized with SETB

```

...
c      N is species number TOTNRP...sum real prtcls each N
c      NPINIT(n)=NP...number of simulation particles requested
c      NSPEC(n) ...instantaneous # of species N sparticles
c      RPPSP(n)....real particles per sparticle (=TOTNRP/NPINIT(N))
...
c      Now Initialize Particles for each species
mpb=mpbuf
call setb(nsp,mpb,mspec,nspec,nbspec,memoly,dimp,nheadb)
c Call to SETB sets up i/o buffering of the particles where...MSPEC
c contains the MAX # of each.prtcl type. (See Appndx 8 on buffering)
c
c      header (PBUF,IPBUF) layout
c      PBUFH(1) = buffer number for this buffer
c          2 = # partcls in this buffer (called NPTHISB)
c          3 = ultimate # prtcl blocks/species--also in NBSPEC(n)
c          4 = species number = n
c          5 = species name = namsp(n) = namesp
c          6 = time step number = it
c          8 = species charge = sq(n) (pchrg=sq(n)/rppsp(n)=sqsgnl(n))
c          9 = species mass = sm(n) (pmass=sm(n)/rppsp(n)=smsngl(n))

```

III.A.4.b) Particle Initialization using Particles/Cell NPARTK

```

dimension npartk(nr2,nz2)
equivalence (npartk(1,1),rho(1,1))
...
```

```

c      now define code parameters and initialize particles
c
      issav=0      numbuf=0
500 n=-2
      call getb(n,npthisb,iparmem,ipbuf,nbspec)
      if (n.lt.0) go to 800
      if (n.ne.issav) then
c Calls to GETB write into memory or disk files the present buffer-
c consisting of NHEADB buffer-describing header words and NDIMP
c floating point numbers that characterise each of the NPTHISB
c prtcls in this buffer. Generally GETB writes present buffer &
c reads the next. NPTHISB must be consistent with the present
c buffer; it will be reset for the new buffer automatically. N
c contains the new species # on return. If all the buffers of the
c present species have been processed, N is incremented and the
c first buffer of the next species is read into the local memory
c IPARMEM. If all the buffers for the last species have been
c processed, N=-1 & control is returned to the calling routine.
c (See Appendix 8 on buffering)

```

The section immediately below integrates the cell center DEN at the start of each species N to find the number of real particles that the entire simulation contains. (In DEN is zero everywhere but particle emission is expected, a uniform fill of the simulation region is assumed.) Next the array NPARTK is filled with integers that represent the number of simulation particles that must be created in each cell in order for that cell to have the expected real density. It is natural here to define for each species the quantity RPPSP(N) (representing Real Particles Per Simulation Particle). Now the quantities SQ(N) and SM(N), representing the real charge and mass per simulation particle can be determined.

```

c      Start new species      (remember DEN is cell centered)
bamp=0.
do 545 nstr=1,nstrmx
545 bamp=bamp+beam(nstr,n,1)
totnrp=0.          refnrp=0.
do 550 i=2,nr1      do 550 j=2,nz1
skyf=1.+skey(i,j)*.5*(1.-skey(i,j))
totnrp=totnrp+den(i,j,n)*volcc(i)*skyf
550 refnrp=refnrp+bamp*volcc(i)*skyf
refnrp=refnrp+totnrp          totnrp=amax1(totnrp,1.)
DEFISH=0.
ntot=0
do 560 j=2,nz1      do 560 i=2,nr1
skyf=1.+skey(i,j)*.5*(1.-skey(i,j))

```

```

      partk=nspec(n)*skyf*den(i,j,n)*volcc(i)/totnrp+DEFISH
      npartk(i,j)=partk
      DEFISH=partk-npartk(i,j)
560   ntot=ntot+npartk(i,j)
      write (59/7,563) nspec(n),ntot,namsp(n),n
563   format(i9,' sparticles rqstd',i9,a8,' spartcls cr8d. n=',i5)
      nspec(n)=ntot    npinit(n)=ntot    rppsp(n)=refnrp/mspec(n)
      if (denmx(n).gt.0.) rppsp(n)=totnrp/nspec(n)
      qvm(n)=sqsgl(n)/smsgl(n)
      sq(n)=rppsp(n)*sqsgl(n)           sm(n)=rppsp(n)*smsgl(n)
      i=1          j=2          npartt=-1
      lpstc=0      npthisb=1+(npinit(n)-1)/nbspec(n)      issav=n
      endif
c
      npthisb=min0(npinit(n)-lpstc,npthisb)      numbuf=numbuf+1
      ipbuf(1)=numbuf    ipbuf(2)=npthisb    ipbuf(3)=nbspec(n)
      ipbuf(4)=n        ipbuf(5)=namsp(n)    ipbuf(6)=0
      pbufh(8)=sq(n)    pbufh(9)=sm(n)
      if (npthisb.eq.0) go to 500
c

```

The next section describes the random creation within each cell the number of particles NPARTK(I,J) needed to represent the desired density DEN(I,J,N). As they are created, they are given random thermal velocities consistent with the temperature TEM(I,J,N), and anisotropy VR/Q/ZVT(N), and have a drift velocity UR/Q/Z(I,J,N) added.

```

      lpstc=lpstc+npthisb
      do 700 lp=1,npthisb*dimp,dimp
610 if (npartt.gt.0) go to 650
      i=i+1
      if (i.le.nr1) go to 640
      i=2      j=j+1
      if (j.le.nz1) go to 640
      stop 'prts>cls'
640 npartt=npartk(i,j)
      vmp=sqrt((boltzk+boltzk)*tem(i,j,n)/smsgl(n))
      go to 610
650 npartt=npartt-1
      r(lp)=rgcc(i)+dr*(ranf()-.5)*dithrl(1,n)
      z(lp)=zgcc(j)+dz*(ranf()-.5)*dithrl(2,n)
      vmag=vmp*sqrt(-alog(1.-.999999*ranf()))
      theta=pi2*ranf()
      vx(lp)=vmag*vrvt(n)*cos(theta)+ur(i,j,n)
      vz(lp)=vmag*vzvt(n)*sin(theta)+uz(i,j,n)

```

```
    vmag=vmp*sqrt(-alog(1.-.999999*ranf()))
    theta=pi2*ranf()
700 vy(lp)=vmag*vqvt(n)*cos(theta)+uq(i,j,n)
      go to 500
800 continue
```

c

Finally, INIT sets the time step DT to zero and runs a complete time step just to fully initialize all the fields. A further benefit is that all the diagnostics run a TIME=0 so you *really* know and can prove the initial state of the simulation. (Important point, read between the lines here!)

```
dtsav=dt      dt=0.
call trans      call dsouce      (Cycles through time
call field      call dfield      step with DT=0)
dt=dtsav      return
end
```

IV. The Central Time Integration Loop

The main program MAIN directs the action of the entire code. MAIN contains not only the central loop with periodic calls to diagnostics but also the calls to the initialization/RESTART routines as well as the procedures to start and clean up at the end of the run. These things will be discussed in due course; for now look at the central time integration loop-found about halfway through the MAIN routine.

```
...
it=0
time=0.
call init
endif
ihist=min0(ihist,ithmx)

c
c          TIME ADVANCE
100 it=it+1      time=time+dt
    if (time.gt.(tmax+dth)) go to 200
    if (nrest.ne.0) then
        if (mod(it,iabs(nrest)).eq.0.or.
            time.ge.(.99999999*tmax)) mkrest=1
        endif
    call trans      Moves all species thru a time step.
                    Generates sources for each species
                    as well as total plasma sources.
        if (mod(it,icvplt).eq.0) call dsouce
    call field
    if (mod(it,icvplt).eq.0) call dfield
    if (mod(it,ihist).eq.0 .or. time.ge.(tmax-dth)) then
        call history
        if (mod(it,needf3).eq.0.and.time.lt.(tmax-dth)) call newf3
    endif
c          Writing a restart?
if (mkrest.eq.1) then
    call wstart(....)
.
endif
c          Check for teletype interrupts.
if (imsg.eq.1) call msgproc(...)
if (imsg.eq.2) go to 200
go to 100
200 call endplt
stop 777
end
```

First note the loop on IT and TIME between 100 and 200. (I use all capitals to distinguish FORTRAN variables within the text). This is the time integration loop: IT is the time step count and TIME is the time in units of seconds. Control stays within these lines until TIME exceeds TMAX. The first call is to subroutine TRANS which integrates or "TRANSports" all plasma quantities (particles positions and velocities, plasma source terms) forward in time through one time step. The details of this process are discussed in later section B). For now it is only important to remember that TRANS is called with particle positions and velocities and E and B fields at time level TIME_n and it returns with the positions and velocities of the ions advanced by DT to TIME_{n+1}. Also returned are the source terms for the number density DEN, mean flow velocity in the three directions UR, UQ, UZ, and the temperature TEM. [Throughout this write up and GYMNOs, Q or q is used to denote theta.] TRANS deals with any and all particle buffering and associated gymnastics to accomplish this time advance. TRANS also calls the diagnostics for each particle buffer as well as diagnostics after each species is processed. Next is a call to DSOURCE that plots and otherwise diagnoses source totals over species.

FIELD computes the potentials A and Φ given the instantaneous particle source terms. A call is made to DFIELD which provides diagnostics, mainly graphical, for the new information available after the field integration. Likewise, a call to HISTORY, if it is time, call to plot selected time histories that may be stored by TRANS as the time integration progresses. We also check to see if it is time to close a plot file and open a new one by comparing the NEEDF3 parameter with the time step. Finally we check if we need to write a restart file at this time step and we check very time step for information from the terminal. When TIME = TMAX, we flush the plot buffers with ENDPLOT and terminate with STOP 777, our normal termination indicator.

IV.A) TRANSport of the particles from TIME to Time+DT

The purpose of TRANS is to coordinate the cycling of particles through a particle pusher. The general strategy is to bring in a buffer of particle data and, if it is a new species type, "finish" the previous species and "start" the new species. Slight adjustments are made not to finish a species if we're just starting a time step nor to start a species if the buffer we have just finished pushing is the last of the species to be processed this time step.

```

subroutine trans
...
c      D3 is the velocity space dimensionality
c      define      Vth=sqrt(boltzk TEM/smsngl(n) )
c                  Vmp=sqrt(2 boltzk TEM/smsngl(n) )
c                  Pressure=D3 DEN boltzk TEM
D3=3.
if (nr.eq.1 .or. nz.eq.1) D3=1.
...
c      1st, is number to (field) emit is consistent with max allowed?
c      Next ready plasma source terms for next IT accumulation.

```

```

call getb(n,npthisb,iparmem,ipbuf,nbspec) {Get 1st prtcl buffer}
go to 400                                {Skip to start species}

c      Top of particle species "loop"
100 call getb(js,npthisb,iparmem,ipbuf,nbspec)
    if (js.lt.0) go to 200 {Last buffer processed, finish species.}
    if (js.eq.n) go to 500 {Species unchanged, go push buffer.}

c

```

IV.A.1) Start New Species, AUTOVLM, DPARTS, Call PBCSET (See V. BCSET), Preset E and B

```

c                      Start a new species
400 npthisb=ipbuf(2)      n=ipbuf(4) .....

if (mod(it,ipplt).eq.0) call dparts
c                           {Diagnostics PARTicle Start}

AUTOVLM here (see appendix) Prepare VLIM's to find velocity extremes
for prtcl plots NEXT time step.

call pbcset(Jphi)          {Set particle BCs for this species N}
c      Zero accumlating arrays, fix E, B with species dependence
hqvm=.5*qvm(n)
hqvmc=hqvm/C
er/q/z(i,j)=er/q/z(i,j)*hqvm
br/q/z(i,j)=br/q/z(i,j)*hqvmc

500 n=ipbuf(4)
c           now push the blk of particles
    call parmov
AUTOVLM (see apdx 7) Find max/min velocities IF Pploting next IT
    Call Diagnostic PARTicle DPART if Pploting NOW
    if (mod(it,ipplt).eq.0) call dpart
    go to 100

700 continue

c      DAPART provides a diagnostics opportunity on particle quantities.
c      Here we use the extra pass to get R-Z space plots of particles.
    if (mod(it,ipplt).eq.0) call dapart {Diagnostics After PARTicles}
    return
    end

```

The point is simply that, with this storage arrangement, all quantities associated with the l^{th} particle may be accessed with the subscript LP. Don't be concerned about the use of V_x for V_r or V_y for V_q —the explanation is found and discussed in the particle pusher section PARMOV.

NPTHISB and N (species number) are generated in GETB; They are carried in the header words IPBUF(2) and (4), respectively. They are appropriately set by GETB which expects the particle buffers, will contain only one type of particles (though adjacent similar particles may be plotted on the same plots), will be contiguous for each species N. The change in species type is signaled by a change in JS that GETB gets by reading the fourth header word IPBUF(4). The end of the particle buffers of the last species is denoted by $JS < 0$.

The call to DPART is used to produce phase-space plots as each buffer load of particles is processed.

IV.A.2) Call PARMOVE-The Particle Time Integration

IV.A.2.a) Main Boris Push, Call TRAP, First Order Advance for Advanced Velocity Moments

The particle time advance is accomplished by a Boris Pusher. It is reversible and second order in time. A derivation follows the Fortran.

The essential concepts are that the particles are all pushed as if they were on a Cartesian X-Y mesh in the R-Theta plane. In axisymmetry all particles are considered to be rings of volume R^*DR^*DZ . At the beginning of each time step, a particle's position is considered to be at $X=R$ and $Y=0$ with $V_x=V_r$ and $V_y=V_q$. V_x and V_y at time $t_{n+1/2}$ are computed using $E_x=E_r$ and $E_y=E_q$. [Remember Q or q is used to denote θ .] The particle is advanced to a new X and Y with these velocities. This is accomplished by the FORTRAN

```
subroutine parmov
```

```
c-----hewett, 1986
c      rotate=2.(0.) gives(stops) particle rotation
c      rotate=2.

c
c      Assume we start here with   r(n),vx(n-1/2)
c      The fields E and B are at the (n) time level.
c

do 70 lp=1,npthisb*dimp,dimp
rr=dri*(r(lp)-rmin)+1.          Note the simple interpolation
rz=dzi*(z(lp)-zmin)+1.          scheme for uniform mesh spacing
i=rr                            DR,DZ.  FR, FRC, FZ FZC are used
fr=rr-i                          to make the fractional parts to
frc=1.-fr                         extract from each of the nearest
```

```

j=rz                                4 cells to the particle location.
fz=rz-j
w1=frc*fzc
w2=fz*frc
w3=fr*fzc
w4=fr*fz
These are the interpolation
factors determining the contri-
bution from each of the cells to
the field at the particle location.

c      do half of E push
eacclr/q/z=dt*(w1*er/q/z(i,j)+w2*er/q/z(i,j+1)
               +w3*er/q/z(i+1,j)+w4*er/q/z(i+1,j+1))
vx/y/z(lp)=vx/y/z(lp)+eacclr/q/z
c      rotation follows
omgr/q/z=dt*(w1*br/q/z(i,j)+w2*br/q/z(i,j+1)
               +w3*br/q/z(i+1,j)+w4*br/q/z(i+1,j+1))
vx/y/zt=vx/y/z(lp)+vy/z/x(lp)*omgz/r/q-vz/x/y(lp)*omgq/z/r
fact=rotate/(1.+omgr*omgr+omgq*omgq+omgz*omgz)
vx/y/z(lp)=vx/y/z(lp)+fact*(vy/z/xt*omgz/r/q-vz/x/yt*omgq/z/r)
c      rotation finished
c      do remaining half of E push
vxh=vx(lp)+eacclr    vyh=vy(lp)+eacclq   vz(lp)=vz(lp)+eacclz
c      now advance to r(n+1) in order to find vx(n+1/2) 4/3/90
x=r(lp)+vxh*dt          dyyy=vyh*dt
z(lp)=z(lp)+vz(lp)*dt    r(lp)=sqrt(x*x+dyyy*dyyy)
costh=x/r(lp)            sinh=dyyy/r(lp)
vx(lp)=costh*vxh+sinh*vyh  vy(lp)=-sinh*vxh+costh*vyh
70 continue
c      push completed, now TRAP LOST particles
call trap

c
c      After TRAP extrapolate vx(n+1/2) to vx(n+1), using new
c      positions r(n+1) and fields E(n+1/2) and B(n+1/2), to
c      accumulate "raw count" sources at the desired time level.

c
do 270 lp=1,npthisb*dimp,dimp
rr=dri*(r(lp)-rmin)+1.      rz=dzi*(z(lp)-zmin)+1.
i=rr      fr=rr-i      frc=1.-fr
j=rz      fz=rz-j      fz=1.-fz
w1=frc*fzc      w2=fz*frc      w3=fr*fzc      w4=fr*fz
c      do half of E push with E(n+1/2)/2
eacclr/q/z=dth*(w1*er/q/z(i,j)  +w2*er/q/z(i,j+1)
               +w3*er/q/z(i+1,j)+w4*er/q/z(i+1,j+1))
vxh=vx(lp)+eacclr    vyh=vy(lp)+eacclq   vzh=vz(lp)+eacclz
c      rotation follows using B(n+1/2)/2 and r(n+1) location
omgr/q/z=dth*(w1*br/q/z(i,j)  +w2*br/q/z(i,j+1)
               +w3*br/q/z(i+1,j)+w4*br/q/z(i+1,j+1))

```

```

vx/y/zt=vx/y/zh+vy/z/xh*omgz/r/q-vz/x/yh*omgq/z/r
fact=rotate/(1.+omgr*omgr+omgq*omgq+omgz*omgz)
vx/y/zh=vx/y/zh+fact*(vy/z/xt*omgz/r/q-vz/x/yt*omgq/z/r)
c           rotation finished
c       do remaining half of E push using E(n+1/2)/2
vxnp1=vxh+eacclr  vynp1=vyh+eacclq  vznp1=vzh+eacclz
c       ACCUMULATE PLASMA SOURCE contribution from each particle
den(i,j,n)=den(i,j,n)+w1
den(i+1,j,n)=den(i+1,j,n)+w3
den(i,j+1,n)=den(i,j+1,n)+w2
den(i+1,j+1,n)=den(i+1,j+1,n)+w4
ur/q/z(i,j,n)=ur/q/z(i,j,n)+w1*vx/y/znp1
...
tloc=vxnp1*vxnp1+vynp1*vynp1+vznp1*vznp1
rhs(i,j)=rhs(i,j)+w1*tloc
...
270 eionr/q/z=eionr/q/z+vx/y/znp1*vx/y/znp1
      return
      end

```

The formulas can be obtained by first writing the desired scheme

$$V_{n+.5} = V_{n-.5} + \Delta t(E_n + \frac{V_{n+.5} + V_{n-.5}}{2} \times B_n)$$

where the charge, mass, and C have been incorporated into the fields as appropriate. We may choose to decompose this expression into the following three parts:

$$V_1 = V_{n-.5} + \frac{\Delta t}{2} E_n \quad (1)$$

$$V_2 = V_1 + (V_2 + V_1) \times \Omega \quad (2)$$

$$V_{n+.5} = V_2 + \frac{\Delta t}{2} E_n \quad (3)$$

where $\Omega = \Delta t B_n / 2$. The clever idea in the Boris pusher is that the middle expression that describes a time-centered implicit rotation can be solved without resorting to a 3X3 linear system for V_2 . To see this, rewrite the middle equation as

$$V_2 - V_2 \times \Omega = V_1 + V_1 \times \Omega \quad (4)$$

and let both sides of the equation equal yet another variable V' . We next add each of these equations together, yielding

$$V_2 - V_2 \times \Omega + V_1 + V_1 \times \Omega = 2V' \quad (5)$$

Rearranging, we have

$$V_2 + V_1 - (V_2 - V_1) \times \Omega = 2V' \quad (6)$$

Taking the cross product from the right with Ω , gives

$$(V_2 + V_1) \times \Omega - [(V_2 - V_1) \times \Omega] \times \Omega = 2V' \times \Omega \quad (7)$$

The first term on the left reduces to $(V_2 - V_1)$ using eqn. (2). The second term reduces to

$$[\Omega \cdot (V_2 - V_1)]\Omega - \Omega^2(V_2 - V_1).$$

We know, again from eqn. (2), that Ω is perpendicular to $(V_2 - V_1)$ so the first term in this expression vanishes. Eqn. 7 then reduces to

$$(1 + \Omega^2)(V_2 - V_1) = 2V' \times \Omega. \quad (8)$$

The net result of this manipulation is that eqn. (2) that describes particle rotation can be replaced by the two equations

$$V' = V_1 + V_1 \times \Omega \quad (9a)$$

and

$$V_2 = V_1 + \frac{2}{(1 + \Omega^2)}V' \times \Omega \quad (9b)$$

The net result is that we may move the particles with time-centered implicit pushes using eqns. (1), (9a), (9b), and (3). If care is taken with precisely what one uses for B_n , it is possible to obtain a pusher that will provide the correct cyclotron frequency *regardless* of DT. We do not keep the extra terms (of order DT^3) but details can be found in Hockney and Eastwood or Birdsall and Langdon.

The transformation of V_x and V_y into V_r and V_q is accomplished by

```

r(1)=sqrt(x*x+dy*dy)
costh=x/r(1)  sinth=dy/r(1)      THROUGHOUT GYMNO$,
vx(1)=costh*vxh+sinth*vyh      for VR and VY for VQ.
vy(1)=-sinth*vxh+costh*vyh

```

where the generation of the new R from the new X and Y is obvious. [We find it convenient to use the variable V_x for V_r and V_y for V_q for ease of storage although on occasion in PARMOV, they actually contain the Cartesian components.] With axisymmetry in 2.5D, no loss of generality is experienced if the particle is assumed to be back at $\theta=0$ for the start of the next time step—so long as the correct value for V_q is maintained.

The remaining physics part of PARMOV is a straightforward reapplication of these steps for the first-order extrapolation of velocities at time $t_{n+1/2}$ (that will be stored) to time level $n+1$ (which will not be stored) for use in the “raw count” source accumulation.

Notice that PARMOV runs one particle at a time through this loop for the entire buffer load set up by TRANS. No thought is given, in this version, to vectorization or optimization (see sections discussing VPARMOV). For the new user of this basic code, loss of readability and flexibility cause more wasted CPU cycles than are usually recouped using clever coding.

We have accumulated “raw counts” in each cell. Returning to TRANS after all particle buffers have been processed, we divide each particle source term by the volume associated with that cell radius. The density DEN, for example, returned to TRANS from PARMOV contains a simulation particle count in each cell. When a species is finished, DEN(I,J,N) will be divided by the volume of the computational cell EFFVOL that turns DEN into a number density of simulation particles in each cell and is ultimately converted to a real plasma density by multiplying by the number of real particles/simulation particle RPPSP. (see discussion of TRANS, Sec. III.A.4.a)

IV.A.2.b) TRAP of the Particles Outside Plasma Region

IV.A.2.b.1) Processing LOST PARTICLES

At the start of a time step, we know all particles are in legitimate simulation regions. Particles are moved to their n+1 location in the top of PARMOV. The call to TRAP after this time step checks each of these new positions to see if they have moved into a prohibited region. TRAP accomplishes this by looking at the cell *center* indices the particle now has

```

c                               Detect lost prtcls.
noutsde=0      npstart=npthisb
npart=0        lp=1-dimp
120 lp=lp+dimp   multitrp=0
130 if (lp.gt.lplast) go to 200
     icen=dri*(r(lp)-rmin)+2.          jcen=dzi*(z(lp)-zmin)+2.
     icen=min0(max0(icen,1),nr2)       jcen=min0(max0(jcen,1),nz2)
     if (skey(icen,jcen).gt.-1.) go to 120
c                           Have found one!

```

and looking at the value of SKEY with those same indices. SKEY(ICEN,JCEN) will be -1. if the volume represented by these indices is *inside* a structure. Even particles that move well outside the simulation region will have ICEN,JCEN moved back to the mesh edges via the MIN0,MAX0 functions and these boundaries are always structures of some kind. (Unless the boundary is periodic—see details in the discussion of structures in GEOSET in subroutine BCSET.)

IV.A.2.b.1.a) Determining Closest Corner

When we find a particle “outside”, we next determine its CFL number (the number of cells through which it is now moving per Δt) in each direction. We then cycle back with smaller time steps DTCLS (such that $CFL < 1$) if necessary to find where last particle/boundary crossing occurred. TRAP may use multiple passes through this procedure to catch those whose reflection puts them in yet another structure. The coding looks like this.

```

c                           Have found one!
pcflr=abs(dri*vx(lp)*dt)

```

```

pcflz=abs(dzi*vz(lp)*dt)
ib=1.+amax1(pcflr,pcflz)
if (ib.ge.mxg1d .or. multitrp.gt.mxg1d) then
c      2 fast or # of tries MULTITRP 2 big....delete
      npart=npart+1
      .....delete particle.....
      go to 130
      endif
c      Find close corner at indices  ICLS,JCLS.
c Cycles back with smaller CFLok DTCLS to find where last particle
c crossing occurred.  Uses multiple passes to catch those whose
c reflection puts them in yet another structure.    DWH 8/31/90
      dtcls=1.000001*dt
      rrun=r(lp)           zrun=z(lp)
      if (ib.gt.1) dtcls=dt/ib
      dthvx=.5*dtcls*vx(lp)      dthvz=.5*dtcls*vz(lp)
73   rrun=rrun-dthvx-dthvx      zrun=zrun-dthvz-dthvz
      if (iperodz.eq.1) zrun=zmin+amod(zrange+zrun-zmin,zrange)
      ib=ib-1
      if (ib.lt.-2) then
c          ib<-2....delete
          noutsde=noutsde+1
          npart=npart+1      r(lp)=r(lplast)      z(lp)=z(lplast)
          vx(lp)=vx(lplast)  vy(lp)=vy(lplast)      vz(lp)=vz(lplast)
          lplast=lplast-dimp
          print 183, ib,icls,jcls,r(lp),z(lp),pcflr,pcflz
183   format(' DLTib<-2 ib,i-jcls,r,z,cflr-z',3i3,1x,4(1pe9.2))
          go to 130
          endif
          icen=dri*(rrun-rmin)+2.
          jcen=dzi*(zrun-zmin)+2.
          icls=dri*((rrun+dthvx)-rmin)+1.5
          jcls=dzi*((zrun+dthvz)-zmin)+1.5
          if (icls.lt.1 .or. icls.gt.nr1 .or. jcls.lt.1 .or. jcls.gt.nz1
              .or. skey(icen,jcen).eq.-1.) go to 73
          ..

```

The purpose of this procedure is to find the indices of the *corner* closest to the place where the particle crossed into a forbidden structure. Armed with the indices of the "closest corner" ICLS,JCLS, the code then interrogates the array IPBC to determine what to do with the offending particle.

IV.A.2.b.1.b) Perfect Reflection or Delete

Since the particles outside the periodic dimensions have already been returned, it is the

physics properties of the closest corner determine which of two remaining courses of action will be followed. 1) If the closest corner is a point on a perfectly reflecting surface (denoted by IPBC(ICLS,JCLS)=51), the particle is perfectly reflected. If not then 2) the particle is logged and deleted by overwriting this particle with the last particle in the current buffer. All processes other than perfect reflection involve absorption and reemission.

Perfect reflection involves first determining the "orientation" of the surface associated with the closest corner ICLS,JCLS. The orientation is found in the array KORNT that has been set in routine GEOMSET using SKEY data. The best documentation is already contained in TRAP comment cards that are reproduced here.

```

ok PERFECT REFLECTION-orient the surface with KORNT
c
c
R-reflect if
  (R(LP)-RG(ICLS))*VX >=0
    and
    KORNT*(|KORNT|-10)*VX <0
      or
      |KORNT| = 1

Z-reflect if
  (Z(LP)-ZG(JCLS))*VZ >=0
    and
    KORNT*(|KORNT|- 1)*VZ <0
      or
      |KORNT| = 10

Besides the obvious conditions of |KORNT|=1 or 10 for the plain
sides, we use two types of conditions for reflection. The first
depends of a product of velocity and position as

(R(LP)-RG(ICLS))>0 so that VR>0 r-reflects
<0 so that VR<0 r-reflects
(Z(LP)-ZG(JCLS))>0 so that VZ<0 z-reflects
<0 so that VZ>0 z-reflects

The second condition depends on corner orientation
  KORNT*(|KORNT|-10)<0 for 8,9,-11,-12 so only VR>0 r-reflects
                >0 for 11,12,-8,-9 so only VR<0 r-reflects
  KORNT*(|KORNT|- 1)<0 for -11,-12,-8,-9 so only VZ>0 z-reflects
                >0 for 11,12,8,9 so only VZ<0 z-reflects

Either the velocity-position or the KORNT conditions will suffice
for outside corners because troublesome locations are not "inside"
boundaries. To do the inside corner correctly for all cases
requires both conditions to be satisfied. For example, we want no
z-reflection from inside +9 corners for particles in upperright
quadrant, when incident from upperleft with vr>0 and vz>0. Without
the KORNT condition it will z-reflect. Similarly, we want no
z-reflection from inside +9 corners for particles in upperright
quadrant when incident from upperleft with vr>0 and vz<0. In this
case KORNT will reflect but x-v condition will not.

flect2=0.
ikornt=iabs(kornt(icls,jcls))

```

```

if (ikornt.eq. 1 .or. ((r(lp)-rg(icls))*vx(lp).ge.0. .and.
.   kornt(icls,jcls)*(ikornt-10)*vx(lp).lt.0)) then
  r(lp)=2.*rg(icls)-r(lp)
  vx(lp)=-vx(lp)
  flect2=tvdrz2
endif
if (ikornt.eq.10 .or. ((z(lp)-zg(jcls))*vz(lp).ge.0. .and.
.   kornt(icls,jcls)*(ikornt- 1)*vz(lp).lt.0)) then
  z(lp)=2.*zg(jcls)-z(lp)
  vz(lp)=-vz(lp)
c  Have we already done an R-reflection? If so fix outgoing
c  by reversing V component perpendicular to slant normal.
  sdz=(10-ikornt)*dz
  svr =vx(lp)-flect2* dr*(vx(lp)*dr+vz(lp)*sdz)
  vz(lp)=vz(lp)-flect2*sdz*(vx(lp)*dr+vz(lp)*sdz)
  vx(lp)=svr
endif                               We are now ready to reprocess this
multitrp=multitrp+1                  particle insure that a reflection
go to 130                            of a very fast particle did not result
                                         in the new position being outside.

```

200 ...

After some bookkeeping to account for all the particles lost, we are ready to consider thermal reemission or, if there is room in this buffer, injection or field emission.

IV.A.2.b.2) Thermal Reemission

If any particles have been deleted in previous operation in TRAP, we will have kept track of the total charge absorbed in each corner ICLS,JCLS in array QABSRB. (If an closest corner was unable to be identified for some reason, the particle's charge is simply lost.)

IV.A.2.b.2.a) Controls from PBCSET

```

c  QABSRB(ICOR,JCOR) is the charge absorbed by cell corner ICOR,JCOR
c  Thermal reemission uses QABSRB as the reservoir. Reemit
c  .02*|IPBC|*QABSRB/SQ using #s in NSTPROP.
c  -----
c               Now Thermal REEMISSION from cell corners
c  Temperature comes from TEM(ICOR,JCOR,N)
c  (new "TEM" accumulated in RHS array until
c  species finished in TRANS)
c  Normal velocity = 0 (use INJECTION if UNBSTR!=0)

```

The controls for thermal reemission are set in PBCSET by a call before processing on the present species was begun. The primary control array for all structures is NST-PROP(NSTR,N). This array is used to determine IPBC(ICOR,JCOR) for each corner from the definition in PBCSET

```

nbtyp=iabs(nstprop(nstr,n))/100
nbpct=iabs(nstprop(nstr,n))-100*nbtyp
690 ipbc(kcij(kcor,1),kcij(kcor,2))=nbpct

```

and was discussed in section A2. Note that no normal velocity is allowed.

IV.A.2.b.2.b) Mechanics

TRAP begins the reemission process by cycling through the structures determine, first, if the structure allows reemission from the structure in question. If reemission is allowed, then the corner points of that structure are considered to see how many, if any, particles were absorbed at that corner and, finally, what fraction of those absorbed are allowed to be reemitted. If a corner passes all these tests, particles may then be given birth at this corner with all the additional specifications given by BEAM and NSTRPROP. The code in question follows

```

if (ireemt.eq.0) go to 505
      delr=dr/drz           delz=dz/drz

      do 501 nstr=1,nstrmx
      nbpct=mod(iabs(nstprop(nstr,n)),100)
      if (nbpct.le.0 .or. nbpct.gt.50) go to 501
c      The transverse temperature of the emitted particles are
c      always Vth(from TEMPSTR). The normal component of the velocity
c      is determined by BEAM(nstr,n,4)=Tnormal/Tperp sugg by DJL 12/12/90
      evtvth=1.
      evnvth=sqrt(bean(nstr,n,4))
      unorm=0.
      if (nstprop(nstr,n).lt.0) then
          edithr=0           edithz=0
      else
          edithr=dr           edithz=dz
      endif
c      (surface area of cone pi*r*l      use l=r*drz/dr)
c      do 500 jcor=1,nz1    (Don't need every corner...dwh 3/29/91)
c      do 500 icor=1,nr1
c          ikornt=iabs(kornt(icor,jcor))
c          if (ikornt.eq.0) go to 500
      do 500 kcor=nstruct(nstr,1),nstruct(nstr,2)
          icor=kcij(nstr,1)

```

```

jcor=kcij(nstr,2)
ikornt=iabs(kornt(icor,jcor))
jpbc=ipbc(icor,jcor)      japbc=iabs(jpbc)
if (japbc.gt.50) go to 500
c           Determine Normal Direction.
sgn=isign(1,kornt(icor,jcor))
if (ikornt.eq.1) then
  sgnr=sgn                  sgnz=0.
  endif
if (ikornt.eq.10) then
  sgnr=0.                   sgnz=sgn
  endif
if (ikornt.eq.11 .or. ikornt.eq.12) then
  sgnr=delr*sgn            sgnz=delz*sgn
  endif
if (ikornt.eq.9 .or. ikornt.eq.8) then
  sgnr=-delr*sgn          sgnz+=delz*sgn
  endif
c           return a fraction of the outside prtcls.
npart=.02*float(japbc)*(1.e-7 + qabsrb(icor,jcor)/sq(n))
if (npart.le.0) go to 500
c           unorm=unbstr(icor,jcor)
vmp=sqrt((boltzk+boltzk)*tem(icor,jcor,n)/smsngl(n))
call birth(npart,vmp,unorm,evnvth,evtvth,kornt(icor,jcor),
           edithr,edithz,sgnr,sgnz,rg(icor),zg(jcor))
qabsrb(icor,jcor)=qabsrb(icor,jcor)-npart*sq(n)
500 continue
501 continue

```

BIRTH obviously creates NPART particles with the other parameters in the argument list. Details are found in a appendix.

IV.A.2.b.3) Injection and Field Emission

In many ways similar to Thermal Reemission, the injection/filed emission section differs in that 1) absorption need not have taken place so that is reason to create new particles. 2) All that matters is that BEAM has a finite density to be emitted, that it is time for it to be "turned on" and that there is a structure that allows that process. Other significant differences are that non zero normal velocities are allowed, and that particles are created from cell faces [Left OR Down, LOR_D, means the face to the left of (or down from) the corner KCOR if LOR_D=1(2)].

IV.A.2.b.3.a) Controls from PBCSET

c QBRS(KCOR,LorD,N) is the resevoir for field emission and injection.

```

c      now FIELD EMIT or NJCT NEW prtcls
c      NSTRUCT(NSTR,1)--name of structure numbered NSTR
c      NSTRUCT(NSTR,2/3)--KCOR start/stop pt in KCIJ.
c      IorJCOR = KCIJ(KCOR,1or2) for structure point KCOR
c      ex: do # kcor=nstruct(nstr,2),nstruct(nstr,3)
c            # key(kcij(kcor,1),kcij(kcor,2))=-1
c            KCORR(icor,jcor)=KCOR  (cross reference array)
c            QSTRU(NSTR,N)--charge of species N lost in NSTR last DT
c            NSTPROP(NSTR,Nspecies)--n00-n50, n51--%ThRemt,PRFLCT
c            (n=1/2/0 Femt/Njct/Neither), negative=no dithr
c            QBRS(KCOR,LorD,Nspecies)... "Reservoir" for emitting/injecting
c                  LorD=1(2) [emitting surface Left(Down) from corner]
c            UNBSTR(kcor,lord,n) = Normal FEMT/NJCT velocity magnitude.
c            TEMSTR(kcor,lord,n) = Thermal spread about injection velocity.
c
c -----

```

IV.A.2.b.3.b) Mechanics

```

505 if (mpb.le.npthisb .or. dt.le.0.) return
      if (npnjct(n).eq.0 .and. npfemt(n).eq.0) return
c          NJECT and FLDEMT after advance before accumulation
      Nnput=0
      do 910 nstr=1,nstrmx
          nbtyp=iabs(nstprop(nstr,n))/100
          if (nbtyp.eq.0) go to 910
          nject=nbtyp-1
          if (nstprop(nstr,n).lt.0) then
              edithr=0.           edithz=0.
              else
                  edithr=dr       edithz=dz
              endif
c      The transverse temperature of the emitted particles are
c      always vth(from TEMPSTR). The normal component of the velocity
c      is determined by BEAM(nstr,n,4)=Tnormal/Tperp sugg by DJL 12/12/90
          evtvth=1.
          evnvth=sqrt(bean(nstr,n,4))
          do 900 kcor=nstruct(nstr,2),nstruct(nstr,3)
c      new stuff 3/20/90 dwh
          if (qbrs(kcor,1,n).eq.0. .and. qbrs(kcor,2,n).eq.0.) go to 900
              i=kcij(kcor,1)      j=kcij(kcor,2)
              if ((i.eq.1 .and. j.eq.1) .or. kornt(i,j).eq.0 .or.
                  kornt(i,j).eq.+11 .or. kornt(i,j).eq.-12) go to 900

```

```

knz=kornt(i,j)/8      knr=kornt(i,j)-knz*10
ikornt=iabs(kornt(i,j))
ldstar=1      ldstop=1
if (j.eq.1 .or. ikornt.eq.10 .or.
x     kornt(i,j).eq.+9 .or. kornt(i,j).eq.-8) go to 880
ldstop=2
if (kornt(i,j).eq.+12 .or. kornt(i,j).eq.-11) go to 880
ldstar=2
880 do 890 lord=ldstar,ldstop
c               ire\ize=1 means emit horizontally\vertically
ire=lord-1      ize=2-lord
npart=qbrs(kcor,lord,n)/sq(n)
if (npart.le.0) go to 890
remt=ire*rg(i)+ize*rgcc(i)           zemt=ire*zgcc(j)+ize*zg(j)
sgnr=isign(ire,knr)                 sgnz=isign(ize,knz)
unorm=unbstr(kcor,lord,n)
c JK Boyd's trick-start cell/4 in to avoid nfnit DEN ChildL FEMT
remt=remt+(1-nject)*sgnr*.5*drh
zemt=zemt+(1-nject)*sgnz*.5*dzh
c               Prepare for PRTCL EMISSION
npart=min0(npart,(mpb-npthisb))
npart=min0(npart,(mspec(n)-nspec(n)))
if (npart.le.0) return
vmp=sqrt((boltzk+boltzk)*temstr(kcor,lord,n)/smsngl(n))
call birth(npart,vmp,unorm,evnvth,evtvth,0,
           edithr,edithz,sgnr,sgnz,remt,zemt)
npfemt(n)=npfemt(n)-npart*(1-nject)
nnpjct(n)=nnpjct(n)-npart*nject
Nnput=Nnput-npart
qbrs(kcor,lord,n)=qbrs(kcor,lord,n)-npart*sq(n)
qstruc(nstr,n)=qstruc(nstr,n)-npart*sq(n)
if (mpb.le.npthisb) return
if (nnpjct(n).eq.0 .and. npfemt(n).eq.0) return
890 continue
900 continue
910 continue

```

Further details about BIRTH can be found in an appendix.

IV.A.3) Finish a Species, AUTOVLM, Restore E and B, DAPART Convert Raw Counts To Densities, Save Histories

```

c           Finish a species
200 if (mod(itp1,ipplt).eq.0) AUTOVLM
c           Set up VLIMs for Pplots NEXT time step.

c           unfix (take species dependence out of) E and B
er/q/z(i,j)=er/q/z(i,j)/hqvm
br/q/z(i,j)=br/q/z(i,j)/hqvmc

c           Convert raw counts to densities by cell vol division
c           EFFVOL adjusted to account for nearby Bs. (Appendix EEFVOL)
den(i,j,n)=den(i,j,n)*rppsp(n)/effvol
ur/q/z(i,j,n)=ur/q/z(i,j,n)*rppsp(n)/effvol

rho(i,j)=rho(i,j)+sqsgnl(n)*den(i,j,n)
jr/q/z(i,j)=jr/q/z(i,j) +sqsgnl(n)*ur/q/z(i,j,n)

vden=1./amax1(den(i,j,n),dmin)
ur/q/z(i,j,n)=ur/q/z(i,j,n)*vden
tem(i,j,n)=rhs(i,j)*rppsp(n)*vden/effvol {RHS contains raw sumUU}
tem(i,j,n)=smsngl(n)*(tem(i,j,n))          {builds TEMperature}
. -ur(i,j,n)**2-uq(i,j,n)**2-uz(i,j,n)**2)/(D3*boltzk)

if (mod(it,ipplt).eq.0) call daspec {Diagnostic After SPECies}

c           Save history

```

The comment to save history is accomplished by a call to HISTORY and that routine is again documented in the DIAGNOS routine, subsection F.

IV.B) Call DSOURCE (See VI. Diagnos)

The diagnostics of the total plasma source terms summed over all species is carried out by the entry DSOURCE in the subroutine DIAGNOS. Details of this routine are given in Sec. VI.D

IV.C) Call FIELD-The Director of the Field algorithm, using FLDSET (See V. BCSET)

The purpose of the subroutine FIELD is to provide the E and B fields needed for the next particle time step given the instantaneous plasma source terms. Whether magnetostatic or Darwin, the field system is elliptic rather than hyperbolic so no previous field history are needed. Previous values make excellent 1st guesses for iterative solvers, however. We

compute A_r , A_q , A_z and Φ using the latest set of source terms. The needed B fields are then computed from the curl of A .

The variable IWHEN has no physics or FORTRAN purpose: should the code die, a check of IWHEN with a debug routine such as DBX reveals the section of FIELD quickly.

The three components of A and Φ are computed from Poisson-type elliptic equations. The r and θ component of A are in fact derived from the curl curl operator. The most important requirement, given the existence of VDKR is to have the boundary conditions arrays KEY, KN, BV, BVR, and BVZ properly set. This is taken care of in FLDSET that is called using appropriate arguments before each call to VDKR.

```

c=====
c          subroutine field
c
c          etest=1.e-3
c          itts=100
c          nerdely=0
c          if (irntyp.eq.'rlbl') nerdely=1
c          if (nerdely.eq.1) go to 400
c
c          B Calculation
c          first Aq
c          -----Set internal boundaries.-----
c          call fldset(Jaq)
c          now set Aq rhs
c          do 230 i=1,nri      do 230 j=1,nz1
c          230   rhs(i,j)=-pi4*jq(i,j)/C
c
c          --- Now solve (del)2 Aq=-pi4*Jq/c -----
c          itt=itts
c          call vdkr(nr2,aq,rhs,tp,tc,tm,t0,fsing,
c          x           dr,nri,rg,dz,nz1,itt,snglstp,srstor,
c          x           err,ier,etest,key,kn,bv,bvr,bvz,omgAq,iperodz,1)
c          write (7,233) itt,itts,omgAq,err,ier
c          if (iotty.ne.0) write (59,233) itt,itts,omgAq,err,ier
c          233 format (' FIELD Aq itt,itts,omgAq,err,ier ',2i4,2e11.3,i7)
c          ----- Now find Br,z from curl Aq. -----
c          call grdcrls(nr2,aq,rhs,br,bz,tm,t0,fsing,
c          x           dr,nri,rg,dz,nz1,itt,itj,snglstp,srstor,
c          x           err,ier,etest,key,kn,bv,bvr,bvz,omgAq,iperodz,1)
c          now Ar
c          -----Set internal boundaries.-----
c          call fldset(Jar)
c          now set Ar rhs
c          do 250 i=1,nri      do 250 j=1,nz1

```

```

250   rhs(i,j)=-pi4*jr(i,j)/C
c
c      --- Now solve (del)2 Ar=-pi4*Jr/c -----
itt=itts
call vdkr(nr2,ar,rhs,tp,tc,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itts,itt,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgAr,iperodz,1)
write (7,253) itt,itts,omgAr,err,ier
if (iotty.ne.0) write (59,253) itt,itts,omgAr,err,ier
253 format (' FIELD Ar itt,itts,omgAr,err,ier ',2i4,2e11.3,i7)
call grdcrlS(nr2,ar,rhs,tp,bq,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itt,itj,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgAr,iperodz,1)
c      now Az
c      -----Set internal boundaries.-----
call fldset(Jaz)
c      now set Az rhs
do 280 i=1,nr1      do 280 j=1,nz1
280   rhs(i,j)=-pi4*jz(i,j)/C
c
c      --- Now solve (del)2 Az=-pi4*Jz/c -----
itt=itts
call vdkr(nr2,az,rhs,tp,tc,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itts,itt,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgAz,iperodz,0)
write (7,283) itt,itts,omgAz,err,ier
if (iotty.ne.0) write (59,283) itt,itts,omgAz,err,ier
283 format (' FIELD Az itt,itts,omgAz,err,ier ',2i4,2e11.3,i7)
c -----
c      To find Curl(Ar,Az)|q
c      call grdcrlS(ndim,Ar,Jr,tp,DzAr,tm,t0,fsing,...,jvc=1)
c      call grdcrlS(ndim,Az,Jz,DrAz,tc,tm,t0,fsing,...,jvc=0)
c      Bq=Curl(Ar,Az)|q=DzAr-DrAz
c -----
c      ----- Now find Bq from curl Ar,Az -----
call grdcrlS(nr2,az,rhs,tp,tc,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itt,itj,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgAz,iperodz,0)
do 290 i=1,nr1      do 290 j=1,nz1
290   bq(i,j)=bq(i,j)-tp(i,j)
c
c      E Calculation
400 continue
do 480 j=1,nz1      do 480 i=1,nr1

```

```

480  rhs(i,j)=-pi4*rho(i,j)
c      -----Set all boundaries.-----
call fldset(Jphi)
c
c      --- Now solve (del)2 Phi=-pi4*Rho -----
itt=itts
if (nr.gt.1 .and. nz.gt.1) then
call vdkr(nr2,phi,rhs,tp,tc,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itt,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgphi,iperodz,0)
write (7,543) itt,itts,omgphi,err,ier
if (iatty.ne.0) write (59,543) itt,itts,omgphi,err,ier
543 format (' FIELD Phi itt,itts,omgphi,err,ier ',2i4,2e11.3,i7)
c
c      ----- Now find El(icor,jcor) from -grad Phi(cor).
call grdcrlS(nr2,phi,rhs,er,ez,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itt,itj,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgphi,iperodz,0)
else
call vdkr1d(nr2,phi,rhs,tp,tc,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itt,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgphi,iperodz,0)
c      ----- Now find El(icor,jcor) from -grad Phi(cor).
call grdcrlS1(nr2,phi,rhs,er,ez,tm,t0,fsing,
x           dr,nr1,rg,dz,nz1,itt,itj,snglstp,srstor,
x           err,ier,etest,key,kn,bv,bvr,bvz,omgphi,iperodz,0)
endif
return
end

```

IV.D) Call DFIELD (See VI. DIAGNOS)

The fields are diagnosed and plotted in a straightforward way in DFIELD. Details are found in Sec. VI.E.

V) User Specified Physics Routines

V.A) Macroscopic Initial Profile Generation by INPROF

For each species, INIT calls the routine INPROF for every cell center. This is the place for the general user to build his/her personalized initial configuration. A few examples are given here.

```
c=====INPROF==*
subroutine inprof(i,j,n,rr,zz,deno,tpo,uro,uqo,uzo)
save jfirst
data jfirst / 0 /
c-----hewett 1988, 78
c
c      for initialization,
c      istate=1      Homogeneous Load
c      =2      Exponential z Profile
c      =3      Realistic LBL Source
c      =4      Uniform .75 Load

iastat=iabs(istate)
deno=0.      tpo=0.
uro=0.      uqo=0.      uzo=0.
go to (100,200,300,400), iastat
c      Homogeneous Load
100 zfac=1.
      tpo=temin(n)*zfac
      deno=denmx(n)*zfac
      return
c      Exponential z Profile
200 znot=z0*(zmax-zmin)
      fac=(zmax-zz)/znot
      deno=denmx(n)*exp(-fac)
      tpo=temin(n)
      return
c      Realistic LBL Source
300 if (sqsgnl(n).lt.0.) go to 350
c                                         IONS
c      ion species first
c                                         RHOS at cell centers
      rwire=10.
      ipwire=dri*(rwire-rmin)+1.
      r1=rg(ipwire)-drh
      tang=tanh((rr-r1)/r0)
      deno=.5*(denmx(n)-denmn(n))*(1.-tang)+denmn(n)
      tpo=temin(n)
```

```

if (ntrnlbc.eq.1) then
  z1=.5*(zmax+zmin)
  arg=-((rr-r1)**2+(zmin-zz)**2)/z1**2
  deno=deno*(1.-exp(arg))
endif
if (deno.gt.0.) uro=udrft(1,n)*denmx(n)/deno
return

c                                     ELECTRONS
350 if (jfirst.eq.0) then
    jfirst=1
    call fldset(Jphi)
c Solve d/dr(r dS/dr)+r d/dz(dS/dz) =-r pi4 e DENi[1-exp(eS/kTe)]
    call phical(...,phi,den,...)           (S=phi)
endif
tpo=temin(n)
deno=0.
if (i.lt.ipwire) deno=den(i,j,1)*exp(evkte*phi(i,j))
uro=0.
return
c      Uniform .75 Load
400 rlimit=rmin+.45*(rmax-rmin)
if (rr.lt.rlimit) then
  deno=denmx(n)
  uro=udrft(1,n)
  tpo=temin(n)
endif
return
end

```

V.B) User Specified Boundary/Structure Physics in BCSET

BCSET is a general group of boundary conditions setting routines. They are quite separate in their functions but can be logically grouped within this one routine using several entries. The advantage is that the close proximity facilitates the physics interplay that needs to occur between the various boundary requirements.

V.B.1) GEOSET

This entry is called very early in the run for the purpose of specifying the boundary in internal structures. It is here that the arrays STRGEN and NSTPROP are set thus determining the physical geometry of the simulation. Presumably, this routine could be called during the run if we want a moving structure.

```

c=====BCSET=====
      subroutine bcset(ifld)
      save tfinite
      namelist /gset/ nstruct, strgen, nstrmx
c -----GEOMETRY SPECIFICATION ARRAYS-----
c SKEY(icen,jcen)=+1.(0.) if DEN>=(<)CTOFF (in the plasma)
c           =-1. if IN boundry (periodic ghost cell only 1.or0.)
c KEY(icor,jcor) =+1(0) if in the plasma(vacuum)
c           =-1 if ON or IN boundry (periodic bndry 1or0 only)
c               Orient the surface with KORNT
c
c KORNT(icor,jcor) =0 not a BC pt.
c   = +1( -1) for surface normal      +-----+-----+
c       in for(back)ward R-direction    | 8 bbb 10 bb 12 |
c   =+10(-10) for surface normal     | bbbbbbbb|bbbbbbb|
c       in for(back)ward Z-direction   +-1 -----+--- 1+
c   =+-12,+11 +dr+dz,-dr-dz direction | bbbbbbbb|bbbbbbb|
c   =+- 8,+ 9 -dr+dz,+dr-dz direction | -12 b -10 b -8 |
c                                         +-----+-----+
c
c
c   /|          -----+          |/          +----+
c   /| +11        /////|          +9 !/        |/ /////
c   /+---        +12 /|          ---+/         |/ +8
c   /////         /|          /////         |/ /
c                           KORNT          4/90
c   |/          /////          |/          /////
c   |/ -12        ---+/          -8 /|        /+---
c   /////         -11 |/          /////         |/ -9
c   +----        /|          ---+         /|
c
c -----STRUCTURE SPECIFICATION ARRAYS-----
c User sets NSTPROP, QBRS, and STRGEN
c GEOMSET sets NSTRUCT, KCIJ, KCORR, SKEY->KORNT and KEY
c
c   NSTRUCT(NSTR,1)--name of structure numbered NSTR
c   NSTRUCT(NSTR,2/3)--KCOR start/stop pt in KCIJ.
c   IorJCOR = KCIJ(KCOR,1or2) for structure corner pt KCOR
c   ex: do # kcor=nstruct(nstr,2),nstruct(nstr,3)
c        # key(kcij(kcor,1),kcij(kcor,2))=-1
c   KCORR(icor,jcor)=KCOR (cross reference array)
c   NSTPROP(NSTR,Nspecies)--n00-n50, n51--%ThRemt,PRFLCT
c   (n=1/2/0 Femt/Njct/Neither), negative=no dithr
c   QBRS(KCOR,LorD,Nspecies)..."Reservoir" for emtng/njectng
c   LorD=1(2) [emtng surface Left(Down) from corner]

```

```

c -----
c      STRGEN(NSTR,ityp,1) r, z, radius, astar, astop (a=0 e_r)
c                           r1, z1, r2, z2, hwidth
c      STRGEN(NSTR,1) 1/2 pie bar (neg = undo)
c      STRGEN(NSTR,2)          rpt rstr
c      STRGEN(NSTR,3)          zpt zstr
c      STRGEN(NSTR,4)          radius rstp
c      STRGEN(NSTR,5)          astar zstp
c      STRGEN(NSTR,6)          astop hwidth
c      data pie,bar/1.,2./
c      data tbsave,tpsave,tvsave / 0.,0.,0. /
c -----
c      entry geoset(ifld)
c      tfinite=1.e-12*dt
c      Rmin R BCs
c BAR  STRGEN(NSTR,1/2/3/4/5/6) 2/rstr/zstr/rstp/zstp/hwidth
nstr=1
cornmn=1.                      cornmx=1.
if (nr.eq.1) cornmn=0.           if (nr.eq.1) cornmx=0.
strgen(nstr,1)=bar
strgen(nstr,2)=rmin  strgen(nstr,3)=zmin-.01*(2.*cornmn-1.)*dz
strgen(nstr,4)=rmin  strgen(nstr,5)=zmax+.01*(2.*cornmx-1.)*dz
strgen(nstr,6)=.01*dr   nstruct(nstr,1)=8h Rmin
do 110 n=1,nsp
110 nstprop(nstr,n)=250
nstprop(nstr,2)=100
c      Rmax R BCs
nstr=2
cornmn=1.                      cornmx=1.
if (nr.eq.1) cornmn=0.           if (nr.eq.1) cornmx=0.
strgen(nstr,1)=bar
strgen(nstr,2)=rmax  strgen(nstr,3)=zmin-.01*(2.*cornmn-1.)*dz
strgen(nstr,4)=rmax  strgen(nstr,5)=zmax+.01*(2.*cornmx-1.)*dz
strgen(nstr,6)=.01*dr   nstruct(nstr,1)=8h Rmax
do 120 n=1,nsp
120 nstprop(nstr,n)=0
c      Zmin Z BCs  3
if (iperodz.ne.1) then
  nstr=nstr+1
  cornmn=0.                      cornmx=0.
  if (nr.eq.1) cornmn=0.           if (nr..eq.1) cornmx=0.
  strgen(nstr,1)=bar
  strgen(nstr,2)=rmin-.01*(2.*cornmn-1.)*dr  strgen(nstr,3)=zmin
  strgen(nstr,4)=rmax+.01*(2.*cornmx-1.)*dr  strgen(nstr,5)=zmin

```

```

        strgen(nstr,6)=.01*dz      nstruct(nstr,1)=8h Zmin
        do 130 n=1,nsp
130      nstprop(nstr,n)=0
c          Zmax Z BCs  4
        nstr=nstr+1
        cornmn=0.                  cornmx=0.
        if (nr.eq.1) cornmn=0.      if (nr.eq.1) cornmx=0.
        strgen(nstr,1)=bar
        strgen(nstr,2)=rmin-.01*(2.*cornmn-1.)*dr  strgen(nstr,3)=zmax
        strgen(nstr,4)=rmax+.01*(2.*cornmx-1.)*dr  strgen(nstr,5)=zmax
        strgen(nstr,6)=.01*dz      nstruct(nstr,1)=8h Zmax
        do 140 n=1,nsp
140      nstprop(nstr,n)=0
        endif
c          Interior BCs  5
        nstrmx=nstr
        if (ntrnlbc.eq.1) then
c          istate=5      "Grid wire" at zmax
c PIE  STRGEN(NSTR,1/2/3/4/5/6)  1/rcen/zcen/radius/astr/astp (a=0 e_r)
        nstr=nstr+1
        rwire=10.
        ipwire=dri*(rwire-rmin)+1.
        strgen(nstr,1)=pie
        strgen(nstr,2)=rg(ipwire)-drh      strgen(nstr,3)=zmin
        strgen(nstr,4)=.002032
        strgen(nstr,5)=0.                  strgen(nstr,6)=360.
        nstruct(nstr,1)=8h Wire
        do 150 n=1,nsp
150      nstprop(nstr,n)=0
        endif
        nstrmx=nstr
c
c -----STRUCTURE NAMELIST CONTROL-----
c     JSTGNSW = 1 use NSTRUCT,STRGEN,NSTRMX as BCSET generated
c                 = 0(-1) read additional namelist from tty(unit 6)
c
        if (jstgnsw.le.0) then
          if (jstgnsw.eq.0) then
            write (59,gset)      read (59,gset)
            else
              read (6,gset)
            endif
            write (7,gset)
            jstgnsw=1

```

```

    endif
    write (100,gset)
    if (nstrmx.gt.nstmax) stop "nstmax"
    call geomset(npthisb,namesp,nstruct,strgen,nstrmx,
    . jstgnsw,nstmax,dr,dz,nr2,nz2,rg,zg,kcij,rmax,rmin,kcorr,
    . kcormx,skey,iperodz,kornt,key)
    return

```

V.B.2) PBCSET

PBCSET is called as processing is initiated on each new species N in TRANS. If, for example, we wish to field emit electron, TRANS calls PBCSET with the latest E field before any electrons are pushed. PBCSET then computes the surface charge density that this field would induced at each corner on all surfaces that are to emit. This charge density is converted to a net charge by multiplying the induced desity by the surface area associated with that corner. This charge then loaded into the charge reservoir QBRS(KCOR,LORD,N) to be emitted from either the surface to the left LORD=1 or down LORD=2 from the KCORth corner for this species N (electrons in this example).

This particular version given here has two options for injection. We often use the run type IRNTYP as a flag to switch options so that the code can be used for several types of work at the same time. The case for IRNTYP='b x ch' (for Beam a X the CHamber) is a simple injection algorithm where the normal velocity, stored in UNBSTR(KCOR,LORD,N) is obtained from BEAM(NSTR,N,2) and net charge to be added to the reservoir, again QBRS(KCOR,LORD,N), is the beam amplitude BAMP [input from BEAM(NSTR,N,1) and suitably modified by the time modifiers also stored in BEAM(NSTR,N,5/6/7/8)] times the species charge SQSNGL(N) times the injection velocity UNBSTR(KCOR,LORD,N) times the area IRE*AREAR+IZE*AREAZ [modified for horizontal or vertial injection] times DT. This is just the charge that such a beam would have moved through the surface in one time step.

The second option is a "DEMAND" injection option. We wish to inject only the particles that are needed to maintain a preselected density, again from BEAM(NSTR,N,1). The idea here is to compare the preselected density with the average density DENAV and add the needed charge to the reservoir to be injected.

Count are made of the number of new particles to be injected and some care is taken to insure there is enough room to accomodate all the particles needed to satisfy these injection/emission requirements. Finally IPBC is set for all corners in the simulation for the species now to be pushed.

```

c -----PARTICLE BC ARRAYS-----
c Given NSTRUCT,KCIJ the user inputs NSTPROP TEMIN BEAM
c & this routne sets IPBC UNBSTR TEM TEMSTR QBRS NPFEML NPNJCT
c IPBC(icor,jcor) = cell corner particle property.
c >= 0-50 *2 is the % returned. A new Vth generated
c from TEMSTR(kcor,lord,n).

```

```

c      51 perfect reflection.
c      QABSRB(icor,jcor) = Absorbed charge at this cell corner.
c      Reemit .02*|IPBC|*QABSRB/SQ using ETNVTP (from BEAM(nstr,n,4))
c      & DITHR from sign of NSTPROP(nstr,n).
c      QBRS(kcor,LorD,n) = FEMT/NJCT reservoir charge at cell face.
c      UNBSTR(kcor,lorD,n) = Normal FEMT/NJCT velocity magnitude.
c      TEMSTR(kcor,lorD,n) = Thermal spread about injection velocity.
c -----
c      For example, to absorb and thermally reemit 50% of the particles
c      species n=3 that hit a NSTR and inject with a DITHRed beam with
c      Bden=1e12 with normal velocity=3C/4 & TEMSTR=3eV Xverse thermal
c      spread that turns on instantly after 1.5 ns, remains on until
c      5 ns & falls linearly to zero by 6.2 ns with anisotropy VN/VTH
c      of 30%, use
c      nstprop(nstr,3)=225
c          beam(nstr,3,1)=1.e12   beam(nstr,3,2)=7.5e9
c          beam(nstr,3,3)=3.     beam(nstr,3,4)=.3
c          beam(nstr,3,5)=1.5e-9 beam(nstr,3,6)=0.
c          beam(nstr,3,7)=5.e-9  beam(nstr,3,8)=1.2e-9
c -----
c      entry pbcset(ifld)
c      ...
c      NSTR=1,2,3,4,5  rmin,rmax,zmin,zmax,prtcl ldr
c      N=1,2,3,4      trg ions, trg electrns, bem ions, new electrns
c      BEAM(NSTR,n,1/   2/   3/   4/   5/   6/   7/   8)
c          Bden,unbstr,temstr,etnvtp, turnon,rise,turnof,fall
c          TURNON--time (from t=0) until a Bden is started
c          TRISE--time (from t=turnon) until full Bden (linear)
c          TURNOF--time (from t=0) that a Bden starts to fall off
c          TFALL--time (from t=turnof) until zero
c          neti=0        nete=0
c          do 670 nstr=1,nstrmx
c              if (nstprop(nstr,n).eq.51.or.nstprop(nstr,n).eq.0) goto670
c              nbtyp=iabs(nstprop(nstr,n))/100
c              nbpct=iabs(nstprop(nstr,n))-100*nbtyp
c              beamq =beam(nstr,n,1)
c              turnon=beam(nstr,n,5)           trise =beam(nstr,n,6)
c              turnof=beam(nstr,n,7)          tfall =beam(nstr,n,8)
c              bamp=0.
c              if (beamq.ne.0. .and. time.lt.(turnof+tfall)) then
c                  if (time.ge.turnon)
c                      bamp=beamq*amin1(1.,(time-turnon)/(trise+tfinite))
c                  if (time.gt.turnof)
c                      bamp=bamp*amax1(0.,1.-(time-turnof)/(tfall+tfinite))

```

```

        endif
do 650 kcor=nstruct(nstr,2),nstruct(nstr,3)
    if (bamp.eq.0. .and. nbtyp.eq.0) go to 645
    i=kcij(kcor,1)           j=kcij(kcor,2)
    if (kornt(i,j).eq.0) go to 650
    if ((i.eq.1 .and. j.eq.1) .or.
        (j.eq.1 .and. nstr.eq.2) .or. (i.eq.1 .and. nstr.eq.4)
x      .or. kornt(i,j).eq.+11 .or. kornt(i,j).eq.-12) goto645
    knz=kornt(i,j)/8          knr=kornt(i,j)-knz*10
    arear=pi2*rg(i)*dz       areaz=pi2*rgcc(i)*dr
    ikornt=iaabs(kornt(i,j))
    ldstar=1                  ldstop=1
    if (j.eq.1 .or. ikornt.eq.10 .or.
x      kornt(i,j).eq.+9 .or. kornt(i,j).eq.-8) go to 630
    ldstop=2
    if (kornt(i,j).eq.+12 .or. kornt(i,j).eq.-11) go to 630
    ldstar=2
630   do 640 lord=ldstar,ldstop
c           IRE\IZE=1 means emit horizontally\vertically
        ire=lord-1             ize=2-lord
        if (nbtyp.eq.1) then
            FIELD EMISSION
            sgnr=isign(ire,knr)           sgnz=isign(ize,knz)
            qnete=(ire*.5*(er(i,j)+er(i,j-ire))*arear*sgnr
                +ize*.5*(ez(i,j)+ez(i-ize,j))*areaz*sgnz)/pi4
            qbrs(kcor,lord,n)=qnete
            neinc=qbrs(kcor,lord,n)/sq(n)
            nete=nete+max0(0,neinc)
            unbstr(kcor,lord,n)=beam(nstr,n,2)
            elseif (nbtyp.eq.2) then
                INJECTION
                if (irntyp.eq.'b x ch') then
c                    uniform ion feed pre 9/23/90
                    unbstr(kcor,lord,n)=beam(nstr,n,2)
                    delq=sqsngl(n)*bamp*unbstr(kcor,lord,n)*
                        (ire*arear+ize*areaz)*dt
                    qbrs(kcor,lord,n)=qbrs(kcor,lord,n)+delq
c                    uniform ion feed pre 9/23/90
                elseif (irntyp.eq.'rlbl') then
c                    DEMAND type of injection algorithm 9/23/89
                    denav=.5*(den(kcij(kcor,1) ,kcij(kcor,2) ,n)
                        +den(kcij(kcor,1)+knr,kcij(kcor,2)+knz,n))
                    deltp=amax1(0.,(beam(nstr,n,1)-denav)*volcc(2))
                    qbrs(kcor,lord,n)=qbrs(kcor,lord,n)+deltp*sqsngl(n)

```

```

        unbstr(kcor,lord,n)=beam(nstr,n,2)
c     DEMAND type of injection algorithm 9/23/89
        endif
        neti=neti+qbrs(kcor,lord,n)/sq(n)
        endif
        temstr(kcor,lord,n)=beam(nstr,n,3)
640      continue
645      if (nbpct.gt.0 .or. nbpct.lt.51 .or. nbpct.eq.61) then
c           SET ThRemt temperature
        tem(kcij(kcor,1),kcij(kcor,2),n)=temin(n)
        endif
650      continue
670      continue
        npnjct(n)=neti          npfemt(n)=nete
        if (iatty.ne.0) write (59,657) it, namsp(n),npnjct(n),
           npfemt(n)
        if (npnjct(n).ne.0 .and. npfemt(n).ne.0)
           write (7,657)   it, namsp(n),npnjct(n),npfemt(n)
657      format(i4,' PBCSET GPBC njt,nfemt ',5x,a8,2i8)
        endif
c           SET IPBC for all partcls and structures
        do 690 nstr=1,nstrmx
        do 690 kcor=nstruct(nstr,2),nstruct(nstr,3)
           nbtyp=iabs(nstprop(nstr,n))/100
           nbpct=iabs(nstprop(nstr,n))-100*nbtyp
690      ipbc(kcij(kcor,1),kcij(kcor,2))=nbpct
        return

```

V.B.3) FLDSET

Here again we have used IRNTYP to switch between various types of runs. We also use the notation INN, IND, etc to express boundary conditions on various surfaces. The second character, either N for Neumann or D for Dirichlet, represents the boundary condition to be applied on the lower boundary and the third character represents the condition to be applied on the top boundary. IPD is also allowed and that gives a periodic condition. This stuff is mostly a holdover from earlier work and it is not as useful as in earlier codes that had no internal boundaries.

The coding below the statement label 2000 contains time dependent boundary condition provisions motivated by physics associated with external circuits. Though primitive at the moment, this section contains the ideas that surfaces connected to external voltage sources can be expected to control Dirichlet boundary conditions for the electrostatic potential and external voltages produce E fields that produce finite time derivatives in A the magnetic vector potential. The time derivitives can then determine a time dependent Dirichlet condition for A that ultimately shows up in a modified B field-just what external

voltages are supposed to do.

```
c -----FIELD BC ARRAYS-----
c Given NSTRUCT,KCIJ the user sets BV,BVR,BVZ and KN
c   CAUTION: Boundaries are built only at cell edges!
c   KN(icor,jcor)=0, Dirichlet pt with value=BV(icor,jcor) on bndry.
c   =+1(-1) on bndry w/ for(back)ward R-drivtv  [=+(-)2 curl-type]
c   =KN+10(-10) on bndry w/ for(back)ward Z-drivtv
c   BVR/Z(icor,jcor) = Neumann value for the R/Z derivative.
c       IFLD=1--Phi    =2--Aq    =3--Ar    =4--Az    =5--Jionc
c       data idn,ipd,ind,idd,inn, Jphi, Jar, Jaq, Jaz/
c           .      1, 2, 3, 5, 6,     1, 3, 2, 4/
c -----
c           entry fldset(ifld)
c           go to (1100,1200,1300,1400), ifld
c           Phi
c 1100 iar=idd
c           if (irntyp.ne.'nh2s') iar=inn
c           if (irntyp.eq.'b x ch') iar=ind
c           if (irntyp.ne.'nh2s') iaz=ind
c           if (irntyp.eq.'b x ch') iaz=idn
c           if (irntyp.eq.'rlbl') iaz=inn
c           iaz=iaz*(1-iperodz)+ipd*iperodz
c           go to 1500
c           Aq
c 1200 iar=idd      iaz=inn*(1-iperodz)+ipd*iperodz
c           go to 1500
c           Ar
c 1300 iar=idn      iaz=idn
c           go to 1500
c           Az
c 1400 iar=ind      iaz=inn
c
c 1500 do 1550 i=1,nr1
c           do 1550 j=1,nz1
c               bvr(i,j)=0.          bvz(i,j)=0.          bv(i,j)=0.
c 1550 kn(i,j)=0
c           General Rmin max Z BCs
c                           " at rmin"
c           if (iar.eq.ind.or.iar.eq.inn) then
c               nstr=1
c               do 1750 kcor=nstruct(nstr,2),nstruct(nstr,3)
c 1750 kn(kcij(kcor,1),kcij(kcor,2))=-1
c           endif
```

```

c                                " at rmax"
if (iar.eq.idn.or.iar.eq.inn) then
nstr=2
do 1755 kcor=nstruct(nstr,2),nstruct(nstr,3)
1755  kn(kcij(kcor,1),kcij(kcor,2))=-1
      endif
c          General Zmin max Z BCs
if (iperodz.eq.1) go to 2000
c                                " at zmin"
if (iaz.eq.ind.or.iaz.eq.inn) then
nstr=3
do 1760 kcor=nstruct(nstr,2),nstruct(nstr,3)
1760  kn(kcij(kcor,1),kcij(kcor,2))=
       kn(kcij(kcor,1),kcij(kcor,2))+10
      endif
c                                " at zmax"
if (iaz.eq.idn.or.iaz.eq.inn) then
nstr=4
do 1765 kcor=nstruct(nstr,2),nstruct(nstr,3)
1765  kn(kcij(kcor,1),kcij(kcor,2))=
       kn(kcij(kcor,1),kcij(kcor,2))-10
      endif
c
2000 continue
do 2010 i=1,nr1
do 2010 j=1,nz1
      bvr(i,j)=0.          bvz(i,j)=0.
2010  bv(i,j)=0.
      go to (2100,2100,2300,2400), ifld
c                                Phi
2100 if (time.le.tpsave) return
      do 2180 nstr=1,nstrmx
c          Voltq--increases NSTR flux with time
c          PHISTR(NSTR,0&1&2&3&4&5)    mag0,mag,turnon,rise,turnoff,foff
c          TURNON--time (from time=0) that a voltage is applied
c          TRISE--time (from time=turnon) until full voltage (linear)
c          TURNOFF--time (from time=0) that a voltage is turned off
c          TFALL--time (from time=turnoff) for linear falloff
      phibm =phistr(nstr,1)
      turnon=phistr(nstr,2)        trise =phistr(nstr,3)
      turnof=phistr(nstr,4)        tfall =phistr(nstr,5)
      if (phibm.ne.0. .and. time.lt.(turnof+tfall)) then
          phib=phibm*amax1(0.,amin1(1.,(time-turnon)/(trise+tfinite)))
          phib=phib*amin1(1.,amax1(0.,1.-(time-turnof)/(tfall+tfinite)))

```

```

      tpsave=time
c      keep TPSAVE in case you call here many times same DT.
do 2150 kcor=nstruct(nstr,2),nstruct(nstr,3)
      kn(kcij(kcor,1),kcij(kcor,2))=0.
2150  bv(kcij(kcor,1),kcij(kcor,2))=phistr(nstr,0)+phib
      endif
2180 continue
      if (iar.eq.ind) then
          kn(1,1)=+11           kn(1,nz1)=-9
          endif
      return
c               Aq
c               "Flux at rmax"
2200 if (time.le.tvsave) return
      do 2280 nstr=1,nstrmx
c          Voltq--increases NSTR flux with time
c          VOLT(NSTR,R&Q&Z,1&2&3&4&5) mag,turnon,rise,turnof,fall
c          TURNON--time (from time=0) that a voltage is applied
c          TRISE--time (from time=turnon) until full voltage (linear)
c          TURNOF--time (from time=0) that a voltage is turned off
c          TFALL--time (from time=turnof) for linear falloff
      voltq =volt(nstr,2,1)
      tunron=volt(nstr,2,3)      trise =volt(nstr,2,3)
      turnof=volt(nstr,2,4)      tfall =volt(nstr,2,5)
      if (voltq.ne.0. .and. time.lt.(turnof+tfall)) then
          volq=voltq*amax1(0.,amin1(1.,(time-turnon)/(trise+tfinite)))
          volq=volq*amin1(1.,amax1(0.,1.-(time-turnof)/(tfall+tfinite)))
          raqwdot=C*volq/pi2
          raqext=raqext+(time-tvsave)*raqwdot
          tvsave=time
c          keep TSAVE in case you call here many times same DT.
      do 2250 kcor=nstruct(nstr,2),nstruct(nstr,3)
2250  bv(kcij(kcor,1),kcij(kcor,2))=raqext/rg(kcij(kcor,1))
      endif
2280 continue
2290 return
c               Ar
2300 continue
      return
c               Az
2400 continue
      return
c               Jionc
2500 flowi=sqsngl(1)*denmx(1)*udrft(1,1)

```

```
c           "Ji at zmin"
nstr=3
do 2320 kcor=nstruct(nstr,2),nstruct(nstr,3)
2320   bvr(kcij(kcor,1),kcij(kcor,2))=-flowi
c
do 118 k=1,nr1      key(k,1)=-1
118  key(k,nz1)=-1
do 119 k=1,nz1      key(1,k)=-1.
119  key(nr1,k)=-1.
c      set corners
kn(1,1)=+11      bvr(1,1)=-flowi
kn(1,nz1)=-9      bvr(1,nz1)=-flowi
return
end
```

VI) User Specified Diagnostics in DIAGNOS

The intent of subroutine DIAGNOS is to provide a common location for all physics I/O for the code. However, diagnostics have much diversity and so in effect DIAGNOS is simply a grouping of these several types of independent diagnostic routines, each with its own entry point, under the one master subroutine name. DIAGNOS itself is never called. The discussion that follows is a guide to some of the diagnostics that have been found useful in the past. Although there are many places in GYMNOS that the inexperienced user is advised NOT to make changes, the choice of diagnostics and how they are displayed do need strong input and involvement by the user. These examples should serve as a guide.

The details of the plot routines used here are given in the Appendix on graphics.

VI.A) DPART Generates Particle Phase-Space Plots

One of the reasons to run a PIC model rather than a fluid or MHD model is that the kinetic behavior of the particles is important. The most effective windows into this behavior are prtcl phase-space plots—point or “scatter” plots of individual particle velocities as functions of their coordinates. A good example is found in the VR vs. R plots of one of the ion source test cases that display clearly the kinetic ion behavior. The corresponding plot of VR vs. Z would display the transverse position of the most active ions.

Since it is expensive to run through the particles if they are stored externally on disk, we try to take advantage of the buffering cycle that must occur to get the particles through PARMOV in the normal course of a time step. After a buffer load of particles has been processed, a call is made to DPART (Diagnostics on PARTicles) to see if this is a time step to do phase-space plots. If it is, the frames and scaling for these plots are set up on the first call and those particles that need to be plotted from the current buffer in core are plotted. Control is then returned to TRANS and the next buffer of particles is brought into core. After this next buffer is advanced in time in PARMOV, DPART is again called. This time the point plot PPLOT routines place points in the plot regions drawn previously.

One can see this logic, set up with the variable JFIRST, in the section DPART in routine DIAGNOS. We find it useful to put all six combinations on a given plot frame. To put them on separate frames would require either separate passes through the particle buffering routines (as occurs in DAPART below) or the ability to open up separate plot channels. A strong benefit of having all six phase-space plots on one frame is the correlation in space immediately available from the individual plots. What we are attempting to describe is a concept similar to a blueprint an object with the 3 standard views, front, top and side. We are arguing that it is useful to have all 3 views on the same piece of paper.

VI.B) DAPART Generates Particle Position-Space Plots

DAPART (Diagnostics After PARTicles) is necessary because another type of information, their relative locations in R, Z space, can be obtained only by plotting information from individual particles. These plots are expensive, because they require a separate pass through the buffering routines, but are essential to the formation of certain types of insight. One could find a way to put all this information on one frame and thus do these plots in DPART but that's too much information on one frame even for me!

VI.C) DASPECS Plots Macroscopic Source Terms from Each Species

The other way to get particle source information is to sum over particles of each species to get the macroscopic moments, density DEN, temperature TEM and drift velocities U. TRANS orchestrates these activities as the particles go through PARMOV; these quantities are necessary for the field solution. We plot this information either in contour plots CPLOTS, vector plots VPLOTS, or averages in one or the other direction with APLOTS.

VI.D) DSOURCE Plots Species-Summed Macroscopic Source Terms

Another of the many activities of TRANS is that when it is finished, when control is returned to main, it will have summed the various species together to form the total charge and current density for use in the field solution. The diagnostic entry DSOURCE is the place where these sources are plotted in various ways.

VI.E) DFIELD Plots Potentials and Fields A, Φ , E, and B

By this point, we have nothing to say that will be a surprise. After the fields are calculated, we plot various parts and pieces with the usual complement of CPLOTS, VPLOTS and APLOTS. It is worth noting that we plot the electron information here because it is not known until after the fields have been advanced in time.

VI.F) HISTORY Plots Histories After “Unraveling” Pointer

The difficulty with maintaining historical information for plotting at the end of the run is obvious; it uses too much storage. Another approach is to maintain smaller arrays for each of the quantities for which history plot are needed, retain only the most recent entries, and plot before we have obtained more new entries than we can store. We have implemented this strategy in GYMNOS. We dimension the history arrays ITHMX and, therefore, save the last ITHMX values. We plot these values the next time HISTORY is called. To achieve full coverage over the course of a long run, one needs to call HISTORY (controlled by IHIST) more frequently than every ITHMX time steps.

To avoid constantly working with a “pushdown” stack when each new entry is added, we keep track of a pointer that contains the location of the oldest element in the file. Every IHIST time steps, a new element is stored there and the pointer is moved to the next oldest entry. When HISTORY is called the array is reordered before plotting so that the oldest information is in the first cell and the latest information is in the ITHMX cell, or the IT cell if IT < ITHMX. Provisions are made to save plot the lastest ITHMX values even if IHIST is bigger than ITHMX, thus causing the loss of some information.

The coding that accomplishes this is illustrated by an example of the history of the electric field energy EEF. For convenience the storage is kept in the common block HISTY that is spread throughout the code using the cliche SHIST.

```
common /histy/ mn6f,ihistpt,eef(0:ithmx),...,mn6l  
c      TRANS  Historical data saved every timme step in TRANS.  
c              IHISTPT is the pointer to the oldest data.
```

```

ihistpt=mod(ihistpt+1,ithmx+1)
eeft=0.                                {Compute E^2/8pi integral.}
do 362 i=1,nr1
  facj=.5                               {A volume "reducer" in mesh corners.}
  do 362 j=1,nz1
    eeft=eeft+facj*vol(i)*(er(i,j)**2+eq(i,j)**2+ez(i,j)**2)
362  facj=.5*(2-j/nz)
  eef(ihistpt)=eeft/pi8

```

The following Fortran in the subroutine DIAGNOS unrolls and plots the latest ITHMX history values of EEF.

```

entry history
write (tprnt,9) time      {Writes TIME on bottom of plot.}
call lblbot(tprnt,20)
tsc(1)=amax1(time-ithmx*dt,0.)
tsc(2)=dt
ittop=min0(ithmx,it)+1
c                         History pointer unroll 2/25/90
if (it.gt.ithmx .and. ihistpt.lt.ithmx1) then
  id=ithmx-ihistpt
  itpp=min0(ihistpt,id)
  do 17 istar=0,itpp
    in=istar
11   in=mod(in+id,ithmx1)
    if (in.lt.istar) go to 17
    if (in.gt.istar) go to 11
    tpef=eef(in)
14   in=mod(in+id,ithmx1)
    tpnef=eef(in)      eef(in)=tpef      tpef=tpnef
    if (in.ne.istar) go to 14
17   continue
    ihistpt=-1
    endif
call lplot(-1,-2,ittop,tsc,eef(0),-1,1,
  'E2energy',8,'time',4,'eef',3)
call empty(8)
return

```

VII) Running GYMNOS (CTSS)

We have set up the file GYMNOS to contain some sample input decks as well as the main code. Thus when GYMNOS is run as input to the COSMOS controller, the input files and controllee are generated when COSMOS is finished.

```
*file name=iatsa
      .
      . . . .
      . . . .
the namelist input file IATSA is here
      .
      . . . .
*file name=tpic
      .
      . . . .
the main GYMNOS Fortran source is here
      .
      . . . .
*precomp tpic ppic
*cft i=ppic,b=bin,l=lgrpic
*ldr i=(bin,/014361/b8),lib=(/014361/pbldc,tv80lib),x=xgrpic
*destroy bin tpic ppic
```

Here the controllee is XGRPIC and the ATSA run can be started by typing XGRPIC ATSA / 1 3 .

XGRPIC is a creation by the loader LDR of the fortran binary BIN created here from the GYMNOS source here called PPIC. The loader will merge BIN with the other binary file B8 containing the support routines such as VDKR, VTRIK, etc. (compiled earlier and residing in local file space) and the plot library PBLDC.

VII.A) Subroutines in GYM30

```
c  GYMNOS (RZ PIC with arbitrary boundaries)
c=====GYMNOS=====
c=====INIT=====
subroutine init
c=====INPROF=====
subroutine inprof(i,j,n,rr,zz,
*   bro,bqo,bzo,ero,eqo,ezo,deno,tpo,uro,uqo,uzo,psi)
c=====TRANS=====
subroutine trans
c=====TRAP=====
subroutine trap
c=====BCSET=====
subroutine bcset(ifld)
c -----
entry geoget(ifld)
c -----
c   entry denset(ifld)
```

```

c -----
c      entry pbcset(ifld)
c -----
c      entry fldset(ifld)
c=====PARMOV=====
c      subroutine parmov
c=====VPARMOV=====
c      subroutine vparmov
c=====VTRAP=====
c      subroutine vtrap
c=====DEATH=====
c      subroutine death(lp,lplast,l,npthisb)
c=====VBIRTH=====
c      subroutine vbirth(npart,vmp,unorm,evnvth,evtvth,krntflg,
c                         edithr,edithz,sgnr,sgnz,remt,zemt)
c=====PARMOVE=====
c      subroutine parmove
c=====PARMOVR=====
c      subroutine parmovr
c=====NBIRTH=====
c      subroutine nbirth(npart,vmp,unorm,evnvth,evtvth,krntflg,
c                         edithr,edithz,sgnr,sgnz,remt,zemt)
c=====BIRTH=====
c      subroutine birth(npart,vmp,unorm,evnvth,evtvth,krntflg,
c                         edithr,edithz,sgnr,sgnz,remt,zemt)
c=====FIELD=====
c      subroutine field
c=====DIAGNOS===
c      subroutine diagnos
c          Plasma Source Plots
c      entry dsource
c          Start Particle-After-Buffer Plots Here
c      entry dparts
c          Particle Plots After Each Buffer
c      entry dpart
c          Call After Each Species is Finished for RHO and J's
c      entry daspec
c          Set Up RZ Space Plot
c      entry dapart
c          E, B, and Potential Plots
c      entry dfield
c          History
c      entry history
*precomp tpic ppic

```

```
*cft i=ppic,b=bin,l=lgrplic
*ldr i=(bin,/014361/b8),lib=(/014361/pbldc,tv80lib),x=xgrplic7
*destroy tpic ppic
```

VII.B) Subroutines in GYMS22

```
C=====FIRST=====
    subroutine first(idrun,idcode,idntfy,memoly,imsg)
c      entry newf3
c      entry tally
c      We're running--do you want a tally?
C=====GETB=====
    subroutine getb(ksp,npthisb,iparmem,ipbuf,nbspec)
    entry resetb
C=====SETB=====
    subroutine setb(nnsp,nmpb,mspec,nspec,nbspec,
      . nmemoly,idimp,nnheadb)
C=====INCB=====
    subroutine incb(n,nbinc,nbspec,ipbuf,iparmem)
C=====RESTART=====
    subroutine restart(mn1f,mn1l,mn2f,mn2l,mn3f,mn3l,mn4f,mn4l,mn5f,
      . mn5l,mn6f,mn6l,ipbuf,iparmem,nbspec,mspec,nspec,numrest,nsp)
c      write a restart dump file
    entry wstart(mn1f,mn1l,mn2f,mn2l,mn3f,mn3l,mn4f,mn4l,mn5f,mn5l,
      . mn6f,mn6l,ipbuf,iparmem,nbspec,mspec,nspec,numrest,nsp)
c      read a restart dump file
    entry rstart(mn1f,mn1l,mn2f,mn2l,mn3f,mn3l,mn4f,mn4l,mn5f,mn5l,
      . mn6f,mn6l,ipbuf,iparmem,nbspec,mspec,nspec,numrest,nsp)
C=====FCOPY=====
    subroutine fcopy(ifrom,ito,ierr)
C=====XPSNPL=====
    subroutine xpnp(a,nr1,nz1)
C=====BPLOT=====
    subroutine bplot(kornt,mstr,nstruct,kcij,nr,nz,
      . rmin,dr,zmin,dz,ndim,nstmax,mxgd,nstr1,nstr2)
C=====VDKR=====
    subroutine vdkr(ndim,s,rhs,tp,tc,tm,t0,fsing,
      x dr,nnr1,rg,dz,nnz1,itt,it,snglstp,srstor,
      x err,ier,etest,key,kn,bv,bvr,bvz,omg1stp,iperodz,jvc)
c      Now compute -GRAD(S) (jvc=0) or CURL(S) (jvc=1)
    entry grdcrlS(ndim,s,rhs,tp,tc,tm,t0,fsing,
      x dr,nnr1,rg,dz,nnz1,itt,it,snglstp,srstor,
      x err,ier,etest,key,kn,bv,bvr,bvz,omg1stp,iperodz,jvc)
```

```

C=====PHICAL=====
  subroutine phical(ndim,s,den,evkte,pi4em,tp,tc,tm,t0,fsing,
x  dr,nnr1,rmin,rg,dz,nnz1,itt,it,snglstp,srstor,
x  err,ier,etest,key,kn,bv,omg1stp,iperodz)
C=====GEMSET=====
  subroutine geomset(npthisb,namesp,nstruct,strgen,nstrmx,
.  jstgnsw,nstmax,dr,dz,nr2,nz2,rg,zg,kcij,rmax,rmin,kgcorr,
.  kcormx,skey,iperodz,kornt,key)
C=====PRT1=====
  subroutine prt1(tp,ndim,nnr1,nz1,name,io)
C=====VTRIDG=====
  subroutine vtridg(fsng,ndim,nnr1,nz1,ijstar,ijskp,
.  cp,cc,cm,q,V,dis,iperod)
C=====MSGPROC=====
  subroutine msgproc(imsg,it,time,idrun,iotty,ektot,eef,ebf,poy,
.  nspec,nspec,nsp)
C=====VDKR1D=====
  subroutine vdkr1d(ndim,s,rhs,tp,tc,tm,t0,fsing,
x  dr,nnr1,rg,dz,nnz1,itt,it,snglstp,srstor,
x  err,ier,etest,key,kn,bv,bvr,bvz,omg1stp,iperodz,jvc)
c      Now compute -GRAD(S) (jvc=0) or CURL(S) (jvc=1)
  entry grdcrls1(ndim,s,rhs,tp,tc,tm,t0,fsing,
x  dr,nnr1,rg,dz,nnz1,itt,it,snglstp,srstor,
x  err,ier,etest,key,kn,bv,bvr,bvz,omg1stp,iperodz,jvc)
C=====MOVLEV=====
  subroutine movlev(a,b,n)

*precomp tpic ptpic
*cft i=ptpic,b=b8,l=lgr8f
*destroy tpic ptpic

```

VII.C) A Test Case: The LBL Source

We give here a typical input deck for an LBL source simulation. The first line is an ASCCI label that is printed across the top of all plot frames. Next is the namelist input for namelist NDATA, followed by namelist GSET, and finally, input for namelist NECHSP once for each of the two species.

```

box z15 ---- GYMNOs      LBL Source 12/18/90
      rmax=10.00762 rmin= 9.99746 zmin=0. zmax=.00635 iperodz=0
      idcode=4hgymn irntyp=4hrlbl istate=3 r0=.0025 z0=.25
      phistr(2,1)=-3.05 phistr(2,2)=2.e-8 ntrnlbc=1
      phistr(5,0)=-.3000 phistr(5,1)=.2934 phistr(5,2)=2.e-8
      dt=.025e-9 tmax=100.e-9 nrest=+1000
      ipplt=200 icvplt=200 ihist=1000

```

```

nspchk=2      jstgnsw=-1
$          nstrmx=5      nstruct(5,1)=8h      wire      strgen(5,1)=1.
strgen(5,2)=9.9993      strgen(5,3)=+0.00
strgen(5,4)=.001905      strgen(5,5)=+0.0      strgen(5,6)=360.
$          np= 6000      npmax=15000      namesp=8h      ions
sdemnx=1.5e12      sdemmn=1.e5      stemin=0.
pchrg=+4.80e-10      pmass=1.992e-23      sdithrl=1.      jestatic=1
nstprp(1)=250      nstprp(2)= 00      nstprp(3)= 51      nstprp(4)= 51      nstprp(5)= 00
sbeam(1,1)=1.5e12      sbeam(1,2)=6.0e5      sbeam(1,3)=0.e0      sbeam(1,4)=1.
$          np= 6000      npmax=15000      namesp=8helectrns
sdemnx=1.5e12      sdemmn=1.e5      stemin=7.
pchrg=-4.80e-10      pmass=9.1095e-25      sdithrl=1.      jestatic=1
nstprp(1)=100      nstprp(2)= 00      nstprp(3)= 51      nstprp(4)= 51      nstprp(5)= 00
sbeam(1,1)=1.5e12      sbeam(1,2)=0.0e5      sbeam(1,3)=7.e0      sbeam(1,4)=1.
$
```

VIII) Summary and Conclusions

Most of the information in this document is in fact contained in the comments statements within the code. The FORTRAN in the code remains the ultimate reference. Plans for the immediate future certainly include the proof testing of the VPARMOV series and the implementation a fluid pusher extracted from a code written in DWH's previous life so that selected species (perhaps even neutral gases) can be represented by a fluid.

DATES	DATES
c GYMNOS (RZ PIC with arbitrary boundaries)	
c date: GYMNOS started July 14, 1988	
c main rough electrostatic code assembled, including	
c KORNT, TRAP, new buffering scheme, Thermal reemission	
c and field emission/injection, by November 1988	
c split BCSET/GEOMSET 6/89	
c has BOYD stuff in FLDEMT 12/13/89	
c have new perfect reflection in TRAP 7/16/89	
c have IOTTY switch have IDRUN default ="drun"	
c have restructured FeEMT Nject with LORD in TRAP and PBCSET	
c 2/6/90	
N rs RG by RGCC then rs RGC by RG 3/1/90 added D3 for temperature	
c new BIRTH 3/6/90	
c ...added inside corner info to KORNT 3/20/90	
c added fast CFL catcher Sept. 8, 1990 DWHewett GYM22	
c Included DJL's PARMOVR 9/13/90 (not used in this code yet DWH)	
c added DEMAND ion injection in PBCSET (again) 9/24/90	
c added vectorized VPARMOV, VBIRTH, DEATH,VTRAP in GYM28 12/26/90 dwh	

APPENDICES

Appendix 1.	geoMset.....	59
Appendix 2.	GRAPHICS.....	63
Appendix 3.	BIRTH.....	70
Appendix 4.	TYPICAL INPUT "DECKS"	72
Appendix 5.	WRSTART.....	75
Appendix 6.	EFFVOL.....	76
Appendix 7.	AUTOVLM.....	78
Appendix 8.	PARTICLE BUFFERING ROUTINES: SETB, GETB, RESETB.....	79
Appendix 9.	VDKR.....	83
Appendix 10.	VTRIK.....	89
Appendix 11.	VPARMOV/VTRAP/VBIRTH/DEATH.....	90
Appendix 12.	PARMOVR.....	96
Appendix 13.	Santa Fe Numerical Simulation Abstract.....	98
Appendix 14.	GYM28 SUMMARY SHEETS.....	102

Appendix 1. geoMset

```
C=====GEOMSET==*
subroutine geomset(npthisb,namesp,nstruct,strgen,nstrmx,
. jstgnsw,nstmax,dr,dz,nr2,nz2,rg,zg,kcij,rmax,rmin,kcorr,
. kcormx,skey,iperodz,kornt,key)
...
c      Set SKEY, KEY, KORNT to zero.
...
kcov=0      clse=.001*amin1(dr,dz)
do 150 nstr=1,nstrmx
nstruct(nstr,2)=kcov+1
if (abs(strgen(nstr,1)).eq.pi) then
c PIE STRGEN(NSTR,1/2/3/4/5/6) 1/r,zcen/radius/astr,stp (a=0 e_r)
do 120 j=1,nz1      do 120 i=1,nr1
delr=rg(i)-strgen(nstr,2)      delz=zg(j)-strgen(nstr,3)
close=sqrt(delr**2+delz**2) +clse
if (close.le.strgen(nstr,4)) then
ang=90.
if (delr.eq.0. .and. delz.lt.0.) ang=270.
if (delr.ne.0.) ang=180.*atan(delz/delr)/pi +360.
if (delr.lt.0.) ang=ang+180.
ang=amod(ang,360.)
if ((ang.ge.strgen(nstr,5) .and. ang.le.strgen(nstr,6))
.or.(close.lt.clse)) then
kcov=kcov+1   kcorr(i,j)=kcov
kcij(kcov,1)=i  kcij(kcov,2)=j
endif
endif
120    continue
else
c BAR STRGEN(NSTR,1/2/3/4/5/6) 2/r,str/r,zstp/hwidth
c      1st,     make sure STRGEN(nstr,2).le.STRGEN(nstr,4)
rtmp=strgen(nstr,2)      ztmp=strgen(nstr,3)
if (rtmp.gt.strgen(nstr,4)) then
strgen(nstr,2)=strgen(nstr,4)      strgen(nstr,4)=rtmp
strgen(nstr,3)=strgen(nstr,5)      strgen(nstr,5)=ztmp
endif
c      if infinite slope, change to horizontal "fat" line 8/1/89
if (strgen(nstr,4).eq.strgen(nstr,2)) then
strgen(nstr,2)=strgen(nstr,2)-strgen(nstr,6)
strgen(nstr,4)=strgen(nstr,4)+strgen(nstr,6)
strgen(nstr,3)=.5*(strgen(nstr,3)+strgen(nstr,5))
strgen(nstr,6)=strgen(nstr,5)-strgen(nstr,3)
```

```

        strgen(nstr,5)=strgen(nstr,3)
        slp=0.
    else
        slp=(strgen(nstr,5)-strgen(nstr,3))/(
            (strgen(nstr,4)-strgen(nstr,2)))
    endif
    vslp=1.e100*(rmax-rmin)    if (slp.ne.0.) vslp=-1./slp
    do 130 j=1,nz1           do 130 i=1,nr1
        rint=(strgen(nstr,3)-zg(j)+vslp*rg(i)
            -slp*strgen(nstr,2))/(vslp-slp)
        if ((rint+clse).ge.strgen(nstr,2) .and.
            (rint-clse).le.strgen(nstr,4)) then
            zint=zg(j)+vslp*(rint-rg(i))
            if (slp.eq.0.) zint=strgen(nstr,3)
            close=sqrt((rg(i)-rint)**2+(zg(j)-zint)**2) +clse
            if (close.le.strgen(nstr,6)) then
                kcor=kcor+1      kcorr(i,j)=kcor
                kcij(kcor,1)=i   kcij(kcor,2)=j
                endif
            endif
        endif
130     continue
        endif
        nstruct(nstr,3)=kcor
150     continue
        print 157, kcor, kcormx
157 format (' kcor, kcormx =',2i7)
        if (kcor.gt.kcormx) stop 'kcor big'
        if (kcor.lt.(.10*kcormx)) stop 'kcor sml'
c                 KCIJ is finished--now use to set SKEY
c     To get SKEY, run over structure corners and hit
c 4 surrounding cells.  If a cell center gets 4 hits..SKEY=-1.
        do 195 nstr=1,nstrmx
        hit=-.250001
{Note "anti-structure" for drilling a hole in a structure with}
        if (strgen(nstr,1).lt.0.) hit=-hit  {neg. STRGEN(NSTR,1)}
        do 190 kcor=nstruct(nstr,2),nstruct(nstr,3)
            skey(kcij(kcor,1) ,kcij(kcor,2) )=amax1(-1.2,
.             skey(kcij(kcor,1) ,kcij(kcor,2) )+hit)
            skey(kcij(kcor,1)+1,kcij(kcor,2) )=amax1(-1.2,
.             skey(kcij(kcor,1)+1,kcij(kcor,2) )+hit)
            skey(kcij(kcor,1) ,kcij(kcor,2)+1)=amax1(-1.2,
.             skey(kcij(kcor,1) ,kcij(kcor,2)+1)+hit)
190 skey(kcij(kcor,1)+1,kcij(kcor,2)+1)=amax1(-1.2,
.             skey(kcij(kcor,1)+1,kcij(kcor,2)+1)+hit)

```

```

do 192 jcen=1,nz2
skey( 1,jcen)=-1.001
192 skey(nr2,jcen)=-1.001
do 195 icen=1,nr2
skey(icen, 1)=iperodz*skey(icen,nz1) -(1-iperodz)*1.001
skey(icen,nz2)=iperodz*skey(icen, 2) -(1-iperodz)*1.001
do 195 jcen=1,nz2
195 skey(icen,jcen)=ifix(skey(icen,jcen))
c                                LOOKOUT for 1 cell wide plasma/vac channels!!!
c
c      SKEY determines surface orientation KORNT and KEY
do 200 j=1+iperodz,nz1      do 200 i=1,nr1
jp1=j+1      if (iperodz.eq.1 .and. j.eq.nz1) jp1=2
sumk=skey(i,j)+skey(i+1,j)+skey(i+1,jp1)+skey(i,jp1)
if (sumk.ne.0. .and. sumk.ne.-4.) then
  if (sumk.eq.-1.) then
    if (skey(i ,j ).eq.-1.) kornt(i,j)=+12
    if (skey(i+1,j ).eq.-1.) kornt(i,j)=+8
    if (skey(i+1,jp1).eq.-1.) kornt(i,j)=-12
    if (skey(i ,jp1).eq.-1.) kornt(i,j)=-8
  endif
  if (sumk.eq.-2.) then
    if (skey(i,j).eq.-1.) then
      kornt(i,j)=-skey(i,jp1)-10*skey(i+1,j)
    else
      kornt(i,j)=+skey(i+1,j)+10*skey(i,jp1)
    endif
  endif
  if (sumk.eq.-3.) then
    if (skey(i ,j ).eq.0.) kornt(i,j)=-11
    if (skey(i+1,j ).eq.0.) kornt(i,j)=-9
    if (skey(i+1,jp1).eq.0.) kornt(i,j)=+11
    if (skey(i ,jp1).eq.0.) kornt(i,j)=+9
  endif
  endif
200 continue
  if (iperodz.eq.1) then
    do 210 i=1,nr1
210      kornt(i,1)=kornt(i,nz1)
    endif
c                                Set KEY
do 255 j=1,nz2      do 255 i=1,nr2
if (skey(i,j).eq.-1.) then
  icm1=max0(i-1,1)      jcm1=max0(j-1,1)

```

```
    ic=min0(i,nr1)          jc=min0(j,nz1)
    key(ic,jc)==-1          key(icm1,jc)==-1
    key(ic,jcm1)==-1        key(icm1,jcm1)==-1
    endif
255 continue
    return
end
```

Appendix 2. Graphics

Almost all plot libraries have the basic routines needed: routines that draw and scale axes, line plotters (with linear-linear, log-linear, etc.) contour plotters, vector plotters and point plotters. GYMNOS uses a personal plot package maintained by DWH but a similar one (derived from this one in 1983 and extended by LANL CTR Division) is available. Our version itself evolved from an even earlier version written by many people in LANL in the early and middle 70's. Our bias (and therefore our advise) is to get by with minimal embellishments such as fancy labels. These codes use lots of CPU time and we want LOTS of pictures to build intuition. Only one in 10,000 pictures ever gets published but they all rob CPU cycles from the computational effort. Therefore, every effort is made to display as much information as cheaply and crudely as possible. Extra effort is expended only on those graphs intended for display. The plot package is capable of many things but, not surprisingly, certainly does not make publication quality pictures! Here is enough documentation to use the plot routines in GYMNOS.

```
C=====LPLLOT=====
subroutine lplot(imx,imy,npts,x,y,inc,iop,
1 ltitle,ntitle,lxname,nxname,lynname,nyname)
```

LPLLOT plots NPTS values of x and y and connects them with a line. It draws a box around the plot with scaling and places labels on the left and bottom axis.

The plot's position on the frame is determined by IMX, IMY. An IMY=1 specifies that the y-coordinate range spans a full frame; IMY=2,3 specify the upper,lower halves of the frame respectively; and IMY=4,5,6 specify the upper,middle, lower thirds of the frame. IMX=1 specifies that the x-coordinate range spans a full frame, & IMX=2,3 specify the left, right halves of the frame. IMX=iabs(4), IMY=1 gives a square plot area--useful if the X range and Y range are equal to see proper aspect ratio. IMX=5,6 are not allowed. For example, (imx,imy) = (1,1) specifies a plot filling the full frame, and (3,3) specifies a plot in the lower right-hand quadrant. Frame advance is automatic with the first plot that extends into the upper left-hand corner of the frame. Such a plot must be the first in any plot sequence intended to appear on one frame. The plot's range usually is expanded to a "round" decimal number by the automatic scaling routines from the minimum range implied by the data. Expansion may be prevented by appending a minus sign to either/both IMX and IMY in any plot call. For example, if xmin=0.17, xmax=359.78, and IMX is positive for automatic scaling, the x-axis scale goes from 0.0 to 400.0. If IMX is negative for exact scaling, the x-axis scale goes from 0.15 to 359.78 with "rounded" tick marks at the correct positions.

-----Summary of IMX,IMY positions

	1,4		2,4	3,4
1,2		2,2 3,2		
1,1	1,5	2,1 3,1	2,5	3,5
(4,1) 1,3			2,3 3,3	
(square)	1,6			2,6 3,6
4,1 gives square full frame				

IMX, IMY < 0 gives exact scaling; > 0 gives automatic scaling.
 |NPTS| The number of y values to be plotted. NPTS<0 draws a curve onto a frame previously setup by LPLOT with IOP<0.
 X(Y) is the table of abcissa(ordinate) values to be plotted.
 |INC| The spacing between successive X and Y elements to be plotted. INC<0 X(1)=Xmin and X(2)=DX
 eg:---If we want to plot 20 points separated by 3 locations with x(1)=Xmin and x(2)=DX, we chose NPTS=20,INC=-3 so that the code will search through 60 locations in order to find the 20 elements to plot.
 IOP > 0 scales and draws frame and plots data from X and Y.
 < 0 draws frame and scales but does not draw curve.
 $= 1000 * N3 + 10 * N2 + N1.$

----- significance-----
 N1 = 1 linear X-axis and linear Y-axis
 N1 = 2,3,4 linear-log, log-linear, log-log
 N2 Indicates the decimal equivalent of the character to be plotted at the data point. Default is no symbol.
 N3 = 0 The points are not connected by lines.
 0<N3<9 A symbol is placed at the first and every n3 pts.
 Other points are connected by lines.
 LTITLE is the title for the graph.
 LX(Y)NAME is the label for the X(Y)-axis.
 NTITLE, NX(Y)NAME is the number of characters in lttitle,LX(Y)NAME, maximum of 16.
 NYNAME < 0 scales and tick marks for plot not drawn.

```

C=====CPLOTX=====
subroutine cplotx(imx,imy,nx,ny,x,y,incx,incy,ndim,z,nc,zc,
1 ltitle,ntitle,lxname,nxname,lyname,nyname,rmax,iquad,lgz)

IMX(IMY)      (see LPLOT internal documentation)
|NX|      The number of points in X; NX <= second dimension of Z.
          NX < 0 draws a frame and scaling only for a plot.
|NY|      The number of points in Y; NY <= first dimension of Z.
          NY < 0 draws contours onto a previously drawn frame,
          possibly created by an earlier call to CPLOT with NX<0.
X(Y)      is the table of abscissa(ordinate) values.
|INCX(Y)|   The skip parameter in a row(column).
          INCX < 0 X(1) = xmin and X(2) = hx. (Similar for Y.)
NDIM      Length of a row in Z. (1st arg dimensn in 2-d arry)
Z         is the 2D function to be contoured; Z should be stored
          so that Z(i,j) is the value of Z at X(i), Y(j).
NC        is the number of contours to be plotted; maximum of 26.
          NC < 0 CPLOT automatically fills ZC with NC values.
          NC > 0 ZC is supplied by the user. Values must be
          stored in increasing order in ZC.
ZC        is the table of contour values.
RMAX      is the max radius of a polar plot.
          = 0, do a cartesian plot.
          > 0, assume X,Y correspond to r,theta(in radians).
          < 0, assume X,Y correspond to r,cos(theta).
IQUAD    for RMAX.ne.0, IQUAD gives the total # of quadrants.
LGZ       =0, scalar contours.
          !=0 gives log10 contours at 10,2,-LGZ intvl/dcde.
          Auto ovride down to 1 intvl/dcde dpdng on # dcdes.
          Neg LGZ stops auto ovride (limits dcdes).
EXTRA LABELS: On upper left is input LGZ and
               number of decades ploted LDEC. On
               upper right is the max(Z) and power
               of 10 of smallest decade plotted.
LTITLE     is the title for the graph.
LX(Y)NAME   is the label for the X(Y)-axis.
NTITLE, NX(Y)NAME is the number of characters
               in LTITLE,LX(Y)NAME, maximum of 16.
NTITLE < 0  no alphabetic labels on contours
NYNAME < 0  scales and tick marks are not drawn.

```

```

C=====VPLOTX=====
subroutine vplotx(imx,imy,nx,ny,x,y,incx,incy,ndim,zx,zy,
1 ltitle,ntitle,lxname,nxname,lyname,nyname,rmax,iquad)

IMX(IMY) (see LPLOT internal documentation)
|NX| the number of points in X; NX <= 2nd dimensn of ZX,ZY.
    NX < 0 draws a frame and scaling only for a plot.
|NY| the number of points in Y; NY <= 1st dimensn of ZX,ZY.
    NY < 0 draws vectors onto a previously drawn plot
        created by a call with NX < 0.
X(Y) provide the monotonic abscissa(ordinate) values.
    Abscissa(ordinate) values are expected from
    i=1,NX,INCX(j=1,NY,INCY).
    If INCX(INCY)<0 then
        X(1) = Xmin and X(2) = dx*iabs(INCX)
        Y(1) = Ymin and Y(2) = dy*iabs(INCY)
    INCX(Y) the skip parameter in a row(column).
NDIM the length of a row in y. (1st arg dimensn in 2-d arry)
ZX,ZY are the two-dimensional components to be vector plotted;
    be stored so that ZX,ZY(i,j) is the value
    of ZX,ZY at X(i), Y(j).
RMAX is the max radius of a polar plot.
    = 0, do a cartesian plot.
    > 0, assume X,Y correspond to r,theta(in radians).
    < 0, assume X,Y correspond to r,cos(theta).
IQUAD for RMAX.ne.0, IQUAD gives the total # of quadrants.

LTITLE is the title for the graph.
LX(Y)NAME is the the label for the X(Y)-axis.
NTITLE, NX(Y)NAME is the number of characters
    in LTITLE,LX(Y)NAME, maximum of 16.
    NTITLE < 0 no vector if both components < mx(du,dz)/10
    NXNAME < 0 no arrowheads are drawn.
    NYNAME < 0 scales, tick marks for plot are not drawn.

```

```
C=====APLOTEX=====
      subroutine aplotx(s,ndim,avy,avx,xsc,ysc,mxya,myya,mxxa,myxa,
     1 nx,ny,nft1,nft2,ltitly,ntitly,ltitlx,ntitlx,ixa,iya,irz)
```

APLOTEX(Y) averages over a matrix of values in one direction and plots the results with respect the other direction. This subroutines only performs the averaging calculation; it calls LPLOT to draw the resulting curve.

S is the 2D table of values 1st dimensioned NDIM.

AVX(AVY) is array that holds the average of S in the x(y)-direction. It is dimensioned NX(Y) in user's program. If a user wants only average returned from the APLOT, set IXA(IYA) = 0. XSC(YSC) is the x(y)-axis information. (i.e. XSC(1)=Xmin,XSC(2)=DX)

MXXA,MYXA,MXYA,MYYA

indicates the location and length of the x(y)-axis.

the meaning is the same as in LPLOT.

(i.e. MXXA,MYXA = IMX,IMY for the xavg plot)

(see LPLOT internal documentation)

< 0 gives exact scaling; > 0 gives automatic scaling.

NY(X) is the (usual) number of points in the y(x) average.

NFT1 and NFT2 specifies a band of points to average over--X-direction only.

NFT1 is i index of the 1st cell in the average calculation.

The value for averaging over the whole function is 1.

NFT2 is # of cells in the band to average over. The value for averaging all cell in the x-direction is NX.

NX(Y) < 0 Vertical scale fixed by previous call to LPLOT.

LTITLX(Y) title for the average plot in the x(y)-direction.

NTITLX(Y) # of characters in LTITLX(Y); max of 16.

IXA(Y) = 1, average plot in x(y)-direction, drawn by LPLOT.

= 0, average made not plotted, returned in AVX(Y).

=-1, neither an average nor a plot is made.

IRZ =0(1) label graphs x(r)-avg and y(z)-avg.

```
C=====PPLOTC=====
```

```
    subroutine pplotc(imx,imy,npts,x,y,inc,z,zmin,zmax)
```

PPLOT plots values in X and Y. Each point is plotted as dot, adjacent points are not connected. PPLOTC plots the point only if its Z < ZMAX and > ZMIN. Both subroutines assume that the frame, scale and labels for this IMX and IMY plot have been generated by LPLOT with IOP = -1. Only linear-linear scaling is allowed. NPTS is the number of elements in X, Y, and Z.

IMX(IMY) (see LPLOT internal documentation)

IMX, IMY < 0 gives exact scaling; > 0 gives automatic.

NPTS is the number of elements in the arrays X, Y, and Z.

X(Y) is the table of abcissa(ordinate) values.

|INC| The spacing between successive X and Y elements.

INC < 0 X(1) = Xmin and X(2) = DX

Z is a function of X and Y.

ZMIN(ZMAX) is the smallest(largest) value plotted.

Typically, a user encodes the TIME his code has reached and uses LBLBOT to write it at the bottom of the frame.

```
C=====LBLTOP=====
```

```
    subroutine lbltop(label,nlabel)
```

LBLTOP enables a user to specify an 80-character label at the top of a page of plots. LBLBOT enables a user to specify a 40-character label at the bottom of a page of plots. Labels are centered at the top and bottom of the page. Subroutines should be called before any plot calls on a page. Label calls remain in effect for each succeeding page until another label call with a different character string is given. e.g.

```
call lbltop ('plasma physics plotting package', 31)
```

```
C=====BEGPLT=====
```

```
    subroutine begplt (iname,ntitle,ltitle)
```

BEGPLT initializes the plotting routines; must be called before any plotting is done. Creates a file named f3abcd0x in the user's local file space on CTSS. The a, b, c, and d in the name are 4 characters specified by user in INAME. e.g. If INAME='abcd', graphics output is written into f3abcd0x. If additional file space needed, new graphics file is created by incrementing the 4th letter as follows: f3abce0x, f3abcf0x, etc. For unique eye-readable microfiche title, call the fortlib routine, SETBOX, before BEGPLT.

e.g. call setbox('box 130 reversed field pinch 1', 30)

The string must start with your box number and the following 23

characters can be anything you choose as a unique identifier.

e.g. call begplt(4habcd,6,6hybrid)

will make a plot file "f3abcd0x" with "hybrid"

following "flm-only" on the top of the resulting fiche.

=====BPLT====*

```
subroutine bplot(kornt,mstr,nstruct,kcij,nr,nz,
    rmin,dr,zmin,dz,ndim,nstmax,mxgd,nstr1,nstr2)
```

c Plots the edges of NSTR=NSTR1,NSTR2 internal boundaries.

c MSTR is an integer work array.

c Use to plot internal boundaries on an existing frame.

c DWH,DJL 9/8/89

Appendix 3. BIRTH

BIRTH is the routine that creates new particles during the course of a run. It is called only from routine TRAP during the processing of buffers by PARMOV as directed by TRANS. BIRTH creates NPART particles just before they experience the first order velocity extrapolation and then are integrated or "accumulated" into the source terms of their species N.

The new particles are created at location REMT.ZEMT with a thermal spread given by VMP, the most probable thermal velocity. BIRTH orients the normal velocity magnitude UNORM using the values COSA.SINA (called SGNR,SGNZ in TRAP) when called by the emit/inject section and using KORNT values for the orientation of the emitting corner if the thermal reemission section is the calling routine.

```

call birth(npart,vmp,unorm,evnvth,evtvth,kornt(icor,jcor),
           edithr,edithz,sgnr,sgnz,rg(icor),zg(jcor))

c=====
c====BIRTH=====
subroutine birth(npart,vmp,unorm,evnvth,evtvth,krntflg,
               edithr,edithz,cosa,sina,remt,zemt)
parameter (bit=1.e-11)
use CREDIT PRCOM CELCOM SORCOM FLDCOM GECOM PHCOM NCOM
common / psh/ npthisb,namesp,n
c                                     D.W.Hewett 4/20/91
c KRNTFLG=0 for LORc njctn/fldemt; =KORNT(IJCOR) for cornr Thremt
knz=krntflg/8 knr=krntflg-knz*10 knp=knz*min0(1,max0(-1,knr))
lstar=npthisb*dimp+1   lstop=lstar+(npart-1)*dimp
do 600 lp=lstar,lstop,dimp
  vmag=vmp*sqrt(-alog(1.-.999999*ranf())) theta=pi*(ranf()-.5)
  vn=vmag*evnvth*cos(theta)+unorm vt=vmag*evtvth*sin(theta)
  vx(lp)=vn*cosa-vt*sina          vz(lp)=vn*sina+vt*cosa
  theta=pi2*ranf()
600  vy(lp)=vt*cos(theta)

c
c1 only need one RANF for the dither location even if corner
do 700 lp=lstar,lstop,dimp
  r(lp)=remt+sina*edithr*(ranf()-.5)+bit*cosa
700  z(lp)=zemt-knp*dz*dri*(r(lp)-remt)+bit*sina
     +(1-iabs(knp))*cosa*edithz*(ranf()-.5)

c
c2 now make sure this is OK for inside corners
if (iabs(knr).eq.1 .and. iabs(knz).eq.1) then
  rbit=knr*(remt+bit*cosa)      zbit=knz*(zemt+bit*sina)
  do 710 lp=lstar,lstop,dimp

```

```

    vx(lp)=knr*abs(vx(lp))      vz(lp)=knz*abs(vz(lp))
    r(lp)=knr*amax1(knr*r(lp),rbit)
710   z(lp)=knz*amax1(knz*z(lp),zbit)
      endif
c  we're SURE particles are now inside
c      & we're sure inside corners only have proper velocities
c
c3 now add normal and xvers velocities
    do 715 lp=lstar,lstop,dimp
        r(lp)=r(lp)+vx(lp)*dt*ranf()
715   z(lp)=z(lp)+vz(lp)*dt*ranf()

c
c4 if these velocities too large, we still must check SKEY
c  If we're out, it must have been the VX or VZ of the last step
    lp=lstar-dimp
720 lp=lp+dimp
730 if (lp.gt.lstop) go to 800
    icen=dri*(r(lp)-rmin)+2.          jcen=dzi*(z(lp)-zmin)+2.
    icen=min0(max0(icen,1),nr2)       jcen=min0(max0(jcen,1),nz2)
    if (skey(icen,jcen).gt.-1.) go to 720
      Have found one!
npart=npart-1
print 763, r(lp),z(lp),vx(lp),vz(lp)
write (7,763) r(lp),z(lp),vx(lp),vz(lp)
763 format(' BRTHfast r,z,vx,vz',4(1pe9.2))
    r(lp)=r(lstop)      z(lp)=z(lstop)
    vx(lp)=vx(lstop)    vy(lp)=vy(lstop)    vz(lp)=vz(lstop)
    lstop=lstop-dimp    go to 730
800 nspec(n)=nspec(n)+npart      npthisb=npthisb+npart
cfocus           species 3 dwh 4/25/90
      if (npart.gt.0 .and. irntyp.eq."b x ch" .and. n.eq.3) then
        lstop=lstar+(npart-1)*dimp
        do 900 lp=lstar,lstop,dimp
900   vx(lp)=vx(lp)-r(lp)*vz(lp)/(pointz*zmax-zmin)
      endif
      return           end

```

Appendix 4. Typical Input "Decks"

```

*file name=idxch
box z15 ---- GYMNOS      beam x chamber 7/14/90
    rmin=0.   rmax=5.0   zmin=0.     zmax=100. iperodz=0
    idcode=4hgymn  irntyp=6hb x ch    istate=1  r0=.0025  z0=.25
    pointz=.75
    phistr(1,1)=0.   phistr(2,1)=-00.0      ntrnlbc=1
    dt=.025e-11   tmax=11.e-9   nrest=+2000
    ipplt=250    icvplt=250   ihist=500
    nspchk=3     jstgnsw=-1
$

nstrmx=5      nstruct(5,1)=8hbeam ldr      strgen(5,1)=2.
strgen(5,2)=0.          strgen(5,3)=0.
strgen(5,4)=4.0        strgen(5,5)=0.      strgen(5,6)=.1e-2
$

np=18000   npmax=18000  namesp=8h      ions
sdenmx=1.0e14 sdenmn=1.e8   stemin=10.
pchg=-4.80e-10 pmass=5.01e-24   sdithrl=1.   jestatic=1
nstprp(1)= 51 nstprp(2)= 00 nstprp(3)= 51 nstprp(4)= 50 nstprp(5)= 51
nstprp(1)= 51 nstprp(2)= 50 nstprp(3)= 51 nstprp(4)= 50 nstprp(5)= 51
$

np=18000   npmax=27000  namesp=8helectrns
sdenmx=1.0e14 sdenmn=1.e8   stemin=10.
pchg=-4.80e-10 pmass=9.1095e-28   sdithrl=1.   jestatic=1
nstprp(1)= 51 nstprp(2)=100 nstprp(3)= 51 nstprp(4)= 50 nstprp(5)= 51
sbeam(2,3)=100.      sbeam(2,4)=.7071068
$

np=0      npmax=15000  namesp=8hbem ions
sdenmx=0.0e13 sdenmn=1.e8   stemin=1.e6
pchg=-4.80e-10 pmass=3.34e-22   jestatic=1
sbeam(5,1)=6.e10   sbeam(5,2)=1.0e10  sbeam(5,3)=1.e6   sbeam(5,4)=1.
sbeam(5,5)=0.e-9   sbeam(5,6)= 0.e-9  sbeam(5,7)=10.e-9
nstprp(1)= 51 nstprp(2)= 00 nstprp(3)= 51 nstprp(4)= 00 nstprp(5)=251
$

HIF Beam X Target Chamber
c PIE STRGEN(NSTR,1/2/3/4/5/6)   1/rcen/zcen/radius/astr/astp (a=0 e_r)
c BAR STRGEN(NSTR,1/2/3/4/5/6)   2/rstr/zstr/rstp/zstp/hwidth
c   sBEAM(NSTR,1/   2/   3/   4/   5/   6/   7/   8)
c           Bden,unbstr,temstr,etnvtp, turnon,rise,turnof,fall
c           VOLT(NSTR,R/Q/Z,1/2/3/5) mag,turnon,rise,turnof,fall
c           PHISTR(NSTR,0/1/2/3/4/5)   mag0,mag,turnon,rise,turnoff,foff
c           NSTPROP(NSTR,NSPECIES)--N00-N50, N51--%THREMT,PRFLCT
c           (N=1/2/0 FEMT/NJCT/NEITHER), NEGATIVE=NO DITHR
c           NSTPROP(NSTR,Nspecies)--n00-n50, n51--%ThRemt,PRFLCT

```

```

c      (n=1/2/0  Femt/Njct/Neither), negative=no dithr

*file name=inh2s
box z15 ---- GYMNOs    1Dinhom T-S      2/08/90
      rmin=0.499   rmax=0.5   zmin=0.   zmax=.4   iperodz=0
      idcode=4hgymn  irntyp=6hnhom2s  istate=2  r0=.0025  z0=.25
      phistr(1,1)=0.  phistr(2,1)=-00.0      ntrnlbc=1
      dt=.010e-12  tmax=10.e-12  nrest=+1000
      ipplt=500   icvplt=500   ihist=2000
      nspchk=4     jstgnsw=-1

$      nstrmx=5      nstruct(5,1)=8hbeam ldr      strgen(5,1)=2.
      strgen(5,2)=0.      strgen(5,3)=+0.
      strgen(5,4)=77.0     strgen(5,5)=+0.0     strgen(5,6)=.1e-2

$      np=2000   npmax=2000  namesp=8h      ions
      sdenmx=5.0e14  sdenmn=1.e8   stemin=100.
      pchrg=+3.84e-8  pmass=3.34e-22  sdithrl=0.   jestatic=1
      nstprp(1)= 51 nstprp(2)= 51 nstprp(3)= 50 nstprp(4)= 50 nstprp(5)= 50

$      np=2000   npmax=2000  namesp=8helectrns
      sdenmx=4.0e16  sdenmn=1.e8   stemin=100.
      pchrg=-4.80e-10 pmass=9.1095e-28  sdithrl=0.   jestatic=1
      nstprp(1)= 51 nstprp(2)= 51 nstprp(3)= 50 nstprp(4)=-100 nstprp(5)= 50

$      np=0      npmax=5000  namesp=8hbem ions
      sdenmx=0.0e13  sdenmn=1.e8   stemin=1.e6
      pchrg=+9.60e-9  pmass=3.34e-22  jestatic=1
      sbeam(5,1)=6.e13   sbeam(5,2)=1.0e10  sbeam(5,3)=1.e6   sbeam(5,4)=1.
      sbeam(5,5)=0.e-9   sbeam(5,6)= 0.e-9   sbeam(5,7)=10.e-9
      nstprp(1)= 51 nstprp(2)= 51 nstprp(3)= 50 nstprp(4)= 00 nstprp(5)=-251

$      np=0      npmax=5000  namesp=8hbem elec
      sdenmx=0.0e15  sdenmn=1.e8   stemin=100.
      pchrg=-4.80e-10 pmass=9.1095e-28  jestatic=1
      sbeam(5,1)=1.2e15   sbeam(5,2)=1.0e10  sbeam(5,3)=100.  sbeam(5,4)=1.
      sbeam(5,5)=0.e-9   sbeam(5,6)= 0.e-9   sbeam(5,7)=10.e-9
      nstprp(1)= 51 nstprp(2)= 51 nstprp(3)= 50 nstprp(4)= 00 nstprp(5)=-251

$      Inhomogeneous Target Two Stream

*file name=iatsa
box z15 ---- GYMNOs    LBL Source 12/18/90
      rmax=10.00762  rmin= 9.99746  zmin=0.  zmax=.00635   iperodz=0
      idcode=4hgymn  irntyp=4hrlbl  istate=3  r0=.0025  z0=.25

```

```

phistr(2,1)=-3.05    phistr(2,2)=2.e-8      ntrnlbc=1
phistr(5,0)=-.3000   phistr(5,1)=.2934   phistr(5,2)=2.e-8
dt=.025e-9     tmax=100.e-9      nrest=+1000
ipplt=200    icvplt=200      ihist=1000
nspchk=2     jstgnsv=-1

$  

nstrmx=5       nstruct(5,1)=8h    wire      strgen(5,1)=1.  

strgen(5,2)=9.9993      strgen(5,3)=+0.00  

strgen(5,4)=.001905      strgen(5,5)=+0.0      strgen(5,6)=360.  

$  

np= 6000    npmax=15000  namesp=8h    ions  

sdemnx=1.5e12  sdemmn=1.e5  stemin=0.  

pchrg=-4.80e-10  pmass=1.992e-23  sdithrl=1.  jestatic=1  

nstprp(1)=250  nstprp(2)= 00  nstprp(3)= 51  nstprp(4)= 51  nstprp(5)= 00  

sbeam(1,1)=1.5e12  sbeam(1,2)=6.0e5  sbeam(1,3)=0.e0  sbeam(1,4)=1.  

$  

np= 6000    npmax=15000  namesp=8helectrns  

sdemnx=1.5e12  sdemmn=1.e5  stemin=7.  

pchrg=-4.80e-10  pmass=9.1095e-25  sdithrl=1.  jestatic=1  

nstprp(1)=100  nstprp(2)= 00  nstprp(3)= 51  nstprp(4)= 51  nstprp(5)= 00  

sbeam(1,1)=1.5e12  sbeam(1,2)=0.0e5  sbeam(1,3)=7.e0  sbeam(1,4)=1.  

$  


```

Included below some of the parameter statements used the code when running with the above input decks.

```

c          (prtcl parameters and buffer arrays)           PRCOM
Inhomogeneous two stream runs NH2S
c          parameter (nheadb=10,mpbuf=1000,dimp=5,memoly=71000,nsp=4) NH2S
Ion source runs RLBL
c          parameter (nheadb=10,mpbuf=1000,dimp=5,memoly=151000,nsp=2) RLBL
Beam across chamber runs BXCH
c          parameter (nheadb=10,mpbuf=1000,dimp=5,memoly=301000,nsp=3) BXCH

c          (cell parameters and work arrays)           CELCOM
c          parameter (nr2=3,nz2=2001,mxg1d=2001,nstmax=8,ithmx=250,
c .          kcormx=4385)    NH2S
c          parameter (nr2=19,nz2=73,mxg1d=73,nstmax=8,ithmx=2000,
c .          kcormx=250)    BXCH
c          parameter (nr2=146,nz2=19,mxg1d=146,nstmax=8,ithmx=2000,
c .          kcormx=700)    RLBL

```

Appendix 5. WRSTART

The RESTART subroutine will write WSTART a restart file that contains everything needed to restart RSTART from that file (given the original input deck) and continue the run as if the interruption had never occurred. This is almost a requirement for a production code, that may run many hours, to allow decision points about the need to run further, to protect against machine failures, to allow the addition of more diagnostics, or change the frequency of the plots. Restarts are nearly indispensable for debugging problems that run significant CPU time before they experience difficulty.

In the routine RESTART, there are entries for both reading and writing these files. As you can see from the comments, the routine automatically provides two layers of backups as a hedge against losing everything—requiring the run be started over from scratch. The variables that are saved in these files can be adjusted by moving them in or outside of the locator flags that are used for the starting and stopping points of the BUFFER IN/OUT routines used for the variables in common blocks. For example, all variables stored between MN1First and MN1Last are included in the restart file and thus CANNOT be changed even though they may appear in the input deck. (Such variables can be changed only dynamically with DDT.)

```
subroutine restart(mn1f,mn1l,mn2f,mn2l,mn3f,mn3l,mn4f,mn4l,mn5f,
      mn5l,mn6f,mn6l,ipbuf,iparmem,nbspec,mspec,nspec,numrest,nsp)
dimension ipbuf(1),iparmem(1),nbspec(nsp),mspec(nsp),nspec(nsp),
      mn1f(1),mn2f(1),mn3f(1),mn4f(1),mn5f(1),mn6f(1)
character*8 rfile, rrfile, idum
data rfile,rrfile /"r","rr"/
c-----hewett-----5/4/89-----
c STRATEGY: create & write new RRFILe file and then exchange names
c   with latest RFILE.  R(RR)FILEs may be stored during the run.
c   NUMREST is always the # of NEXT restart file you will write.
c INSTRUCTIONS: To WRITE a restart file....mn1f=nheadb $ mn2f=dimp
c mn3f=memoly $ mn4f=mpb $ mn5f=idrun $ mn1l=nr2 $ mn2l=nz2 $ mn3l=it
c call wstart(mn1f,mn1l,mn2f,mn2l,mn3f,mn3l,mn4f,mn4l,mn5f,mn5l,
c   . mn6f,mn6l,ipbuf,iparmem,nbspec,mspec,nspec,numrest,nsp)
c   To RESTART, need restart named RFILE in local file space &
c   mn1f=nheadb $ mn2f=dimp $ mn3f=memoly $ mn4f=mpbuf $ mn5f=idrun
c   idum=idrun  call rstart(mn1f,mn1l,mn2f,mn2l,mn3f,mn3l,mn4f,mn4l,
c   . mn5f,mn5l,mn6f,mn6l,ipbuf,iparmem,nbspec,mspec,nspec,numrest,
c   . nsp) $ mpb=mn4f $ nchg=it $ print 89, idrun,dt,newdt
c 89 format(" idum,olddt,newdt ",a8,2e14.7)
c   if (idrun.ne.idum) then $ print 91, idrun, idum
c 91  format(" Restart trouble.  Expecting restart idrun ",a8/
c   . "and got ",a8," instead.") $ call exit(1) $ endif
c   print 95, numrest,idrun,it
c 95 format(' Successful restart... numrest,idrun,it ',i7,a8,i7)
```

Appendix 6. EFFVOL

This section discusses the conversion of the "raw particle counts" that PARMOV accumulates at cell corners for each species into particle and current densities. There are two important issues here—one is a simple geometric consideration and the other is a subtle r-dependent systematic error. The systematic error is due to a particle near the small r side of a cell volume contributing the same amount to the cell density as a particle near the large r side of a cell volume. In the limit of a infinite number of particles to represent the plasma, the density loss by particles on the inside is just cancelled by the density gained by those on the outside. The trouble is that on a boundary we only have one type of particle or the other. In particular, at $r=R_{\text{MIN}}$ where the error is largest, we only have the outside particles and this systematic error gives rise to density spikes on the axis, if uncorrected. The correction factors, computed in the limit of an infinite number of particles by integration, reduce to the simple forms

$$RFACOT = \frac{3(4RG(I) + DR)}{4(3RG(I) + DR)}$$

for cell corners with $LR=+1$ and

$$RFACIN = \frac{3(4RG(I) - DR)}{4(3RG(I) - DR)}$$

for cell corners with $LR=-1$.

The geometric corrections are applied in addition to the systematic corrections. The cell corners, where raw counts are accumulated, may be in the 1) plasma region, 2) in a boundary, or 3) on a boundaries. If 1), just divide by the volume of the associated cell, VOL(I). If 2) it doesn't matter, there will be no particles. If 3) we have to consider further possibilities.

For a cell corner on a boundary, we must decide the orientation of the corner. KORNT(I,J) contains all the required information and it is decoded into unit "vectors" LR and LZ that are $+(-)1$ if the direction *into* the plasma region is in the $+(-)$ direction. If we are simply on a edge, say $LR=0$ and $LZ=+1$ for a point on the bottom or Z_{MIN} boundary, then we wish to divide the particle count at that corner by one half the normal cell volume at that radius VOL(I). If the point is $LR=+1$ and $LZ=0$, then we must divide by the outside part of the cell volume. Similar considerations apply for cell corners but now we must distinguish between "inside" and "outside" corners. In the case of an outside corner, the cell volume is roughly $3/4$ of VOL(I) and only roughly $1/4$ for the inside corner. Of course the usual r corrections are employed to correct for the outside cell volume being slightly larger than the inside cell volume.

Now that the reader knows what to expect, it should be easy to pick out the various parts of the effective volume EFFVOL calculation in the Fortran below.

```
...
c      Now correct VOLCs near boundaries and fix raw counts.
      do 360 i=1,nr1
      rfacot=.75*(4.*rg(i)+dr)/(3.*rg(i)+dr)      (Systematic
```

```

rfacin=.75*(4.*rg(i)-dr)/(3.*rg(i)-dr)      correction)
do 360 j=1,nz1
lz=kornt(i,j)/8    lr=kornt(i,j)-lz*10
jotsdcr=lr/2        lr=lr-jotsdcr   jotsdcr=iabs(jotsdcr)
c     LZ\LR=1,-1 if pt. on Z\R Bdy else 0
c     LRP=1 if LR=1    JOTSDCR=1 if "outside" corner
c     /|+11          ----+          +9|/          +----+
c     /+----          //||           -----+/-       |/////
c     /////           +12|/          /////          |/+8
c                                     KORNT
c     |-12            /////          -8|/          /////
c     |///             -----+/-       //||           /+-----
c     +----           -11|/          -----+/-       /|-9
c VOL's have Rmin,max adjustments built in (though not used here!)
c     EFFVOL are here adjusted to account for nearby Bs.
lrp=.5*(1+lr)
rfac=lrp*rfacot+(1-lrp)*rfacin
effvol=(1-.5*iabs(lz))* ((1-iabs(lr))*vol(i)
c     . +iabs(lr)*pi*drh*dz*(rg(i)+rgcc(i+lrp)))*rfac  bug dj1 5/91
c     . +iabs(lr)*pi*drh*dz*(rg(i)+rgcc(i+lrp))/rfac
effvol=effvol+.5*vol(i)*jotsdcr    veffvol=1./effvol
den(i,j,n)=den(i,j,n)*rppsp(n)*veffvol
ur/q/z(i,j,n)=ur/q/z(i,j,n)*rppsp(n)*veffvol
...
360 continue

```

Appendix 7. AUTOVLM

The AUTOVLM provides automatic scaling for the particle phase plots during the course of a run. The plan is to store in VZLIM(1,N) the smallest VZ of *any* particle of species N on the time step such prior to the time on which the phase plots are made.. VZLIM(2,N) The contains the largest VZ, and so on for the VR and VQ components. AUTOVLM provides a 5during the time step on which the plotting actually occurs. Several options exist and are coded into the parameter AUTOVLM. Options available are given in the comments. This coding is all found in TRANS though there is some default initialization in INIT not shown here.

```
c           Finish a species  (in TRANS)
c   Set up VLIM's for prtcl plots NEXT time step.
200 if (mod(itp1,ipplt).eq.0) then
    vr/q/zlim(1,n)=vr/q/zmn-.05*(autovlm+1.)*(vr/q/zmx-vr/q/zmn)
    vr/q/zlim(2,n)=vr/q/zmx+.05*(autovlm+1.)*(vr/q/zmx-vr/q/zmn)
c   t V limits if more than 1 species to be on same V plot.
if (nold.ne.-7.and.namsp(n).eq.namsp(nold).and.
    sq(n).eq.sq(nold).and.sm(n).eq.sm(nold)) then
    vr/q/zlim(1,n)=amin1(vr/q/zlim(1,nold),vr/q/zlim(1,n))
    vr/q/zlim(2,n)=amax1(vr/q/zlim(2,nold),vr/q/zlim(2,n))
    vr/q/zlim(1,nold)=vr/q/zlim(1,n)
    vr/q/zlim(2,nold)=vr/q/zlim(2,n)
endif
endif
.....
c           Start a new species  (in TRANS)
400 npthisb = ipbuf(2)
.....
if (mod(it,ipplt).eq.0) call dparts
AUTOVLM auto phase space (Vx-x) plot scaling
c   =+1. includes 0 and "growth" factor over max from IT-1
c   =-1. scaling will stretch but not shrink !=|1.|
c
c Prepare VLIM's to find vlcty extremes for prtcl plots NEXT DT.
if (mod(itp1,ipplt).ne.0) then
  if (autovlm.eq.1.) then
    vr/q/zmn=0.                      vr/q/zmx=0.
    else
      vr/q/zmn=vr/q/zlim(1,n)      vr/q/zmx=vr/q/zlim(2,n)
    endif
  endif
```

Appendix 8. Particle Buffering Routines: SETB, GETB, RESETB

The particle buffering routines are set up to do a reasonable job of managing the particles without the user having to know very much about the process. Since memories are big these days, those of you with short memories may not know that not too long ago it was important to be able to buffer particles to disk files. This package allows for that possibility but, generally, all particles are stored in memory.

The plan is that once the code knows how many species you want, the maximum number of particles of each species, how many quantities per particle, and the desired maximum number of particles per buffer, the user calls SETB. SETB determines the number of buffers for each species NBSPEC among other things. Subsequent calls to GETB then write buffers to storage as needed and get new buffers to be processed as consistent with the information in NBSPEC. Examples abound in this manual.

These particle quantities are stored serially

($R_1, Z_1, Vr_1, Vq_1, Vz_1, R_2, Z_2, Vr_2, Vq_2, Vz_2, R_3, Z_3, Vr_3, \dots$)

in the 2D array PBUF. [Again, throughout this write up and GYMNOS, Q or q is used to denote theta.] This storage scheme allows the particles to be managed with the loop index that skips through the loop index as follows:

```
do 450 lp=1,npthisb*dimp,dimp
      i=dxi*(r(lp)-rmin)+1.
      vx(lp)=vx(lp)+dt*ex(i) etc
      r(lp)=r(lp)+dt*vx(lp)
450   z(lp)=z(lp)+dt*vz(lp)
```

More of the comments found in these buffering routines are given below.

```
c=====
c=====GETB=====
c
c subroutine getb(ksp,npthisb,iparmem,ipbuf,nbspec)
c where...NBSPEC contains # of buffers for each prtcl type.
c       NSP the # of different prtcl types.
c       NDIMP the # of words required for each prtcl.
c       MPB the max # of prtcls per buffer.
c       NWB=NHEADB+NDIMP*MPB the total words in a buffer
c                           containing up to MPB prtcls.
c       NHEADB The number of header words/buffer.
c       KSP is the species # of the prtcl buffer.
c       IBCNT cumulative buffer count. ALWAYS 1ST HEADER WORD!
c       NPTHISB # of prtcls now in buffer. ALWAYS H. WORD 2!
c uses....IPBUF a buffer of NHEAD header words followed by
c                   MPB*NDIMP particle quantities.
c       IPARMEM(MEMOLY) core memory if MEMOLY>1 or disk file
c                           units 1 and 2 if MEMOLY<=1.
```

c For storage on disk, just set MEMOLY=1. For memory (MEMOLY>1),
c you must choose MLMOLY>=sumNSP NBSPEC(n)*(NHEADB+MPBUF*NDIMP)
c where NBSPEC(n)=1+((MSPEC(n)-1)/MPBUF). NBSPEC is computed for
c you in SETB where MPBUF may be reduced MPB for disk.

c

c GENERAL: This two routines do particle buffering. SETB sets up
c buffering & GETB cycles through the particles as in triple prtl
c memory buffering. This procedure is similar to Nielson's triple
c prtl disk buffering circa 1973. This buffering package is
c generalized to allow partially filled buffers with "header" words
c on each to give information about the buffer and its contents.

c

c USAGE: SETB is called at the beginning of the run to initialize
c parameters used to buffer the prtl's. Subsequent calls to GETB
c write into memory or disk files the present buffer--consisting
c of NHEADB buffer-describing header words and NDIMP floating point
c numbers that characterize each of the NPTHISB particles in this
c buffer. Generally GETB writes the present buffer and reads the
c next. NPTHISB must be consistent with the present buffer; it will
c be reset for the new buffer automatically. KSP contains the new
c species # on return. If all the buffers of the present species
c {the # of buffers for each species is contained in NBSPEC(KSP)}
c have been processed, KSP is incremented and the first buffer of
c the next species is read into the local memory IPARMEM. If all
c the buffers for the last species have been processed, KSP=-1 and
c control is returned to the calling routine.

c

c DETAILS: Call SETB(NSP,MPB,MSPEC,NSPEC,NBSPEC,MEMOLY,DIMP,NHEADB)
c initializes the routines. MPB starts as MPBUF (maximum dimensioned
c number of particles in PBUF) and is used in the calling routine.
c If MEMOLY>1 buffering MPB will be reduced to the largest # of
c prtl's + NHEADB that will fit into 2 disk sectors.
c Call RESTB(KSP,NPTHISB,iparmem,ipbuf,nbspec) is used before a
c bufr cycle goes through all bufrs for each species to arrest the
c buffering cycle. For MEMORY BUFFERING (MEMOLY>1), this resets
c parameters so that next call GETB ignores and overwrites present
c contents of IPBUF and reads in buffer 1 of species 1. For DISK
c BUFFERING (MEMOLY=1), call causes everything to be reset as it
c was before this pass started. All particle updates written on
c this aborted pass ignored. (A disk write "pushes" an EOF in front
c of it so if you rewind halfway out, the EOF is left halfway out.)
c Once you a disk file, all information farther out is lost.)
c It is possible to cycle through all particles without writing
c (NPTHISB<0 on input) as is useful in initializing the t=0 prtl's.

```

c Another option is to cycle through particles without reading
c (KSP=-2 on input) for such purposes such as diagnostic passes.
c           SUMMARY..Never write if NPTHISB<0. Never read if KSP=-2.
c Can't write on 1st bufr 1st spcs or read on last bufr last spcs.
c

c     EXAMPLE
c     parameter (nheadb=10,mpbuf=1000,dimp=5,memoly=80160,nsp=2)
c     common /PLCM/ iparmem(memoly),..
c     .   ...nbspec(nsp),mspec(nsp),nspec(nsp),mpb
c     common /PARBUF/ ipbuf(nheadb),pbuf(dimp,mpbuf)
c     dimension r(1),z(1),..
c     equivalence (r(1),pbuf(1,1)), (z(1),pbuf(2,1)),....
c...
c           1st ---> [Reserve Buffers]--> mpb=mpbuf $ call setb(nsp,
c                                         mpb,mspec,nspec,nbspec,
c
c           [Initialize (write only)] <- memoly,dimp,nheadb)
c           ksp$av=0 $ 200 ksp=-2
c           call getb(ksp,npthisb,iparmem,ipbuf,nbspec)
c           if (ksp.lt.0) go to 300 $ if (ksp.ne.ksp$av) then $ lpspc=0
c           npthisb=1+(npinit(ksp)-1)/nbspec(ksp) $ ksp$av=ksp $ endif
c           npthisb=min0(npinit(ksp)-lpspc,npthisb) $ lpspc=lpspc+npthisb
c           ipbuf(5) =8hELECTRNS      etc
c           do 280 lp=1,npthisb*dimp,dimp      [Cycle (read & w) particles]
c 280   r(lp)=          etc      300 it=it+1
c           go to 200                  400 call getb(n,npthisb,iparmem,
c                                         .ipbuf,nbspec)$if(n.lt.0)goto900
c           [Diag (read only) pass]      do 450 lp=1,npthisb*dimp,dimp
c           900 ksp=0                  450 r(lp)=r(lp)+dt*vx(lp)  etc
c           910 call getb(n,npthisb,iparmem,          go to 400
c             .ipbuf,nbspec) $ if (n.lt.0) go to 980
c             if (n.ne.ksp) then $ ksp=n $ spname=ipbuf(5)
c               call cplot(..,spname,8,'r',1,...) $ endif
c               call pplot(1,1,npthisb,r(1),z(1),dimp) $ npthisb=-1 $ goto 910
c 980   go to 400      Can immediately call GETB to start again
c-----hewett 4/3/89, 88, 77
c
```

```

c=====SETB=====
c
c subroutine setb(nnsp,nmpb,mspec,nspec,nbspec,
c   .nmemory,idimp,nnheadb)
c
c The call to SETB sets up i/o buffering of the particles.
c where...MSPEC contains the MAX # of each prtcl type.
c NSPEC contains PRESENT # of each prtcl type.
c Other parameters are documented in GETB.

```

```

c
c   Call to SETB initializes the buffering routines.  MPB starts
c   as MPBUF (maximum dimensioned number of particles in PBUF) and
c   is used in the calling routine.  If MEMOLY>1, MPB will be
c   reduced to the largest number of particles+NHEADB that will fit
c   into 2 disk sectors.  (see GETB for documentation)
c-----hewett 4/3/89, 88, 77
c   Compute MPB, max # prtcls that fit in 2 disk sector records.

c   Check memory and determine NBSPEC

c=====INCB=====
c
c   subroutine incb(n,nbinc,nbspec,ipbuf,iparmem)
c   Call to INCB enlarges # of bufrs allowed for species N by NBINC.
c   Other parameters are documented in GETB.
c   where...NBSPEC contains # of buffers for each prtcl type.
c           NSP the # of different prtcl types.
c           NWB the total words in a buffer
c           IBCNT cumulative buffer count.  ALWAYS 1st HEADER WORD!
c           NPTHISB # partcls this buffer.  ALWAYS 2nd HEADER WORD!
c   uses....IPBUF a buffer of MWB words.
c   Memory if MEMOLY>1 or disk files units 1 & 2 if MEMOLY=1.
c-----hewett 2/90

c          DISK
c
c   March through disk file from front, putting extra dummy writes
c   writes in file after existing NBSPEC(N) buffers of specie. N.
c   Can only do this after a full sweep through all buffers so that
c   IT1 & IT2 have just been switched.

c          CORE
c
c   March back towards front of particle core storage, copying the
c   trailing buffers NBINC*MWB farther out.  Process stops at species
c   N+1 so that room has been made for the desired extra NBINC buffers
c   for species N.  Could do this within a full sweep anytime before
c   species N is started--like after a RESETB.

```

Appendix 9. VDKR

This is the routine that started the whole GYMNO\$ project; see the HISTORY section in the introduction. The salient feature is that this routine treats all point, boundary or interior equally. Any boundary condition can be applied at any point.

And it all vectorizes.

```
=====
      subroutine vdkr(ndim,s,rhs,tp,tc,tm,t0,fsing,
      x dr,nr1,rg,dz,nz1,itt,it,snglstp,srstor,
      x err,ier,etest,key,kn,bv,bvr,bvz,omglistp,iperodz,jvc)
c
c      This subroutine solves either
c          d/dr(1/r d(rS)/dr) + d/dz(dS/dz) = RHS   (jvc=1)
c      or      d/dr(r dS/dr)      + r d/dz(dS/dz) = r RHS   (jvc=0)
c              using Dynamic ADI. D.W. Hewett 90,88,87,84,79,77
c
c      S & RHS are stored on cell corners in the R-Z plane. Here
c the mesh is laid out as RGrid(i)=(i-1)*DR+RMIN, RG(NR1)=RMAX
c with a corresponding definition of ZG. BC's are applied at at
c all pts. with KEY(i,j)=-1. If KN(i,j)=0, the pt. is a Dirichlet
c bndry pt. If KN=+/-1{10}, pt. is a Neumann for(back)ward +(-)
c derivative in R{Z}. [The Neumann condition is "curl-type"
c (includes cylindrical R factors as in (rS)' ) if KN=+/-2. This
c type of BC is not allowed at RMIN if zero.] Boundary values
c carried by BV(i,j). All pts. that are nonDirichlet, KN!=0, need
c RHS values set since all Neumann conditions make use of the same
c basic eqn as interior pts. for 2nd order accuracy (except at RMJN
c if it is 0). Routine has been configured to do both scalar
c Laplacian (JVC=0) or GRAD(DIV)-CURL CURL (JVC=1).
c
c      This routine contains an entry GRDCRLS with same arguments that
c will return in locations TP, TC the r and z components of either
c GRAD(S) or CURL(S), JVC=0,1 respectively, at all cell corners.
c This entry eases the difficulty of reconstructing the finite
c difference equations necessary to evaluate the derivatives on
c the cell corners on the edge of the mesh. (DEC. 88)
c
c      This routine expects to use the ranges 1.le.I(J).le.NR1(NZ1).
c The work arrays must be dimensioned accordingly. For example,
c arrays dimensioned (-1:NR,-1:NZ) so that the first "real" location
c is (0,0) in calling routine may use this routine by using (0,0) in
c the argument list so that in this routine we have a (1:NR1,1:NZ1)
c mesh here. [NDIM must be NR2 in this case.] D.W. Hewett, Jan/88
c-----
c      This version of VDADI uses "KEY"s to construct int'l bndrys.
```

```

c      KEY ==+1 if in the plasma
c          = 0 if in a vacuum region
c          ==-1 if ON or IN intrl boundry
c      KN = 0, BC pt is Dirichlet
c          ==+1(-1) on boundary w/ for(back)ward R-derivative
c          ==+2(-2) curl-type Neumann
c          ==+8(-8) on bdry w/ +(-)Z & -(+)R(curl)-derivative
c          ==+9(-9) on bdry w/ +(-)Z & -(+)R-derivative
c          ==+10(-10) on intrl boundary w/ for(back)ward Z-drvtv
c          ==+11(-11) on bdry w/ +(-)Z & +(-)R-derivative
c          ==+12(-12) on bdry w/ +(-)Z & +(-)R(curl)-derivative
c      BV = the value specified on and in the intrl boundary.
c      CAUTION: Intrl boundaries are built only at cell edges!
c                  Double Neumann pt both use same BVR and BVZ.
c -----
c      KEY(i,j)=  1=plasma,           0=vacuum,      -1=BC pt.
c      kd==key(i,j)*(1-key(i,j))/2   gives 1 on a BC pt. else 0
c      omkd=1-kd                   gives 1 for nonBC pt.
c      ky=iabs(key(i,j))          gives full eq. in plasma & on BC
c      omky=1-ky                   gives 1 in vacuum
c      KN(i,j)=10=forward Z-Numan,    2=forward curl R-Numan,
c          1=forward R-Neumann, 0=Dirichlet, -1=backward R-Neuman,
c          -2=backward curl R-Neumann, -10=backward Z-Neuman
c          12=forw R(curl)-N forw Z-N,   11=forw R-N forw Z-N,
c          9=back R-N forw Z-N,        8=back R(curl)-N forw Z-N,
c          -8=forw R(curl)-N back Z-N, -9=forw R-N back Z-N,
c          -11=back R-N back Z-N,     -12=back R(curl)-N back Z-N
c          knz=kn(i,j)/8            gives +/-1      for z-Neumann else 0
c          knr=kn(i,j)-knz*10       gives +/-2 or 1 for r-Neumann else 0
c          ktyp=knr/2              gives +/-1      to make r-Neumann KTyp
c          knr=knr-ktyp            gives +/-1      for r-Neumann else 0
c          ktyp=iabs(ktyp)         gives 0/1      for curl-type R Neumann
c -----
c      General BC set up
c      Basic eqn.  crp*s(i+1,j)+crm*s(i-1,j)+czp*s(i,j+1)+czm*s(i,j-1)
c                  +(crc+czc-omg)*s(i,j)=rhs-omg*s(i,j)
c      R pass coding
c      Interior                      KD=KNR=KNZ=0
c      tp=crp             cp=czp
c      tc=crc-omg        cc=czc+omg   t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=crm             cm=czm
c      BC      Dirichlet point          KD=1 KNR=KNZ=0
c      tp=0                cp=0
c      tc=1                cc=0          t0=bv

```

```

c      tm=0          cm=0
c BC      Neumann point (+r into region)  KD=1 KNR=1 KNZ=0
c      tp=crp+crm*rp1/rm1 cp=czp           (extra R's only for curl-type)
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=0              cm=czm           +bvr*2*dr*crm*r/rm1
c BC      Neumann point (-r into region)  KD=1 KNR=-1 KNZ=0
c      tp=0              cp=czp           (extra R's only for curl-type)
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=crm+crp*rm1/rp1 cm=czm           -bvr*2*dr*crp*r/rp1
c BC      Neumann point at RMIN=0        KD=1 KNR=1 KNZ=0
c      tp=2*dris         cp=dzis          (no curl-type allowed)
c      tc=-2*(dris+dzis) cc=0             t0=rhs  (BV must be zero)
c      tm=0              cm=dzis          (JSC=1 only)
c BC      Neumann point (+z into region)  KD=1 KNR=0 KNZ=1
c      tp=crp            cp=czp+crm
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=crm            cm=0             +bvz*2*dz*czm
c BC      Neumann point (-z into region)  KD=1 KNR=0 KNZ=-1
c      tp=crp            cp=0
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=crm            cm=czm+czp       -bvz*2*dz*czp
c -----
c BC  Double Neumann point (+r+z into regi n) KD=1 KNR=1 KNZ=1
c      tp=crp+crm*rp1/rm1 cp=czp+crm     (extra R's only for curl-type)
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=0              cm=0             +2*(bvr*dr*crm*r/rm1+bvz*dz*czm)
c BC  Double Neumann point (+r-z into region) KD=1 KNR=+1 KNZ=-1
c      tp=crp+crm*rp1/rm1 cp=0           (extra R's only for curl-type)
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=0              cm=czm+czp       +2*(bvr*dr*crm*r/rm1-bvz*dz*czp)
c BC  Double Neumann point (-r+z into region) KD=1 KNR=-1 KNZ=1
c      tp=0              cp=czp+crm     (extra R's only for curl-type)
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=crm+crp*rm1/rp1 cm=0           +2*(-bvr*dr*crp*r/rp1+bvz*dz*czm)
c BC  Double Neumann point (-r-z into region) KD=1 KNR=-1 KNZ=-1
c      tp=0              cp=0           (extra R's only for curl-type)
c      tc=crc            cc=czc           t0=rhs-cp*s(j+1)-cc*s(j)-cm*s(j-1)
c      tm=crm+crp*rm1/rp1 cm=czm+czp       +2*(-bvr*dr*crp*r/rp1-bvz*dz*czp)
c -----
c Ideas;
c   If a point is double Neumann (or periodic?),
c   DIAGONAL STACK 10/24/89
c   To find Div(C),
c   call grdcrlS(ndim,Cr,rhsCr,tp,Dz,tm,t0,fsing,...,jvc=1)

```

```

c      call grdcrlS(ndim,Cz,rhsCz,Dr,tc,tm,t0,fsing,...,jvc=1)
c      Div(C)=Dz-Dr
c      To find Curl(Ar,Az)|q
c      call grdcrlS(ndim,Ar,Jr,tp,DzAr,tm,t0,fsing,...,jvc=1)
c      call grdcrlS(ndim,Az,Jz,DrAz,tc,tm,t0,fsing,...,jvc=0)
c      Bq=Curl(Ar,Az)|q=DzAr-DrAz
c -----

```

The coding that actually accomplishes the above set up follows. The z pass section is identical except that the definitions for TP(I,J) and CP, TC(I,J) and CC, and TN(I,J) and CM are interchanged *and* the sign of the ADI acceleration factor is changed. In addition Neumann boundary conditons for the scalar Poisson equation require an application of l'Hospital's rule to avoid dividing by zero in the finite difference algorithm if RMIN=0. The details are given in the FORTRAN below. It's "straightforward"!

```

c      set up r direction pass on s.          ***s-r**
jm1=j1pnzm1
do 150 j=1,nz1
  jp1=j+1- ( j /nz1)*j1pnzm1
  do 140 i=1,nr1
    rm1=rg(i)-dr   rp1=rg(i)+dr   rp=rg(i)+drh   rpi=1./rp
    rm=rg(i)-drh   rmi=1. / (rg(i)-drh+dr*1.e-12)   rr=rg(i)
c      Plasma, vacuum, BC pt detection and control flags.
  kd=-key(i,j)*(1-key(i,j))/2  omkd=i-kd  knz=kn(i,j)/8
  knr=kn(i,j)-knz*10 ktyp=knr/2 knr=knr-ktyp ktyp=iabs(ktyp)
c      Basic equation FD set up.
c      d/dr(1/r d(rS)/dr) + d/dz(dS/dz) = RHS
c      d/dr(r dS/dr) + r d/dz(dS/dz) = r RHS
  crp=( rp1*rpi*dris )*jvc +( rp*dris )*jsc
  crc=(-rr*(rpi+rmi)*dris )*jvc +(- (rp+rm)*dris )*jsc
  crm=( rm1*rmi*dris )*jvc +( rm*dris )*jsc
  czp=( dzis )*jvc +( rr*dzis )*jsc
  czc=(-(dzis+dzis) )*jvc +(- (rr+rr)*dzis )*jsc
  czm=( dzis )*jvc +( rr*dzis )*jsc
c      Basic equation FD done--now fix up BCs
  rbmvp=1-ktyp+ktyp*rm1/rp1      rbvp =1-ktyp+ktyp*rr /rp1
  rbpvm=1-ktyp+ktyp*rp1/(rm1+1.e-12*dr)
  rbvm =1-ktyp+ktyp*rr /(rm1+1.e-12*dr)
  akr=iabs(knr)      akz=iabs(knz)
  yesn=akr+akz      iyesdn=.50001*yesn    yesn=yesn-iyesdr
  akr=akr-iyesdn    akz=akz-iyesdn
  opknr=knr*(1+knr)      opknz=knz*(1+knz)
  omknz=knz*(1-knz)      omknz=knz*(1-knz)
  tp(i,j)=(omkd+akz)*crp   + .5*opknr*(crp+rbpvm*crm)

```

```

tc(i,j)=(omkd+yesn)*(crc-omg) +kd*(1.-yesn)
tm(i,j)=(omkd+akz)*crm -.5*omknr*(crm+rbmvp*crp)
cp =(omkd+akr)*czp +.5*opknz*(czp+czm)
cc =(omkd+yesn)*(czc+omg)
cm =(omkd+akr)*czm -.5*omknz*(czm+czp)
c      diagonal stacking on double Neumann pts.    10/26/89
tc(i,j)=tc(i,j)+iyesdn*(cc-omg)
cc=cc-iyesdn*(cc-omg)
140 t0(i,j)=(omkd+yesn)*(jsc*rr+jvc)*rhs(i,j)
x           -cp*s(i,jp1)-cc*s(i,j)-cm*s(i,jm1)
x   +kd*(bv(i,j)*(1.-yesn)
x   +bvz(i,j)*dz*(opknz*      czm+omknz*      czp)
x   +bvr(i,j)*dr*(opknr*rbvm*crm+omknr*rbvp*crp))
if (jsc.eq.1 .and. rg(1).eq.0.) then
  knz=kn(1,j)/8      knr=kn(1,j)-knz*10
  if (knr.eq.+1) then
c      From xpandng Dr^2, l'Hospital, & KNR=+1 (!=2 !!)  1/10/90
    tp(1,j)=4*dris  tc(1,j)=-4*dris-dzis-dzis  tm(1,j)=0.
    t0(1,j)=rhs(1,j)+4*dri*bvr(1,j)
    .     -iabs(knz)*(dzis+dzis)*(s(1,j+knz)-knz*dz*bvz(1,j))
    .     -(1-iabs(knz))*dzis*(s(1,jp1)+s(1,jm1))
    endif
  endif
  fsing(j)=0.
  if (is.gt.1) go to 150
    do 145 i=1,nr1
      im1=i-1+((nr2-i)/nr1)*i1pnrm1      ip1=i+1-(i/nr1)*i1pnrm1
      tep=t0(i,j)
      .     -tp(i,j)*s(ip1,j)-tc(i,j)*s(i,j)-tm(i,j)*s(im1,j)
      ier=cvmgmm(j+i*1000,ier,abs(err)-abs(tep))
145   err=cvmgmm(tep,err,abs(err)-abs(tep))
150 jm1=j
  if (is.eq.1) then
    err=err*vnorm
    if (abs(err).lt.etest.and.it.gt.1 .or. it.eq.itt) goto 750
    endif
  call vtrikr(fsing,ndim,nr1,nz1,1,1,tp,tc,tm,t0,s,dris,0)
  ....
c      Now compute -GRAD(S) (jvc=0) or CURL(S) (jvc=1)
c      entry grdcrlS(ndim,s,rhs,tp,tc,tm,t0,fsing,
x dr,nr1,rg,dz,nnz1,itt,it,snglstp,srstor,
x err,ier,etest,key,kn,bv,bvr,bvz,omg1stp,iperodz,jvc)

```

```

c      Vector returned; R-comp in TP...Z-comp in TC

c      1st, set Values around 4 outside walls (do 850 i & 825 j)

c          Basic equation FD set up.
c      d/dr(1/r d(rS)/dr) + d/dz(dS/dz) = RHS
c          d/dr(r dS/dr) + r d/dz(dS/dz) = r RHS
c          If this mesh edge pt is 1 of the 4 corners, fix it.

c          DIRICHLET CORNER POINT SECTION
c          If corner Dirichlet, assume both bdys are Dirichlet
c          & do 1-sided differences as if bdys were linear. 10/24/89

c          Check assumption--If 1 of the Bdys meeting in corner not
c          Drchlt, project Drchlt linearly & compute Sout in other.

c          R bdy Drchlt so treat z-drvtive as given (as just xtrpolted)
c          & compute Sout[=s(0orNR2,1orNZ1)] for r-drvtive as below.

c          Z bdy Drchlt so treat r-drvtive as given (as just xtrpolted)
c          & compute Sout[=s(1orNR1,0orNZ2)] for z-drvtive as below.

c          Must have been Dirichlet on both boundaries--pt finished.

c          NEUMANN CORNER POINT SECTION

c          KNR=1,KNZ=0      Z-derivative known (TP)... need R-der. (TC)

c          KNR=0,KNZ=1        R-drvtv known.. need Z-drvtv.

c          Finish interior -grad(s) or curl(s)

```

Appendix 10. VTRIK

This routine is a fully vectorized tridiagonal solver. Consider a system of NR1 by NZ1 points. For say the implicit R pass, we have NZ1 lines in the R direction of the overall matrix that need to have a tridiagonal matrix solved. Even though the tridiagonal algorithm is inherently recursive on each line, we can still vectorize by doing all the NZ1 lines at once. VDKR accomplishes this with separate entries for R and Z passes and, within each entry, is a switch for producing solutions with periodic boundary conditions.

```
c=====
c====subroutine vtridg(fsng,ndim,nr1,nz1,ijstar,ijskp,
c      cp,cc,cm,q,V,dis,iperod)
c-----hewett, 88,84,81--dwh,cwn, 1977    added DIS 11/89
c
c      RMAX=RMIN+NR*DR
c      R(i)=(i-1)*DR+RMIN    for i=1,NR1      (NR=NR1-1)
c      Returns V(i) including V(1) and V(NR1) values.
c      (Could return V in Q: use Q for both Q & V in FORTRAN call.)
c      A row (or column) solution assumed singular if FSNG(J)=1.
c      CP, CC, and CM contents are destroyed.
c
c      real cp(ndim,2),cc(ndim,2),cm(ndim,2),q(ndim,2),
c      .  V(ndim,2),fsng(2)
c
```

The entry VTRIKR provides the R pass solution.

```
entry vtrikr(fsng,ndim,nr1,nz1,jstar,jskp,
c      cp,cc,cm,q,V,dis,iperod)
```

This entry solves tridiagonal matrices equations

$$cp(i,j) * V(i + 1.j) + cc(i,j) * V(i,j) + cm(i,j) * V(i - 1.j) = q(i,j)$$

for J=JSTAR,NZ1,JSKP where JSTAR is an arbitrary first line and JSKP is a skip parameter. (This same routine is often used for red-black line SOR and the starting offset and skipping by 2 makes this convenient.) IPEROD=1 declares the system to be periodic; and FSNG(J)=1. declares the system to be singular (e.g. when the solution is arbitrary to within a constant).

The entry for Z passes is

```
entry vtrikz(fsng,ndim,nr1,nz1,istar,iskp,
c      cp,cc,cm,q,V,dis,iperod)
```

and this causes the system of columns

$$cp(i,j) * V(i,j + 1) + cc(i,j) * V(i,j) + cm(i,j) * V(i,j - 1) = q(i,j)$$

to be solved for simultaneously for I=ISTAR,NR1,ISKP. Controls analogous to the R pass are applied via FSNG(I) and IPEROD.

Appendix 11. VPARMOV/VTRAP/VBIRTH/DEATH

```

c=====VPARMOV===
subroutine vparmov
    ...
        (new equivalence used here!)
dimension vsad(11), sorvec(nr2,nz2,nsp,1)
equivalence (den(1,1,1),sorvec(1,1,1,1))
common /parvec/ i(mpbuf),j(mpbuf), fr(mpbuf), fz(mpbuf),
. xtil(mpbuf), ztil(mpbuf), vxh(mpbuf), vyh(mpbuf), vzh(mpbuf),
. omgr(mpbuf),omgq(mpbuf),omgz(mpbuf),
. w1(mpbuf),w2(mpbuf),w3(mpbuf),w4(mpbuf),skout(mpbuf),
. vxnp1(mpbuf),vynp1(mpbuf),vznp1(mpbuf),
. eacclr(mpbuf),eacc1q(mpbuf),eacc1z(mpbuf)
.
c-----hewett, 1991,1986
c      rotate=2.(0.) gives(stops) particle rotation
rotate=2.
c
c      Assume we start here with   r(n),vx(n-1/2)
c      The fields E and B are at the (n) time level.
c
do 20 l=1,npthisb                      lp=(l-1)*dimp+1
    rr=dri*(r(lp)-rmin)+1.              rz=dzi*(z(lp)-zmin)+1.
    i(l)=rr                  fr(l)=rr-i(l)          frc=1.-fr(l)
    j(l)=rz                  fz(l)=rz-j(l)          fz=1.-fz(l)
    w1(l)=frc *fzc      w2(l)=frc *fz(l)      w3(l)=fr(l)*fzc
20   w4(l)=fr(l)*fz(l)
do 30 l=1,npthisb
    eacclr/q/z(l)=
        . w1(l)*er/q/z(i(l),j(l)) +w2(l)*er/q/z(i(l),j(l)+1)
        . +w3(l)*er/q/z(i(l)+1,j(l)) +w4(l)*er/q/z(i(l)+1,j(l)+1)
30   omgr/q/z(l)=
        . w1(l)*br/q/z(i(l),j(l)) +w2(l)*br/q/z(i(l),j(l)+1)
        . +w3(l)*br/q/z(i(l)+1,j(l)) +w4(l)*br/q/z(i(l)+1,j(l)+1)
do 40 l=1,npthisb                      lp=(l-1)*dimp+1
40   vx/y/zh(l)=vx/y/z(lp)+eacclr/q/z(l)
do 45 l=1,npthisb                      lp=(l-1)*dimp+1
    did .5 of E push           rotation follows
    vx/y/zt=vx/y/zh(l)
    . +vy/z/xh(l)*omgz/r/q(l)-vz/x/yh(l)*omgq/z/r(l)
    fact=rotate/
    . (1.+omgr(l)*omgr(l)+omgq(l)*omgq(l)+omgz(l)*omgz(l))
c      vx,y,znp1 used as temporary storage
45   vx/y/znp1(l)=vx/y/z(lp)

```

```

        +fact*(vy/z/xt*omgz/r/q(l)-vz/x/yt*omgq/z/r(l))
do 48 l=1,npthisb
      rotation finished      do remaining half of E push
      vx/y/zh(l)=vx/y/znp1(l)+eacclr/q/z(l)
48   x/ztil(l)=r/z(lp)+vx/zh(l)*dt
      do 50 l=1,npthisb
      lp=(l-1)*dimp+1
      now advance to r(n+1) in order to find vx(n+1/2) 4/3/90
      z(lp)=ztil(l)
50   r(lp)=sqrt(xtil(l)*xtil(l)+vyh(l)*dt*vyh(l)*dt)
      do 52 l=1,npthisb
      lp=(l-1)*dimp+1
      costh=xtil(l)/r(lp)           sinth=vyh(l)*dt/r(lp)
      vz(lp)=vzh(l)
      vx(lp)= costh*vxh(l)+sinth*vyh(l)
52   vy(lp)=-sinth*vxh(l)+costh*vyh(l)
      PUSH COMPLETED, now find new indices & lost particles
      do 60 l=1,npthisb          lp=(l-1)*dimp+1
      rr=dri*(r(lp)-rmin)+1.      rz=dzi*(z(lp)-zmin)+1.
      i(l)=rr          fr(l)=rr-i(l)          frc=1.-fr(l)
      j(l)=rz          fz(l)=rz-j(l)          fzr=1.-fz(l)
      w1(l)=frc *fzc      w2(l)=frc *fz(l)
      w3(l)=fr(l)*fzc      w4(l)=fr(l)*fz(l)
      icen=rr+1.          jcen=rz+1.
      icen=min0(max0(icen,1),nr2)    jcen=min0(max0(jcen,1),nz2)
60   skout(l)=skey(icen,jcen)

c
c           TRAP LOST prtcls, REFLCT, THREMT, NJCT and FLDEMT
call vtrap
c           ACCUMULATE PLASMA SOURCES
c
do 70 l=1,npthisb
      eacclr/q/z(l)=
      . w1(l)*er/q/z(i(l),j(l)) +w2(l)*er/q/z(i(l),j(l)+1)
      . +w3(l)*er/q/z(i(l)+1,j(l)) +w4(l)*er/q/z(i(l)+1,j(l)+1)
70   omgr/q/z(l)=
      . w1(l)*br/q/z(i(l),j(l)) +w2(l)*br/q/z(i(l),j(l)+1)
      . +w3(l)*br/q/z(i(l)+1,j(l)) +w4(l)*br/q/z(i(l)+1,j(l)+1)
      do 80 l=1,npthisb          lp=(l-1)*dimp+1
cdon't need to save arrays for VXH,VYH
      vx/y/zh(l)=vx/y/z(lp)+eacclr/q/z(l)
      did .5 of En(r(n+1)) push--rotation follows
      vx/y/zt=vx/y/zh(l)
      . +vy/z/xh(l)*omgz/r/q(l)-vz/x/yh(l)*omgq/z/r(l)

```

```

fact=rotate/
(1.+omgr(l)*omgr(l)+omgq(l)*omgq(l)+omgz(l)*omgz(l))
vx/y/zh(l)=vx/y/zh(l)
+fact*(vy/z/xt*omgz/r/q(l)-vz/x/yt*omgq/z/r(l))
c      rotation done using Bn(r(n+1))--do last .5 E push
80    vx/y/znp1(l)=vx/y/zh(l)+eacclr/q/z(l)
c      den, rhs,ur,uq,uz
c      MVEC      1,   2,  3, 4, 5
      mvtop=5
      do 200 l=1,npthisb
      vsad(1)=1.
      vsad(2)=vxnp1(l)*vxnp1(l)+vynp1(l)*vynp1(l)+vznp1(l)*vznp1(l)
      vsad(3)=vxnp1(l)
      vsad(4)=vynp1(l)
      vsad(5)=vznp1(l)
      do 100 mvec=1,mvtop
100    sorvec(i(l) ,j(l) ,n,mvec)=
      sorvec(i(l) ,j(l) ,n,mvec)+w1(l)*vsad(mvec)
      do 120 mvec=1,mvtop
120    sorvec(i(l)+1,j(l) ,n,mvec)=
      sorvec(i(l)+1,j(l) ,n,mvec)+w3(l)*vsad(mvec)
      do 140 mvec=1,mvtop
140    sorvec(i(l) ,j(l)+1,n,mvec)=
      sorvec(i(l) ,j(l)+1,n,mvec)+w2(l)*vsad(mvec)
      do 160 mvec=1,mvtop
160    sorvec(i(l)+1,j(l)+1,n,mvec)=
      sorvec(i(l)+1,j(l)+1,n,mvec)+w4(l)*vsad(mvec)
200    continue
      return
      end

```

```

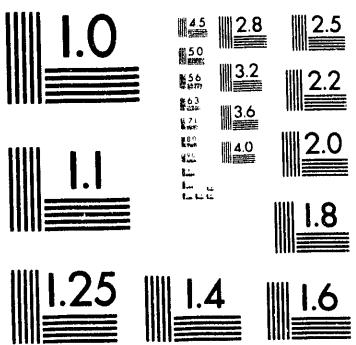
c=====
=====VTRAP=====
      subroutine vtrap
c                               Detect lost prtcls.
      noutsde=0          npart=0
      l=0
120 l=l+1          multirp=0
130 if (l.gt.npthisb) go to 200
      if (skout(l).gt.-1.) go to 120
      lp=(l-1)*dimp+1
      lplast=(npthisb-1)*dimp+1
c      Have found one!
      pcflr=abs(dri*vx(lp)*dt)      pcflz=abs(dzi*vz(lp)*dt)
      ib=1.+amax1(pcflr,pcflz)

```

```

if (ib.ge.mxgid .or. multitrp.gt.mxgid) then
c      2 fast or # of tries MULTITRP 2 big....delete
      npart=npart+1          noutsde=noutsde+1
      call death(lp,lplast,l,npthisb)
      npthisb=npthisb-1      lplast=lplast-dimp
      go to 130
      endif
      Find close corner at indices ICLS,JCLS.
c Cycles back with smaller CFLok DTCLS to find where last prtcl
c crossing occurred. Uses multiple passes to catch those whose
c reflection puts them in yet another structure.      DWH 8/31/90
      dtcls=1.000001*dt
      rrun=r(lp)      zrun=z(lp)
      if (ib.gt.1) dtcls=dt/ib
      dthvx=.5*dtcls*vx(lp)      dthvz=.5*dtcls*vz(lp)
73   rrun=rrun-dthvx-dthvx      zrun=zrun-dthvz-dthvz
      if (iperodz.eq.1) zrun=zmin+amod(zrange+zrun-zmin,zrange)
      ib=ib-1
      if (ib.lt.-2) then
c          ib<-2....delete
          npart=npart+1          noutsde=noutsde+1
          call death(lp,lplast,l,npthisb)
          npthisb=npthisb-1      lplast=lplast-dimp
          go to 130
          endif
          icen=dri*(rrun-rmin)+2.      jcen=dzi*(zrun-zmin)+2.
          icls=dri*((rrun+dthvx)-rmin)+1.5
          jcls=dzi*((zrun+dthvz)-zmin)+1.5
          if (icls.lt.1 .or. icls.gt.nr1 .or. jcls.lt.1
              .or. jcls.gt.nz1 .or. skey(icen,jcen).eq.-1.) go to 73
c
c          Is this prtcl PERFECT REFLECTED?
c          Now ICLS,JCLS are the indices of the closest corner.
c          if (iabs(ipbc(icls,jcls)).ne.51) then
c              DELETE the prtcl.
c              if (ipbc(icls,jcls).le.50
c                  .or. ipbc(icls,jcls).eq.61) ireemt=ireemt+1
c              qabsrb(icls,jcls)=qabsrb(icls,jcls)+sq(n)
c              npart=npart+1
c              call death(lp,lplast,l,npthisb)
c              npthisb=npthisb-1      lplast=lplast-dimp
c              go to 130
c              endif
c          ok PERFECT REFLECTION-orient the surface with KORNT

```



20f2

```

...same as TRAP

c -----
c           Now Thermal REEMISSION from cell corners

...same as TRAP

call vbirth(npart,vmp,unorm,evnvth,evtvth,kornt(icor,jcor),
            edithr,edithz,sgnr,sgnz,rg(icor),zg(jcor))

...same as TRAP

c -----
c           now FIELD EMIT or NJCT NEW prtcls

...same as TRAP

call vbirth(npart,vmp,unorm,evnvth,evtvth,          0,
            edithr,edithz,sgnr,sgnz,remt,zemt)

...same as TRAP

c=====DEATH=====
subroutine death(lp,lplast,l,npthisb)

common /parvec/ i(mpbuf),j(mpbuf), fr(mpbuf), fz(mpbuf),
. vxh(mpbuf),vyh(mpbuf), omgr(mpbuf),omgq(mpbuf),omgz(mpbuf),
. w1(mpbuf),w2(mpbuf),w3(mpbuf),w4(mpbuf),skout(mpbuf),
. vxnp1(mpbuf),vynp1(mpbuf),vznpi(mpbuf),
. eacclr(mpbuf),eacclq(mpbuf),eacclz(mpbuf)

c
r(lp)=r(lplast)      z(lp)=z(lplast)
vx(lp)=vx(lplast)    vy(lp)=vy(lplast)      vz(lp)=vz(lplast)
i(l)=i(npthisb)       j(l)=j(npthisb)
fr(l)=fr(npthisb)     fz(l)=fz(npthisb)
skout(l)=skout(npthisb)
c
vxh(l)=vxh(npthisb)
c
vyh(l)=vyh(npthisb)
c
omgr(l)=omgr(npthisb)
c
omgq(l)=omgq(npthisb)
c
omgz(l)=omgz(npthisb)
w1(l)=w1(npthisb)   w2(l)=w2(npthisb)
w3(l)=w3(npthisb)   w4(l)=w4(npthisb)

```

```

c      eacclr(l)=eacclr(npthisb)
c      eacclq(l)=eacclq(npthisb)
c      eacclz(l)=eacclz(npthisb)
      return
      end
c=====VBIRTH=====
      subroutine vbirth(npart,vmp,unorm,evnvth,evtvth,krntflg,
                         edithr,edithz,cosa,sina,remt,zemt)
                         SAME AS BIRTH
      common /parvec/ i(mpbuf),j(mpbuf), fr(mpbuf), fz(mpbuf),
      . vxh(mpbuf),vyh(mpbuf), omgr(mpbuf),omgq(mpbuf),omgz(mpbuf),
      . w1(mpbuf),w2(mpbuf),w3(mpbuf),w4(mpbuf),skout(mpbuf),
      . vxnp1(mpbuf),vynp1(mpbuf),vznp1(mpbuf),
      . eacclr(mpbuf),eacclq(mpbuf),eacclz(mpbuf)
                         SAME AS BIRTH
c4 if these velocities too large, we still must check SKEY
c  If we're out, it must have been the VX or VZ of the last step
c
c               new vectorized stuff   12/16/90   4/29/91
      l=npthisb
    720 l=l+1
    730 if (l.gt.npthisb+npart) go to 800
      lp=(l-1)*dimp+1
      rr=dri*(r(lp)-rmin)+2.    rz=dzi*(z(lp)-zmin)+2.
      icen=rr      icen=min0(max0(icen,1),nr2)
      jcen=rz      jcen=min0(max0(jcen,1),nz2)
      if (skey(icen,jcen).eq.-1.) then
c $$$      Have found one!
      npart=npart-1      r(lp)=r(lstop)      z(lp)=z(lstop)
      vx(lp)=vx(lstop)  vy(lp)=vy(lstop)  vz(lp)=vz(lstop)
      go to 730
      else
      i(l)=rr-1.  fr(l)=rr-i(l)  j(l)=rz-1.  fz(l)=rz-j(l)
      w1(l)=(1.-fr(l))*(1.-fz(l))  w2(l)=(1.-fr(l))*     fz(l)
      w3(l)=    fr(l) *(1.-fz(l))  w4(l)=    fr(l) *     fz(l)
      go to 720
      endif
    800 nspec(n)=nspec(n)+npart      npthisb=npthisb+npart
                         SAME AS BIRTH
      return      end

```

Appendix 12. PARMOVR

David J. Larson's relativistic modification of PARMOV.

```
=====
 subroutine parmovr
c Relativistic version of Hewett's PARMOV DJLarson 7/6/90
c-----hewett, 1986
c      rotate=2.(0.) gives(stops) particle rotation
c      rotate=2.
c2i=1./C**2
c          Assume we start here with r(n),vx(n-1/2)
c          The fields E and B are at the (n) time level.
c          In this relativistic pusher, VX(LP) has the
c          value of GAMMA(LP)*velocity. DjL
do 70 lp=1,npthisb*dimp,dimp
rr=dri*(r(lp)-rmin)+1.          rz=dzi*(z(lp)-zmin)+1.
i=rr  fr=rr-i  frc=1.-fr        j=rz  fz=rz-j  fzc=1.-fz
w1=frc*fzc          w2=fz*frc    w3=fr*fzc          w4=fr*fz
c      do half of E push
eacclr/q/z=w1*er/q/z(i,j)+w2*er/q/z(i,j+1)
           +w3*er/q/z(i+1,j)+w4*er/q/z(i+1,j+1)
vx/y/z(lp)=vx/y/z(lp)+eacclr/q/z
gamma=sqrt(1.+(vx(lp)*vx(lp)+vy(lp)*vy(lp)+vz(lp)*vz(lp))*C2i)
gi=1./gamma
c          rotation follows
omgr/q/z=(w1*br/q/z(i,j)+w2*br/q/z(i,j+1)
           +w3*br/q/z(i+1,j)+w4*br/q/z(i+1,j+1))*gi
vx/y/zt=vx/y/z(lp)+vy/z/x(lp)*omgz/r/q-vz/x/y(lp)*omgq/z/r
fact=rotate/(1.+omgr*omgr+omgq*omgq+omgz*omgz)
vx/y/z(lp)=vx/y/z(lp)+fact*(vy/z/xt*omgz/r/q-vz/x/yt*omgq/z/r)
c          rotation finished
c      do remaining half of E push
vx/yh =vx/y(lp)+eacclr/q      vz(lp)=vz(lp)+eacclz
gamma=sqrt(1.+(vxh*vxh+vyh*vyh+vz(lp)*vz(lp))*C2i)
gi=1./gamma
c      now advance to r(n+1) in order to find vx(n+1/2) 4/3/90
x=r(lp)+vxh*dt*gi            dyyy=vyh*dt*gi
z(lp)=z(lp)+vz(lp)*dt*gi      r(lp)=sqrt(x*x+dyyy*dyyy)
costh=x/r(lp)                 sinh=dyyy/r(lp)
vx(lp)=costh*vxh+sinh*vyh     vy(lp)=-sinh*vxh+costh*vyh
70 continue
c          push completed, now TRAP LOST particles
call trap
c          After TRAP extrapolate vx(n+1/2) to vx(n+1) using new
c          positions r(n+1) and fields E(n+1/2) and B(n+1/2), in
```

```

c          order to accumulate sources at the desired time level
c          As of 5/9/90 have yet to update E & B to n+1/2 level.
do 270 lp=1,npthisb*dimp,dimp
rr=dri*(r(lp)-rmin)+1.      rz=dzi*(z(lp)-zmin)+1.
i=rr  fr=rr-i   frc=1.-fr    j=rz   fz=rz-j   fz=1.-fz
w1=frc*fzc      w2=fz*frc   w3=fz*fzc      w4=fr*fz
c          do half of E push with E(n+1/2)
eacclr/q/z=.5*(w1*er/q/z(i,j)+w2*er/q/z(i,j+1)
               +w3*er/q/z(i+1,j)+w4*er/q/z(i+1,j+1))
vx/y/zh=vx/y/z(lp)+eacclr/q/z
gammah=sqrt(1.+(vxh*vxh+vyh*vyh+vzh*vzh)*C2i)
gi=1./gammah
c          rotation follows using B(n+1/2) and r(n+1)
omgr/q/z=.5*(w1*br/q/z(i,j)+w2*br/q/z(i,j+1)
               +w3*br/q/z(i+1,j)+w4*br/q/z(i+1,j+1))*gi
vx/y/zt=vx/y/zh+vy/z/xh*omgz/r/q-vz/x/yh*omgq/z/r
fact=rotate/(1.+omgr*omgr+omgq*omgq+omgz*omgz)
vx/y/zh=vx/y/zh+fact*(vy/z/xt*omgz/r/q-vz/x/yt*omgq/z/r)
c          do remaining half of E push using E(n+1/2)
vx/y/zh =vx/y/zh+eacclr/q/z
gamman=sqrt(1.+(vxh*vxh+vyh*vyh+vzh*vzh)*C2i)
gi=1./gammah      vx/y/znp1=vx/y/zh*gi
den(i ,j ,n)=den(i ,j ,n)+w1
den(i+1,j ,n)=den(i+1,j ,n)+w3
den(i ,j+1,n)=den(i ,j+1,n)+w2
den(i+1,j+1,n)=den(i+1,j+1,n)+w4
ur/q/z(i ,j ,n)=ur/q/z(i ,j ,n)+w1*vx/y/znp1
ur/q/z(i+1,j ,n)=ur/q/z(i+1,j ,n)+w3*vx/y/znp1
ur/q/z(i ,j+1,n)=ur/q/z(i ,j+1,n)+w2*vx/y/znp1
ur/q/z(i+1,j+1,n)=ur/q/z(i+1,j+1,n)+w4*vx/y/znp1
tloc=vxnp1*vxnp1+vynp1*vynp1+vznp1*vznp1
rhs(i ,j )=rhs(i ,j )+w1*tloc
rhs(i+1,j )=rhs(i+1,j )+w3*tloc
rhs(i ,j+1)=rhs(i ,j+1)+w2*tloc
rhs(i+1,j+1)=rhs(i+1,j+1)+w4*tloc
270 eionr/q/z=eionr/q/z+vx/y/znp1*vx/y/znp1
      return           end

```

Appendix 13. Santa Fe Numerical Simulation Abstract

The following was presented at the 13th Conference on the Numerical Simulation of Plasmas in Santa Fe, NM, September, 17-20, 1989.

An Algorithm for Particle Reflection from Internal Boundaries on Orthogonal Uniform Meshes

D.W. Hewett, H.L. Rutkowski,* and S. Humphries, Jr.**

University of California

Lawrence Livermore National Laboratory

Livermore, California 94550

Internal structures greatly expand the utility of the PIC method. Recently such structures have been incorporated in an electrostatic R-Z code whose primary mission is to simulate the interaction between a plasma and a wire grid. The grid is held at voltages chosen to confine plasmas density buildups in an optically useful shape until larger external voltages are applied to extract ions. The primary motivation for this effort is the LBL source development effort for proposed heavy ion accelerators that are of interest as possible ICF drivers. We also anticipate using these procedures in codes using more efficient and general algorithms for the Darwin field model (see Larson and Hewett, these proceedings).

The algorithm we describe here is intended to simply detect PIC particles that have moved into regions defined as "inside" a structure. This procedure also serves as a monitor of particles leaving the simulation domain if the external boundary is defined to be a "structure" as well. As a particle incursion is detected, the most difficult task geometrically is to determine which surface it came through and how to reflect it if the entry point is near a corner. Three other commonly particle boundary conditions, absorption, injection/field emission, and thermal reemission, are either less demanding geometrically or special cases of the reflection condition.

In the discussion that follows, it is assumed that we have an orthogonal mesh with uniform spacing and the boundaries of the internal structures as well as the external domain pass through corners of the two dimensional cells.

1) Detection of "lost" particles

After a group or block of particle has been advanced in time, correct particle physics accounting requires some method to determine if the particle still lies within the acceptable domain. If correct physics is not sufficiently compelling, scattering particle attributes throughout computer memory during the accumulation stage provides stronger motivation. Our procedure is first to determine the cell CENTER indices nearest the particle for each particle and check a structure properties array with those indices, here called KEY(Icen,Jcen), is to determine the contents of that cell. We use the code 1,0,-1 for plasma, vacuum, and structure, respectively, and the -1 values are set at the start of the run. If KEY(Icen,Jcen)=-1 then the cell center indices closest to the particle's present position are the indices of a cell whose volume is within a structure; therefore the particle

* Lawrence Berkeley Laboratory, Berkeley, CA 94720

** University of New Mexico, Albuquerque, NM 87131

itself is within that structure and must be processed. If these indices are themselves completely outside the domain of interest, then the particle is truly lost and special handling is required. Such cases cannot occur if rows and columns of "structures" are defined around the external boundary and if particles are moving less than a cell per time step. Special circumstances can cause the occasional particle far out on the thermal tail to escape-we log these particles and delete them from the simulation-but if more than a few particles get lost, a thorough understanding of their origin is needed.

Simple, unoptimized FORTRAN that accomplishes the detection of lost particles for a group of KLAST particles is given here.

```

k=0
120 k=k+1
130 if (k.gt.klast) go to 200           {definitions}
      icen=dri*(r(k)-rmin)+2.            {dri=1/dr}
      jcen=dzi*(z(k)-zmin)+2.            {dzi=1/dz}
      if (icen.lt.1 .or. icen.gt.nrmax .or.
          jcen.lt.1 .or. jcen.gt.nzmax) then
c          Particle completely outside mesh...Log and Delete
          go to 130
      endif
      if (key(icen,jcen).gt.-1) go to 120
c          Have found one!

```

Most particles will generally be inside so that steps requiring the most care in codes for speed (vectorization, parallelizing, etc.) are limited to these steps already discussed. What we do to a particle if it is inside a structure is not as constrained by minimizing CPU cycles. We now discuss handling of those particles that are inside of a structure.

2) Lost particle processing

The first issue to address after finding a particle in a structure is determine through which surface it passed to get inside. The trick we use, after first verifying that the particle has not moved more than a cell in the last time step, is to move the particle backward in time one-half of a time step and then determine to which cell CORNER it is closest.

```

partCFLr=abs(dri*vr(k)*dt)
partCFLz=abs(dzi*vz(k)*dt)
if (partCFLr.ge.1. .or. partCFLz.ge.1.) then
c          Particle TOO Fast.....Log and Delete
      klast=klast-1
      go to 130          (New TRAP in Sec. IV.A.2.b catches
      endif              these TOO fast particles. 3/21/91)
c          Back DTH=DT/2 to find closest corner at ICLS,JCLS.
      icls=dri*((r(k)-dth*vr(k))-rmin)+1.5
      jcls=dzi*((z(k)-dth*vz(k))-zmin)+1.5

```

The orientation of the surface through which the particle has passed can be obtained as a function of the closest-corner indices Icls,Jcls from an array KORNT set up at the start of the run. KORNT specifies the surface orientation, as defined in Fig. 1, through the following system:

KORNT	Normal Direction into the PIC Region
+1, -1, +10, -10	+R, -R, +Z, -Z
+11, -11, +9, -9	(+R, +Z), (-R, -Z), (-R, +Z), (+R, -Z)

The reflections are themselves simple processes. To illustrate, consider R-reflections. To reflect the Kth particle in R, we reverse the sign of the R velocity and place the particle to the left (or right) of the boundary the same distance it has moved to the right (left) of the boundary. The two FORTRAN statements,

```
r(k)=2.*rgc(icls)-r(k)
vr(k)=-vr(k)
```

accomplish the reflection where the R-boundary is located by the grid array RGC(Icls). Remembering that the particle in question is presently inside a boundary cell, we can determine easily when an R-reflection is NOT needed as follows:

- If $| \text{KORNT}(Icls, Jcls) | = 10$, the particle has crossed a surface with a surface normal purely in the Z-direction.
- Only if $[R(k) - \text{RGC}(Icls)] * Vr(k) > 0$ has the particle crossed the R-boundary in question. (This condition is necessary but not sufficient for some cases. see below.)

3) Reflections from Inside and Outside Corners

If the closest cell corner Icls,Jcls has a KORNT that is +11,+9,-9, or -11, either an R- or Z-reflection or both-depending on the particle incident direction-may be required to obtain the particle reflection. If we find a situation in which the criteria for both R- and Z-reflection are met, both reflections are done. This amounts to reversing the sign of both components of velocity and, except for a small lateral translation, the particle is returned in precisely the same direction. We prefer the particle be reflected such that the angle of incidence equal the angle of reflection with respect to the surface normal that, at a corner, depends on the relative size of the grid spacing DR and DZ. The procedure we use, if both an R- and Z-reflection have been done, is to form the dot product of the outgoing velocity V_o with the unit vector lying in the

```
R-reflect if           vr><0 | vr><0 | vr><0 | vr><0
(R(k)-RGC(ICLS))*VR >=0   vz><0 | vz<0 | vz<0 | vz><0
    and
    -----+-----+-----+-----+
KORNT*(|KORNT|-10)*VR <0   vr>0 | 9 // 10 // 11 | vr<0
    or
    -----+-----+-----+-----+
    KORNT = +-1
    -----+-----+-----+-----+
Z-reflect if           |//////////|//////////|
(Z(k)-ZGC(JCLS))*VZ >=0   vr>0 |//////////|//////////| vr<0
    and
    -----+-----+-----+-----+
    KORNT*(!KORNT!-1)*VZ <0   vz><0 | -11 // -10 // -9 | vz><0
    or
    -----+-----+-----+-----+
    KORNT = +-10
    -----+-----+-----+-----+
```

Figure 1

V_o with the unit vector lying in the surface face with components proportional to DR and DZ, respectively. The result is the magnitude of the V_o component tangential to the

face. The desired outgoing velocity is obtained by subtracting this vector twice from V_o . This algorithm is expressed by the following FORTRAN.

```
c          ok PERFECT REFLECTION-orient the surface with KORNT
flect2=0.
ikornt=iabs(kornt(icls,jcls))
if (ikornt.eq.1 .or. ((r(k)-rgc(icls))*vr(k)).ge.0. .and.
    kornt(icls,jcls)*(ikornt-10)*vr(k).lt.0)) then
    r(k)=2.*rgc(icls)-r(k)
    vr(k)=-vr(k)
    flect2=2./(dr*dr+dz*dz)
endif
if (ikornt.eq.10 .or. ((z(k)-zgc(jcls))*vz(k)).ge.0. .and.
    kornt(icls,jcls)*(ikornt- 1)*vz(k).lt.0)) then
    z(k)=2.*zgc(jcls)-z(k)
    vz(k)=-vz(k)
c Have we already done an R-reflection? If so fix outgoing.
sdz=(10-ikornt)*dz
svr =vr(k)-flect2* dr*(vr(k)*dr+vz(k)*sdz)
vz(k)=vz(k)-flect2*sdz*(vr(k)*dr+vz(k)*sdz)
vr(k)=svr
endif
c               Next Particle, Please..
go to 120
200 continue
```

The extra logic involving KORNT is less straightforward but is needed to manage special cases that arise when inside corners as well as outside corners (3 quadrants of "structure" for inside, only 1 for outside) are considered. The differences arise from those particles that lodge in one of the two extra quadrants, thus needing one but not both reflections.

The next task is to incorporate these ideas into a vectorized particle pusher.

Acknowledgements

Work performed under the auspices of the United States Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

Appendix 14. GYM28 Summary Sheets

GYMNO\$ (GYM28) HELP 1/3/91 units----CGS, eV
 parameter (nr2=73,nz2=19,mxg1d=73,nstmax=8,ithmx=2000,kcormx=700)
 parameter (nheadb=10,mpbuf=1000,dimp=5,memoly=80160,nsp=2)
 For disk storage, just set MEMOLY=0. For memory storage,
 $\text{MEMOLY} = \text{sumNSP} [1 + (\text{MSPEC}(n)/\text{MPBUF})] * (\text{NHEADB} + \text{MPBUF} * \text{DIMP})$.
 NREST !=0 write a restart every |NREST| DTs and at TIME=TMAX.
 =0 no restart read or written. <0 start from restart
 NUMREST is always the restart number you will write next time.
 IMSG =1[0] do[n't] process keyboard messages: choices IOTTY N ABORT
 AUTOVLM auto phase space (Vx-x) plot scaling
 = 1. includes 0 and "growth" factor over max from IT-1
 =-1. scaling will stretch but not shrink !=|1.| fixed
 //GEOSSET-----STRUCTURE SPECIFICATION ARRAYS-----
 RG ZG VOL(Rmn,Rmx corrected) cell cornrs RGCC ZGCC VOLCC cell cntr
 NSTRUCT(nstr,3) STRGEN(nstr,6) KCIJ(kcor,2) KCORR(ijcor)
 KORNT(ijcor) SKEY(ijcen) KEY(ijcor)
 User sets NSTRUCT(nstr,1) and STRGEN (can read with JSTGNSW)
 calls GEOMSET to set NSTRUCT(nstr,2/3), KCORR, KCIJ->SKEY->KORNT & KEY
 NSTRUCT(NSTR,1)--name of structure numbered NSTR
 PIE STRGEN(NSTR,1/2/3/4/5/6) 1/rcen/zcen/radius/astr/astp (a=0 e_r)
 BAR STRGEN(NSTR,1/2/3/4/5/6) 2/rstr/zstr/rstp/zstp/hwidth
 NSTR=1,2,3,4 generally Rmin,mx,Zmn,mx CORNMN/MX=0(1) nclds cornr
 CAUTION: Boundaries are built only at cell edges!
 JSTGNSW=1 use STRGEN,NSTRUCT,NSTRMX as set in BCSET or =0(-1) read tty(6)
 NSTRUCT(NSTR,2/3)--KCOR start/stop pt in KCIJ.
 ex: do # kcor=nstruct(nstr,2),nstruct(nstr,3)
 # key(kcij(kcor,1),kcij(kcor,2))=-1
 IorJCOR=KCIJ(KCOR,1or2) for structure corner pt KCOR(=KCORR(ijcor) xref)
 -----Orient Bndry surface points with KORNT-----
 KORNT(icor,jcor) =0 not a B surface pt.
 = +1(-1) for surface normal in for(back)ward R-direction
 =+10(-10) for surface normal in for(back)ward Z-direction
 /| +11 -----+ +9 /| +-----
 /+---- //||| | -----+ /| //|||
 //||| +12 /| //||| /| +8
 4/90
 |/ -12 //||| -8 /| //|||
 ||||| -----+ /| //||| /+----
 +---- -11 /| -----+ /| -9
 Plasma GEOMETRY ARRAYS--(plasma/vacuum set n DENSET)
 SKEY(icen,jcen) ==+1. if DEN>=CTOFF (in plasma) =0. if vacuum
 =-1. if IN boundary (only 1./0. in periodic ghost cell)
 KEY(icor,jcor) ==+1 if in the plasma, = 0 if in a vacuum

```

--1 if ON or IN bndry (periodic bndry 1or0 only)
//PBCSET-----PRTCL/BEAM SPECIFICATION ARRAYS-----
    BEAM(nstr,nsp,8)          IPBC,QABSRB(ijcor) TEM(ijcor,nsp)
    UNBSTR,TEMPSTR,QBRS(kcor,lorD,nsp)

Given NSTRUCT,KCIJ, the user inputs NSTPROP TEMIN BEAM,
& this routine sets IPBC UNBSTR TEM TEMPSTR QBRS NPFEML NPNJCT
NSTPROP(NSTRUCT,Nspecies)--n00-n50, n51-%THREMT,PRFLCT
    (n=1/2/0 Femt/Njct/Neither), negative=no dithr
IPBC(ijcor) = cell corner particle property = last 2 digits NSTPROP.
    61 returns IFF reduces QABSRB total on surface. (NOT YET ever?)
QABSRB(icor,jcor) = Absorbed charge at this CELL CORNER.

THREMT...Reemit .02*|IPBC|*QABSRB/SQ using TEM & ETNVTP
    (from BEAM(nstr,n,4) ) & DITHR from sign of NSTPROP(nstr,n).
Normal velocity=0                      TEM(ijcor,lorD,n)= Txvrs.
NJCT/FEMT...reservoir charge QBRS(kcor,LorD,n) at this CELL FACE.
    LorD=1(2) [emitting surface Left(Down) from corner]
BEAM(nstr,n,1&      2&      3&      4&      5&      6&      7&      8)
    Bden,unbstr,temstr,etnvtp, turnon,rise,turnof,fall
    UNBSTR(kcor,lorD,n)=Nrml vlcty mag. TEMPSTR(kcor,lorD,n)= Txvrs.
Tnormal/Txvrs from BEAM(nstr,n,4) & DITHR from sgn NSTPROP for all.

E.G., to absorb & THREMT 50% of the prtcls species n=3 that hit NSTR &
njct a DITHRed beam with Bden=1e12 & Vnormal=3C/4 & 3eV xverse thermal,
on instantly after 1.5 ns, remaining on until 5 ns & falling linearly
to 0 by 6.2 ns with anisotropy Tn/Txrs of 30%, use temin(3)=3.
    nstprop(nstr,3)=225   beam(nstr,3,1)=1.e12   beam(nstr,3,2)=7.5e9
    beam(nstr,3,3)=3.     beam(nstr,3,4)=.3     beam(nstr,3,5)=1.5e-9
    beam(nstr,3,6)=0.     beam(nstr,3,7)=5.e-9   beam(nstr,3,8)=1.2e-9

//FLDSET-----Given NSTRUCT, KCIJ user sets BV,BVR,BVZ & KN (all IJCOR)
KN(icor,jcor)=0, Dirichlet pt with value=BV(icor,jcor) on&in bndry.
    =+1(-1) on bndry w/ for(back)ward R-drivtv [=+2(-2) curl-type]
    =KN+10(-10) on bndry w/ for(back)ward Z-drivtv
BVR/Z(icor,jcor) = Neumann value for the R/Z derivative.
    VOLT(NSTR,R/Q/Z,1/2/3/4/5) mag/turnon/rise/turnof/fall
    PHISTR(NSTR,0/1/2/3/4/5)     mag0/mag/turnon/rise/turnoff/foff
        MAG is the voltage ultimately added to MAG0

//INIT/INPROF---INPROF quantites for prtcl ntlnztn are CELL CENTERED!
NPINIT(n)=NP...# of sprtcls rqstd.....NSPEC(n)...# at any time
RPPSP(n) is obtained from TOTNRP/NSPEC(n) at time=0 if NPINIT(n)!=0
    If NSPEC(n)=0, assume all injected beam densities uniformly fill
    region to give REFNRP then RPPSP(n)=REFNRP/MSPEC(n).

def: Vth=sqrt(boltzk TEM/smsngl(n))    Vmp=sqrt(2 boltzk TEM/smsngl(n))
    Pressure=VD3 DEN boltzk TEM    VD3 is the velcty space dimensnality
    ipbuf(1)=numbuf    ipbuf(2)=npthisb    ipbuf(3)=nbspec(n)    ipbuf(4)=n

```

```

ipbuf(5)=namsp(n)    ipbuf(6)=it      pbufh(8)=sq(n)    pbufh(9)=sm(n)
//TRANS-----orchestrates particle advance and source accumulation
TRANS converts "raw counts" to densities by dividing by EFFVOLs with
by Rfactors at R bndys complete with in/outside corner distinctions.
RFACOT=3(4 rg(i)+dr)/4(3 rg(i)+dr)   RFACIN=3(4 rg(i)-dr)/4(3 rg(i)-dr)
Two species plotted as same species if a) they are adjacent
in the buffering, b) they have the same NAMSP(n), & c) they
have the same SQ(n) & SM(n)

///PARMOVers----All movers use Boris that assumes we start with
R(n), VX(n-1/2) & fields E and B are at the (n) time level.
    PARMOV     VPARMOV     PARMOVE     PARMOVR
    fullEM     fullEMv     ES          fullEMrelativistic
E & B have been initialized by TRANS to carry QVM*DTH*E & QVM*DTH*B/C
Calls TRAP after prtcls pushed to 1) insure none have invalid
locations, 2) to reflect or absorb/THREMT or FEMT or NJCT
Tried to rotate the n+.5 velctis back to Q=0 with n+.5 positions but--
changed to original Boris algorithm using n+1 positions to avoid a
density pile-up on axis. 4/3/90
///TRAP---insures valid prtcl locations/ IPBC(ijcor)--set for each NSP
If a prtcl is in cell where SKEY is -1., move or deleted it.
If perfect reflection, put back inside with bounce off boundary,
even if prtcl exceeds CFL. Deleted partcls add charge to QABSRB(ijcor)
Perfect reflection for prtcls with CFL>1 via back cycles with smaller
CFLok DTCLS to find where last prtcl crossing occurred. Multiple
passes catch those whose rflctn puts them in yet another structure.
THREMT..QABSRB(ijcor cls)=resevoir 4 each NSP--TEM(ijcor,n)=Txvrs..Unrml=0
Reemit .02*|IPBC|*QABSRB/SQ .....CORNERS
NJCT/FEMT...QBRs,TEMPSTR,UNBSTR(kcor,lord,n)...resevoir,Txvrs,Unrml..FACES
Tnormal/Txvrs from BEAM(nstr,n,4) & DITHR from sgn NSTPROP for all.
c JK Boyd's trick to FEMT .25 cell away from bndy (else den infinite)

```

**DATE
FILMED**

11 / 3 / 93

END

