



Spring 5-2004

## **General Curvilinear Coordinate Transformation and Visualization for Plasma Equilibria**

Katherine Fiona White  
*University of Tennessee - Knoxville*

Follow this and additional works at: [https://trace.tennessee.edu/utk\\_chanhonoproj](https://trace.tennessee.edu/utk_chanhonoproj)

---

### **Recommended Citation**

White, Katherine Fiona, "General Curvilinear Coordinate Transformation and Visualization for Plasma Equilibria" (2004). *Chancellor's Honors Program Projects*.  
[https://trace.tennessee.edu/utk\\_chanhonoproj/805](https://trace.tennessee.edu/utk_chanhonoproj/805)

This is brought to you for free and open access by the Supervised Undergraduate Student Research and Creative Work at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Chancellor's Honors Program Projects by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact [trace@utk.edu](mailto:trace@utk.edu).

# UNIVERSITY HONORS PROGRAM

## SENIOR PROJECT - APPROVAL

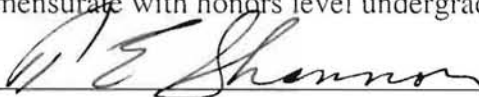
Name: Katherine White

College: Arts and Sciences Department: Mathematics

Faculty Mentor: Dr. Tom Shannon, Mechanical & Aerospace Engineering Sciences

PROJECT TITLE: General Curvilinear Coordinate Transformation and Visualization  
for Plasma Equilibria

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: , Faculty Mentor

Date: 5/6/03

Comments (Optional):

# **General Curvilinear Coordinate Transformation and Visualization for Plasma Equilibria**

**Date April 21, 2003**

**Prepared by  
Katherine White  
HERE student, Fusion Energy Division**

#### DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via the U.S. Department of Energy (DOE) Information Bridge:

**Web site:** <http://www.osti.gov/bridge>

Reports produced before January 1, 1996, may be purchased by members of the public from the following source:

National Technical Information Service  
5285 Port Royal Road  
Springfield, VA 22161  
**Telephone:** 703-605-6000 (1-800-553-6847)  
**TDD:** 703-487-4639  
**Fax:** 703-605-6900  
**E-mail:** [info@ntis.fedworld.gov](mailto:info@ntis.fedworld.gov)  
**Web site:** <http://www.ntis.gov/support/ordernowabout.htm>

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange (ETDE) representatives, and International Nuclear Information System (INIS) representatives from the following source:

Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831  
**Telephone:** 865-576-8401  
**Fax:** 865-576-5728  
**E-mail:** [reports@adonis.osti.gov](mailto:reports@adonis.osti.gov)  
**Web site:** <http://www.osti.gov/contact.html>

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States government nor any agency thereof, 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 product, 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 any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL-XXXX

**GENERAL CURVILINEAR COORDINATE TRANSFORMATION AND VISUALIZATION  
FOR PLASMA EQUILIBRIA**

Katherine White

Date Published: (to be published Summer 2003)

Prepared by  
OAK RIDGE NATIONAL LABORATORY  
P.O. Box 2008  
Oak Ridge, Tennessee 37831-6285  
managed by  
UT-Battelle, LLC  
for the  
U.S. DEPARTMENT OF ENERGY  
under contract DE-AC05-00OR22725



## CONTENTS

	<b>Page</b>
ACKNOWLEDGEMENTS .....	v
ABSTRACT .....	1
1. OVERVIEW OF THE ORIGINAL CODE.....	1
2. UPDATING THE CODE	
2.1 GENERALIZED CODE INPUTS.....	2
2.2 GENERALIZED THE GRID .....	2
2.3 MODERNIZATION AND VISUALIZATION OF CODE OUTPUTS .....	2
3. IMPROVING THE MAGNETIC FIELD VISUALIZATION	
3.1 ADDING TAYLOR SERIES TERMS .....	3
3.2 STAGGERED MESH SOLUTION .....	4
3.3 INVALIDATING DATA .....	5
4. IMPROVING THE RUNTIME OF THE CODE .....	6
5. SUGGESTED FURTHER IMPROVEMENTS .....	7
REFERENCES .....	8
APPENDIX A. TEXT OF vmec_setup.f .....	9





## **ACKNOWLEDGMENTS**

Thanks to Mark Carter, Fred Jaeger, Don Spong, Nermin Uckan, and Steve Hirshman at Oak Ridge National Laboratory Fusion Energy Division, and to Tom Shannon, Mechanical and Aerospace Engineering Sciences at the University of Tennessee, Knoxville.



## ABSTRACT

The Variation Moments Equilibrium Code (VMEC)<sup>1</sup> written by Steve Hirshman outputs data for plasma in a stellarator geometry. The visualization of such data is of use to theorists working on improving codes, engineers designing antennas, and also in papers to explain the geometry of a physics problem. Visualization of the VMEC output requires a coordinate transformation from the curvilinear coordinates of the VMEC code into cylindrical coordinates. A Fortran code had been written (by Fred Jaeger) to perform such a transformation and output data in cylindrical coordinates for visualization (`vmec_setup.f`), but the code was in need of improvement. Inputs into `vmec_setup.f` were not compatible with the latest VMEC release, `vmec_setup.f` did not use a generalized grid, and outputs from `vmec_setup.f` were in an obsolete two-dimensional format. Magnetic fields and streamlines were visualized, but for general use, transformation of the magnetic fields needed to be improved. The original implementation did not preserve divergence-free magnetic fields. A possible solution to the problem of preserving the divergence-free magnetic fields has been found, but not yet implemented. A final problem with the original code was the runtime. These problems of updating the code and improving the runtime have been solved. The purpose of this paper is to detail the steps taken in solving problems in the `vmec_setup.f` code and to offer ideas for the future improvement of the code.

### 1. OVERVIEW OF THE ORIGINAL CODE

The `vmec_setup.f` code was written by Fred Jaeger to transform the data from VMEC output in curvilinear coordinates to cylindrical coordinates. The original code was about 2000 lines long, containing a main program, and subroutines `transform`, `deriv_psi`, and `deriv_theta`.

The main program reads data values for the magnetic fields and for the  $\psi$  values from the VMEC output file. It then sums the Fourier harmonics for the major radius ( $R$ ), the elevation ( $Z$ ), and the magnitude of the magnetic fields at specific  $\psi$ ,  $\theta$ , and  $\phi$ . Then, because the VMEC magnetic field data uses a staggered mesh, it interpolates the magnetic field data onto a whole mesh. This interpolation step was not recognized as a problem until late in the project, and it is a likely source of difficulty in maintaining divergence-free properties as will be discussed in section 3.2. It then calls the `transform` subroutine to perform the coordinate transformation and output the data in a two-dimensional (the old code) or three-dimensional (the newest code) visualization.

The subroutine `transform` creates a grid based on user defined values for the number of nodes in the  $r$ ,  $z$ , and  $\phi$  directions. Then it calls the subroutines `deriv_psi` and `deriv_theta` to calculate derivatives (on the VMEC grid). The subroutines `deriv_psi` and `deriv_theta` calculate derivatives with respect to  $\psi$  and  $\theta$  using a second order centered finite differencing method. Once the derivatives are calculated, the code creates a table of values from VMEC for visualization. First, it finds the VMEC coordinates ( $R$  and  $Z$ ) nearest neighbor to each mesh point on the user defined grid ( $x$  and  $y$ ). This nearest neighbor value is used as the first guess to the data value at each particular mesh point. To get a second guess, the  $R$  and  $Z$  functions of  $\psi$  and  $\theta$  were Taylor expanded (only to the first derivative term in the original code) about the particular point (nearest neighbor)  $\psi_k$  and  $\theta_k$ . Then the Taylor expanded equations are solved for  $\psi$  and  $\theta$ . This is the second guess. If the Jacobian of  $R$  and  $Z$  is equal to zero, then the nearest neighbor ( $\psi_k$ ,  $\theta_k$ ) is used for the second guess. Once a value of  $\psi$  and  $\theta$  is found at a particular  $r$ ,  $z$  location, the data is found at that  $r$ ,  $z$  location for the  $b_x$ ,  $b_y$ , and  $b_z$  component of the magnetic fields. This is done by Taylor expansion. Then the magnitude of the magnetic field at each point is calculated, and the data is visualized. The original code visualized data using a program called `PLplot`.

## 2. UPDATING THE CODE

### 2.1 GENERALIZED CODE INPUTS

The `vmec_setup.f` code used inputs from an old release of VMEC that was no longer standard. The latest version of VMEC outputs data in a standard "netcdf" data format. The `vmec_setup.f` code read data in a user defined data format called a "wout" file. Don Spong has written some libraries of functions to read the netcdf data format and create arrays of the data in Fortran. These functions were implemented in the `vmec_setup.f` code to make it read the new netcdf format. One problem created by this change is that now data contained in old "wout" files cannot be read by the newest version of `vmec_setup.f`, but such improvements have been made to the VMEC code that data contained in wout files is relatively obsolete. If there is a need to visualize old data sets, this can be solved by rerunning the VMEC code on the data set to create a new netcdf data file instead of a wout file. One can also use outdated files by running an old version of `vmec_setup.f` which has the updated generalized grid and Opendx output but not the updated netcdf input capability.

### 2.2 GENERALIZED THE GRID

The original `vmec_setup.f` code also read inputs from a file called `aorsa3d.in`. The purpose of these inputs was to setup an exact match to the grid used in the All Orders Spectral Algorithm (AORSA) code. However, not all users of VMEC will want to use the AORSA grid, and it would be useful to have to grid defined by the user in a less complex file format than the `aorsa3d.in` file. So the module `modeparams.f` was written. This module is only 8 lines long, and it allows the user to input values for the number of nodes in the x, y, and  $\phi$  direction that they would like in their final output. Since this code is a module, it can be included with other libraries needed to run the code.

### 2.3 MODERNIZED AND VISUALIZED CODE OUTPUTS

The original `vmec_setup.f` code visualized the transformed data using a program called PLplot. The program was useful for visualization of contour lines of  $\psi$  and the magnitude of the magnetic fields, but the version being used only visualized in two dimensions, and it only used eight-bit color. The eight-bit color was a problem for users using monitors made in the last few years. Many are set to a default of 256 colors or more, and PLplot would not run under such settings. There are newer versions of PLplot available, but OpenDx is becoming more widely used in the Fusion Energy Division.

The decision was made to use OpenDx, a visualization package developed by IBM and now an open source package. OpenDx has the capability of visualizing data in three dimensions, or four dimensions if series data is taken over time. It also had none of the color problems associated with PLplot. Another reason for choosing the OpenDx package was an already existing module (`dxdump.f`) written by myself which takes data from a Fortran code and outputs it in a native OpenDx format which made importing data into OpenDx much easier.

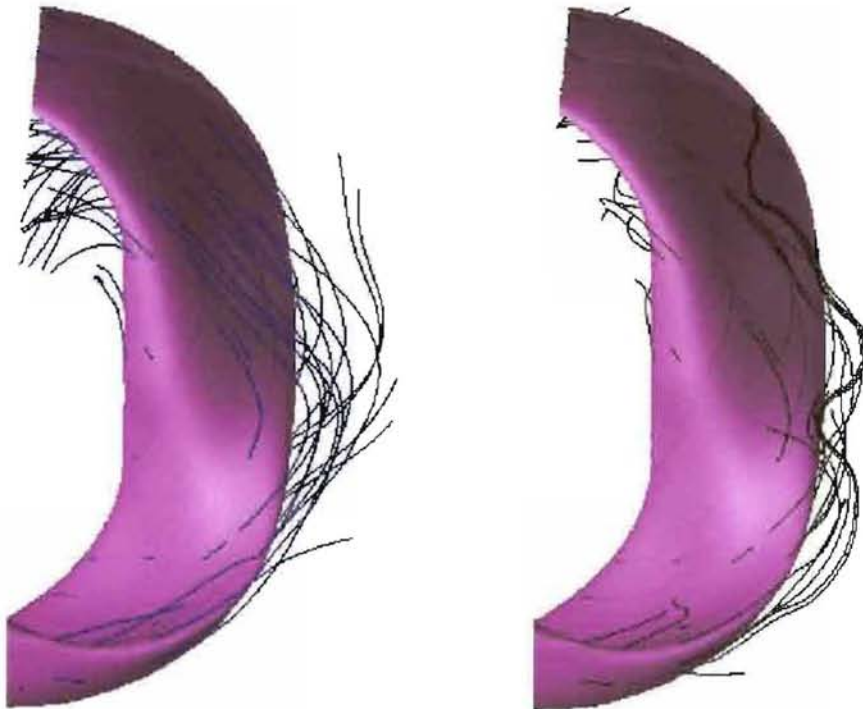
The `vmec_setup.f` code now makes calls to the subroutines contained in the `dxdump` module and writes out native OpenDx files of data for  $\psi$  and the magnetic fields. Programs have been written in the OpenDx visual program editor which visualize flux surfaces, magnetic fields, and streamlines.

### 3. IMPROVING THE ACCURACY OF THE CODE

#### 3.1 ADDING TAYLOR SERIES TERMS

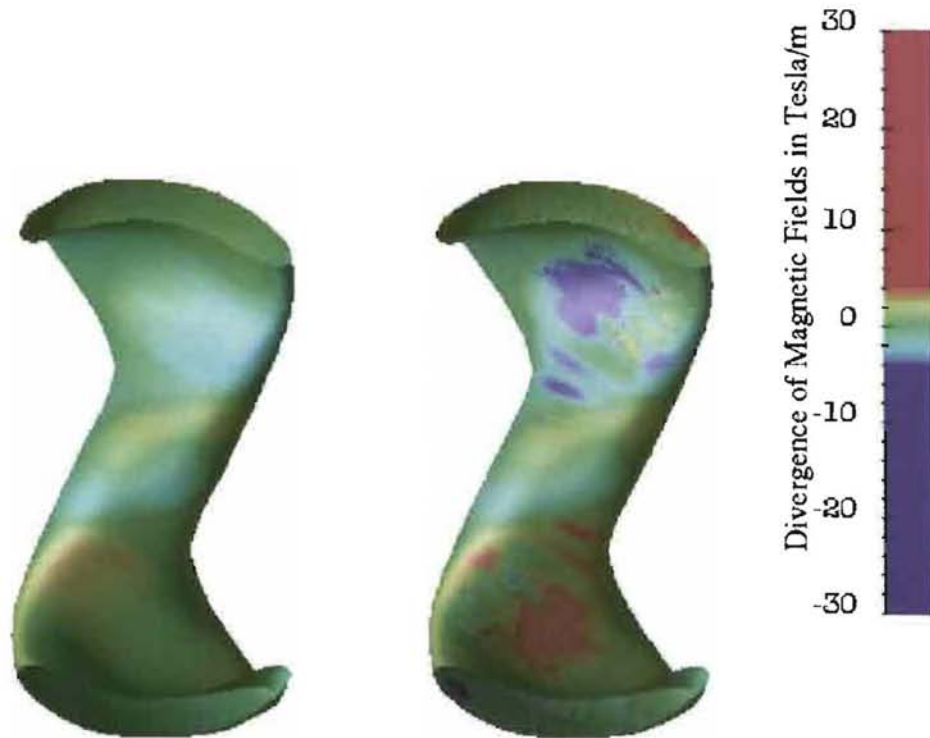
While visualizing streamlines and magnetic vector fields it was discovered that the data for the magnetic fields was not accurate enough to make reasonable streamlines for a divergence free field. The divergence of the magnetic field data was not sufficiently close to zero. It was thought that a way to fix this might be to make the code more accurate by adding second order terms of the Taylor series expansion. (The data values for each node on the grid are found by Taylor expanding R and Z about the nearest neighbor on the VMEC grid).

All of the Taylor series expansions were extended to include second order terms, using Maple<sup>2</sup>. So, it was necessary to write subroutines to calculate the second order derivatives, so `deriv_psi2` and `deriv_theta2` were written. They take the resulting arrays from `deriv_psi` or `deriv_theta` and again use a second order centered finite differencing method to calculate the derivative.



**Figure 1 Streamline comparison at  $\psi=0.8$ .**

The resulting data using the second order Taylor series expansions was slightly better than the previous data. Streamlines visualized did not appear to diverge completely from the flux surface, but they were still not a reasonable output for a divergence free field. Figure 1 shows a comparison of streamlines on the flux surface  $\psi$  equals 0.7. The divergence of the field using the second order Taylor series expansion was inaccurate as well. For the most part, the data was improved or the same compared to the first order Taylor series expansion, but on the inside of the torus there were some significantly high and low numbers in the field. Figure 2 shows the divergence of a section of the torus using first order Taylor series data (on the left) and a section of the torus using second order Taylor series data (on the right).



**Figure 2 Divergence comparison at  $\psi=0.8$ .**

Clearly, the second order terms of the Taylor series, based on the original mesh, did not significantly improve the behavior of the magnetic field lines. However, this visualization result led to the identification of the loss of divergence properties in the interpolation step in the main program (as described in section 1).

### 3.2 STAGGERED MESH SOLUTION

After it was found that adding second order Taylor series terms and invalidating extrapolated data were not enough to improve the streamline visualizations, the magnetic field data was traced through code. It had been thought that the data was not accurate enough after transformation in the transform subroutine, but now the idea was explored that the data might be corrupted prior to entering the transform subroutine.

While tracing through the code it was discovered that the magnetic field data was being interpolated onto the user defined grid using a linear interpolation method. Therefore, all terms other than the first order terms of the Taylor series of the magnetic field were inaccurate. This explained the high divergence for the data using the second order terms (shown in Figure 4). Adding in the second order terms for the magnetic field data resulted in adding in more erroneous data.

This problem was discovered too late to correct it before the writing of this paper, but it does give new insight in how to improve the visualization of magnetic field data. Because the magnetic field data is on a staggered mesh, new differentiation subroutines must be written to take derivatives of the data. Once

the derivatives are corrected, the Taylor series expansions can be corrected, and the the final result should have a magnetic field divergence of almost zero.

### 3.3 INVALIDATING DATA

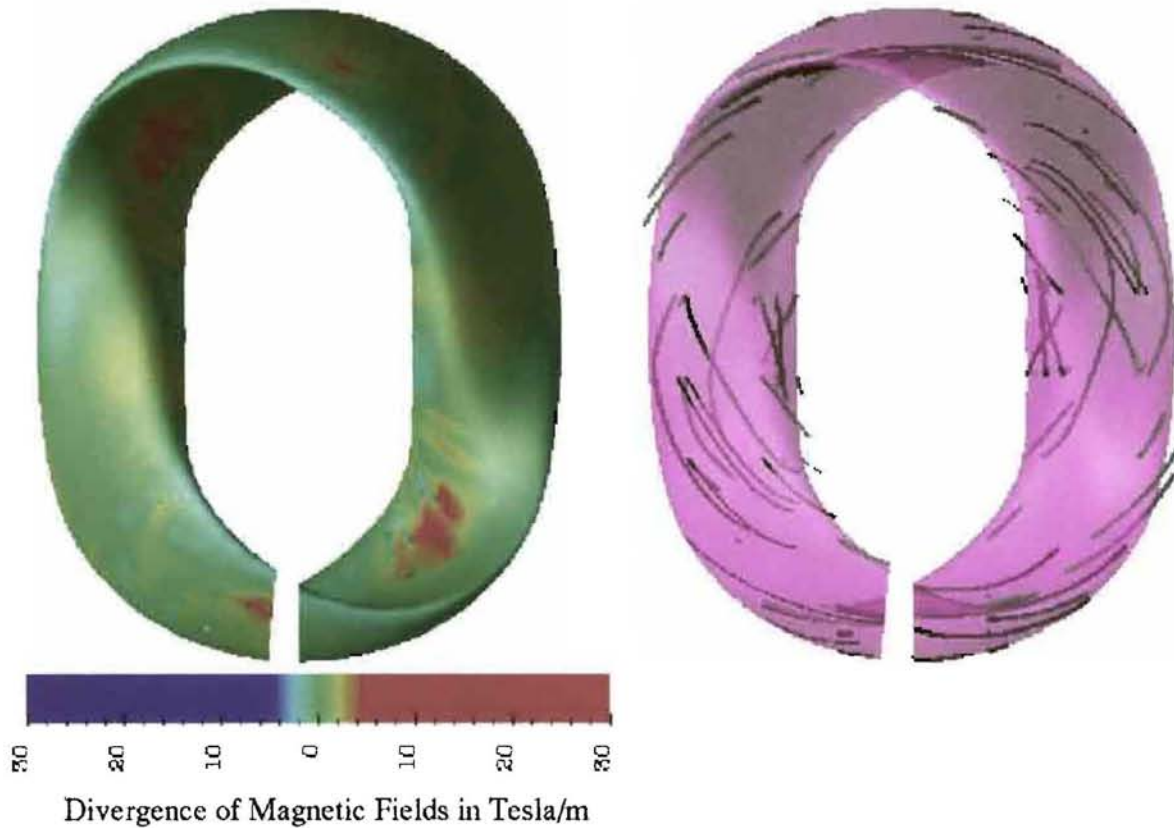
Regardless of the accuracy of the interpolation, the issue of extrapolation beyond the last closed flux surface ( $\psi=1$ ), must also be addressed. VMEC output is only accurate for those values within the  $\psi=1$  flux surface, but the generic grid that `vmec_setup.f` outputs on has values outside that surface. Data values for those positions outside  $\psi=1$  are extrapolated from the data on the last flux surface. So it was thought that another way to improve the streamlines would be to mask the bad data gained from extrapolating outside the  $\psi=1$  flux surface. Figure 3 shows a picture of valid and invalid positions. The gray area is all of the data, but only the purple area contains valid positions. So, for the magnetic field visualizations, the position data for  $\psi$  can be marked invalid for every point for which the data value is greater than one, and then the magnetic field data can be mapped onto the grid with some positions marked as invalid.



**Figure 3 Valid and Invalid positions at  $\psi=0.7$ .**

Figure 4 below shows magnetic field data with values outside  $\psi=1$  marked as invalid. On the left the divergence of the magnetic field is calculated on the flux surface  $\psi=0.7$ , and on the right streamlines are visualized beginning each streamline on the flux surface. This data uses the second order terms of the Taylor expansion. As the picture shows, the streamlines stay within the bounds of  $\psi=1$  as they should, but they do not extend very far. Basically, the streamlines are still wrong, but the visualization is improved because the erroneous data is not visualized. This shows that the main problem in the visualization of the magnetic fields is overall accuracy. The data outside  $\psi=1$  is bad due to the extrapolation, but, even within the  $\psi=1$  boundary, there are still some problems with the magnetic field data. Streamlines should extend all the way around the inside of the torus, and they do not even extend

through one field period (half the torus). This means that the divergence of the magnetic field is not 0 inside the bounds of  $\psi=1$ .



**Figure 4 Improved streamlines with invalidated data at  $\psi=0.7$ .**

#### 4. IMPROVING THE RUNTIME OF THE CODE

The `vmecc_setup.f` code varies in the amount of time to run dependent on the size of the VMEC output file that is being used and on the user specified grid for the final output. Within the code, one of the parts that takes the greatest percentage of runtime is the nearest neighbor routine. In order to put the VMEC gridded data onto the user specified grid, the code runs through every single point on the user specified grid and calculates the distance between that point and all the data values on the VMEC grid. When it finds the shortest distance, it assigns that value from the VMEC grid to the data value for the user specified grid. In order to have greater resolution in the visualization, a user must input more nodes in the  $x$ ,  $y$ , and  $\phi$  directions, resulting in significantly increased runtime for a small increase in resolution.

One way to improve this method is to restrict the search for the closest VMEC grid point. Rather than running through the entire grid for every node, simply search a specified distance on either side of the last found value in the  $r$ ,  $z$ , and  $\phi$  direction. This method was tried, but it has not been fully tested at this



time. Preliminary testing shows a large improvement in run time with virtually no loss of accuracy. Searching the entire grid for an 80x81x32 grid the subroutine transform ran in 652.87 seconds, but restricting the search in one direction, the subroutine transform ran in 206.50 seconds—less than a third of the previous run time.

Another way to improve this method is to make sure that the loops searching the grid are in the most effective order (the inner most loop should be that of the fastest varying variable). An attempt was made to do this using the intrinsic Fortran function "minloc." Minloc takes an array and a specified value and returns the location in the array of the value closest to the specified value. The newest version of vmec\_setup.f uses the minloc function rather than nested loops, but no improvement was seen in the runtime of the code. This could be due to the fact that the Lahey Fortran 95 compiler detects out of order loops and switches them, or it could be that the loops were already in correct order. Since the runtime is effectively the same, the only improvement that using the minloc function gives is improved readability of the code.

## 5. SUGGESTED FURTHER IMPROVEMENTS

The magnetic field visualizations can still be improved. The staggered mesh solution to the visualization problem needs to be implemented and tested. The problem also needs further work with regard to invalidating positions based on data. As some improvement was shown in the visualization of streamlines (Figure 4), this method might be used in addition to the staggered mesh solution for improving accuracy of the data. However, invalidating the positions within the visualization program, OpenDx can be expensive time wise. It is possible to invalidate the positions before the OpenDx data sets are written out, but then one would need to know how to write an OpenDx native format data set with invalidated positions. The writing out of the invalid data file could be an option in the dxdump module. Another way to invalidate data is to use the modules in OpenDx to mark the data outside  $\psi=1$  as bad, and then use the 'export' module in the visual program editor to write out a native dx file with the positions outside  $\psi=1$  marked as invalid. Then the new data files with invalid positions can be visualized in a separate OpenDx program.

The runtime can be further improved by continuing the work of searching a smaller area rather than the entire grid when finding the nearest neighbor. It could also be improved by employing linked list techniques, or perhaps other data structures to gain a better perspective of where the best candidates for a nearest neighbor match are, and thus searching an even smaller range of values.

Eventually, the code should be modularized so that it can be included as an option with the VMEC code. It could be written as an optional interface of functions which could be called at the end of the VMEC code to output native OpenDx files. Additionally, if the code is included in a VMEC release, it would be useful to include several example OpenDx programs which visualize data output from vmec\_setup.f.

## REFERENCES

1. D. Lee et al. 1988. *Optimum Fourier Representations for Stellarator Magnetic Flux Surfaces*, Nucl. Fusion Vol. 28, No. 8, 1351.
2. <http://www.maplesoft.com/products/Maple8/index.shtml>

**APPENDIX A**  
**TEXT of vmec\_setup.f**

```

program vmec_setup
c To compile this code you must have the appropriate VMEC
c module and library files present.
c Compilation script:
c
c xlf -qnlm -o xrf -O4 -I read_wout_mod.mod stell_rf_data.f libstopt.a
c
c
c   use modeparams
c   use read_wout_mod, rmnc_w=>rmnc, zmns_w=>zmns, lmns_w=>lmns,
c   &   xm_w=>xm, xn_w=>xn, phip_w=>phip, mpol_w=>mpol,
c   &   ntor_w=>ntor, nfp_w=>nfp, ns_w=>ns, mnmax_w=>mnmax
c
c
c   implicit none

C-----
C   L o c a l   P a r a m e t e r s
C-----
integer, parameter :: nsd = 101
integer, parameter :: mpold = 9
integer, parameter :: ntord = 9
integer, parameter :: nthetad = 360
integer, parameter :: nzetad = modesphi
integer, parameter :: mpoldd = mpold - 1
integer, parameter :: ntorld = 1 + ntord
integer, parameter :: mnmx = ntorld + mpoldd*(1 + 2*ntord)
integer, parameter :: nmodesxmax = modesx
integer, parameter :: nmodesymax = modesy
integer, parameter :: nmodesphimax = modesphi

C-----
C   L o c a l   V a r i a b l e s
C-----
integer, dimension(mnmx*nsd) :: lmns
integer :: ns, mpol, ntor, mnmax, ljm, js, mn, ipsi,
1  nstart, j, i, k, nfp, ierr
integer nzeta, ntheta
real FHtime1, FHtime2
real, dimension(nsd,mnmx) :: bigr, zeq, bmodeq, bsupu, bsupv
real, dimension(mnmx*nsd) :: rmnc, zmns, bsuptm, bsupzm, bmodm
real, dimension(mnmx) :: xm, xn
real, dimension(nsd) :: hiota, phip
real :: unit, file, status,
2  twopi, zmin, zmax, thet, zeta, bu, bv, buj, bvj, bmj,
3  rmaj, zz, bb, arg, fl, bmin, bmax, r0, rmin, rmax,
4  drdth, dzdth, zetav

real capr(nsd, nthetad, nzetad),
.   capz(nsd, nthetad, nzetad),
.   bmod(nsd, nthetad, nzetad),
.   br(nsd, nthetad, nzetad),
.   bz(nsd, nthetad, nzetad),
.   bzeta(nsd, nthetad, nzetad),
.   drdtheta(nsd, nthetad, nzetad),
.   dzdtheta(nsd, nthetad, nzetad)

real bmodh(nsd), buh(nsd), bvh(nsd)

character arg1*20,warg1*25,bozout*25
integer nargs, numargs, numchars
integer iargc, getarg

integer nmodesx, nmodesy, nmodesphi

```

```

real rwleft, rwright, awally,
. rt, signbz

real qavg0, ymax, zeta0, phistart

C-----

c-----set default values
  phistart = 0.0

  nzeta = nmodesphimax
  ntheta = nthetad

  numargs = iargc()
  numchars = getarg(1,arg1)
  if( numargs.ne.1 )then
    print *, ' MUST ENTER FILENAME ON COMMAND LINE '
    stop
  endif

c
c
  warg1 = arg1
  call read_wout_file(warg1,ierr)

c
c  mnmax = mnmx total number of modes (toroidal and poloidal)
c  Note: This may not be simply mpold*ntord since m = 0 may not
c  have both + and - n's (due to symmetry) whereas m > 0 will
c  have + and - n's.
c  nsdd = nsd = ns = number of radial (flux-coordinate) grid points
c  mpold = number of poloidal modes used
c  ntor = number of toroidal modes used
c  ntor0, mn0, nit, ifsq = not used here
c  xm, xn = poloidal and toroidal mode number pairs
c  rmnc, zmns = Fourier decomposition of major radius and z-elevation on
c  a flux surface (rmnc - cos series, zmns - sin series)
c  bmod = Fourier decomposition of |B| on flux surface
c
c
c  Quantities read from VMEC wout file:
c
  mpol = mpol_w
  ntor = ntor_w
  nfp = nfp_w
  ns = ns_w ! must have this for transform (kate)
  mnmax = mnmx_w
  ljmn = 0
  do js = 1,ns
    hiota(js) = iotas(js)
    phip(js) = phip_w(js)
    do mn=1,mnmax
      ljmn = ljmn + 1
      xm(mn) = xm_w(mn)
      xn(mn) = xn_w(mn)
      rmnc(ljmn) = rmnc_w(mn,js)
      zmns(ljmn) = zmns_w(mn,js)
      lmns(ljmn) = lmns_w(mn,js)
      bmodmn(ljmn) = bmnc(mn,js)
      bsuptmn(ljmn) = bsupumnc(mn,js)
      bsupzmn(ljmn) = bsupvmnc(mn,js)
    enddo
  enddo

```

```

        enddo
    enddo

c
    call read_wout_deallocate
c

    write (*, 780) ns, mpol, ntor, mnmax
    write (*, 781) nsd, mpold, ntord, mnmx
780 format(1x,
1 "ns = ",i4,2x,"mpol = ",i4,2x,"ntor = ",i4,2x,"mnmax = ",i4)
781 format(1x,"nsd = ",i4,2x,"mpold = ",i4,2x,"ntord = ",i4,2x,
1 "mnmx = ",i4)
    write(*,
1'("(numbers in second row must be > than those in first row)")')

c
c     Test to see if dimensions specified in parameter
c     statement were big enough
c
    if (nsd < ns) stop 'NSD<NS'
    if (mpold < mpol) stop 'MPOLD<'
    if (ntord < ntor) stop 'NTORD<'
    if (mnmx > mnmx) stop 'MNMX<'

*
*     NOTE: RMN, ZMN, LMN ARE ON FULL RADIAL GRID POINTS
*           BMN, GMN, BSUPU,VMN ARE AT HALF GRID POINTS
c
c     Read array data from wout file
c
    ljmn = 0
    do js = 1, ns
        do mn = 1, mnmax
            ljmn = ljmn + 1
            bigr(js,mn) = rmnc(ljmn)
            zeq(js,mn) = zmns(ljmn)
            bmodeq(js,mn) = bmodmn(ljmn)
            bsupu(js, mn) = bsuptmn(ljmn)
            bsupv(js, mn) = bsupzmn(ljmn)
        end do
    end do

c
c
c     twopi = 8.*atan(1.)
c
c     Set up psi, theta and phi grids
c
    zmin = 1.E+10
    bmin = 1.E+10
    rmin = 1.E+10
    zmax = -1.E+10
    bmax = -1.E+10
    rmax = -1.E+10

    write(6, *)"number of field periods is: ",nfp

    zeta0 = phistart / nthetad * twopi
    zeta0 = zeta0 / float(nfp)

    call CPU_TIME(FHtimel)

```

```

do i = 1, ntheta
  thet = (i - 1)*twopi/float(ntheta - 1)
  do k = 1, nzeta
    zetav = (k - 1)*twopi/float(nzeta) / float(nfp) + zeta0

*
* -----
* change direction of zeta to match aorsa coordinate system
* -----
c     zeta = - zetav
c     zeta = zetav

c
c     if(i .eq. 1)
do j = 1, ns
  rmaj = 0.
  zz = 0.
  bb = 0.
  bu = 0.
  bv = 0.
  drdth = 0.
  dzdth = 0.

c
c     Add up Fourier harmonics for R (major radius), z(elevation),
c     and |B| at specific psi, theta, phi

  do mn = 1, mnmax
    arg = xm(mn)*thet - xn(mn)*zeta
    rmaj = rmaj + bigr(j,mn)*cos(arg)
    zz = zz + zeq(j,mn)*sin(arg)
    bb = bb + bmodeq(j,mn)*cos(arg)
    bu = bu + bsupu(j,mn)*cos(arg)
    bv = bv + bsupv(j,mn)*cos(arg)
    drdth = drdth - xm(mn) * bigr(j,mn) * sin(arg)
    dzdth = dzdth + xm(mn) * zeq(j,mn) * cos(arg)
  end do
  zmin = min(zmin,zz)
  bmin = min(bmin,bb)
  rmin = min(rmin,rmaj)
  zmax = max(zmax,zz)
  bmax = max(bmax,bb)
  rmax = max(rmax,rmaj)
  r0=(rmax-rmin)*nfp

  capr(j, i, k) = rmaj
  capz(j, i, k) = zz

  bmodh(j) = bb
  buh(j) = bu
  bvh(j) = bv

  drdtheta(j, i, k) = drdth
  dzdtheta(j, i, k) = dzdth

end do

*
* -----
* Interpolate bmod, bu, bv from VMEC half-mesh to the whole mesh:
* -----
do j = 1, ns
  if(j .eq. 1)then
    bmj = bmodh(j+1)
    buj = buh(j+1)

```

```

        bvj = bvh(j+1)
    end if
*
    if(j .gt. 1 .and. j .lt. ns)then
        bmj = (bmodh(j) + bmodh(j+1)) /2.
        buj = (buh(j) + buh(j+1)) /2.
        bvj = (bvh(j) + bvh(j+1)) /2.
    end if
*
    if(j .eq. ns)then
        bmj = bmodh(j)
        buj = buh(j)
        bvj = bvh(j)
    end if

        bmod(j,i,k) = bmj
        br(j, i, k) = buj * drdtheta(j, i, k)
        bz(j, i, k) = buj * dzdtheta(j, i, k)
        bzeta(j, i, k) = bvj * capr(j, i, k)
    end do
end do

end do
    call CPU_TIME(FHtime2)
    write(*,*)"adding Fourier Harmonics took",FHtime2-FHtime1,
        "seconds"
.
write(*,*)"Finished adding Fourier harmonics!"

write(*,*) "r0=",r0,"rmax",rmax,"rmin=",rmin," zmin=",zmin,
.
    " bmin=",bmin
write(*,*) "zmax=",rmax," zmax=",zmax," bmax= ",bmax

! KATE add 10% to the R and Z bounds
rmin = rmin-.1*rmin
rmax = rmax+.1*rmax
zmin = zmin-.1*zmin
zmax = zmax+.1*zmax

write(*,*) capr(1,1,1)
write(*,*) bmod(1,1,1), bzeta(1,1,1)

.
call transform(arg1, ns, ntheta, nzeta, capr, capz,
.   bmod, br, bz, bzeta,
.   drdtheta, dzdtheta, nsd, ntheta, nzeta, nfp,
.   r0, rmin, rmax, zmin, zmax)

write(*,*)"finished calling transform in main"

100 format(i10, 1p8e12.4)

310 format(1p6e12.4)
309 format(10i10)

stop
end

```



```

c
c*****
c
      subroutine transform(dxname, ns, ntheta, nzeta, capr_vmec,
.      capz_vmec,
.      bmod_vmec, bx_vmec, by_vmec, bz_vmec,
.      drdth_vmec, dzdth_vmec, nsd, ntheta_d, nzeta_d, nfp,
.      r0, rmin, rmax, zmin, zmax)

      use modeparams
      use dxdump
      implicit none

      real TransformTime1, TransformTime2, totaltime
      real r0, rmin, rmax, zmin, zmax ! KATE
      character dxname*20
      integer nxmx, nymx, nphimx, ieg, jcol
      integer ier, l
1     ninteg, nd

      integer nrow, ncol, norder

      integer
.     nky1, nky2,
.     nphil, nphi2

      real tmem, tsys, tio, ttotal, time0, time, dummy, second1
      real tmin, gflops, gflobsp, ops

      real psi_lim, dpsi, dtheta, dzeta
      real diffx, diffy, diff, diffmin, tsurf
      integer jvmec, ivmec, jvmec_min, ivmec_min, jk, ik
      integer ns, ntheta, nzeta, nsd, ntheta_d, nzeta_d

      real capa, capx, capy, z, hz, coshz, sinhz, r2,
.     bcapx, bcapy, bcapz, xkh

      integer nmodesmax, mmodesmax, lmodesmax, nrhomax
      real epsl, r, angle, sinang, cosang

      parameter (nmodesmax = modesx)
      parameter (mmodesmax = modesy)
      parameter (lmodesmax = modesphi)

      parameter (nxmx = nmodesmax)
      parameter (nymx = mmodesmax)
      parameter (nphimx = lmodesmax)

      parameter (nrhomax = nmodesmax)

c***  VMEC arrays:

      real capr_vmec(nsd, ntheta_d, nzeta_d),
.      capz_vmec(nsd, ntheta_d, nzeta_d),
.      bmod_vmec(nsd, ntheta_d, nzeta_d)

      real bx_vmec(nsd, ntheta_d, nzeta_d),
.      by_vmec(nsd, ntheta_d, nzeta_d),

```

```

.      bz_vmec(nsd, ntheta, nzeta),
.      drdth_vmec(nsd, ntheta, nzeta),
.      dzdth_vmec(nsd, ntheta, nzeta)

real psi_vmec(nsd),
.      theta_vmec(ntheta)
c      real zeta_vmec(nzeta)

real dzdpsi(nsd, ntheta),
.      dzdth(nsd, ntheta),
.      drdpsi(nsd, ntheta),
.      drdth(nsd, ntheta),
.      dbdpsi(nsd, ntheta),
.      dbdth(nsd, ntheta),
.      dbxdth(nsd, ntheta),
.      dbydth(nsd, ntheta),
.      dbzdth(nsd, ntheta),
.      dbxdpsi(nsd, ntheta),
.      dbydpsi(nsd, ntheta),
.      dbzdpsi(nsd, ntheta)

real ! Partial Derivatives
.      drdpsidpsi(nsd, ntheta),
.      drdthdth(nsd, ntheta),
.      drdthdpsi(nsd, ntheta),
.      dzdpsidpsi(nsd, ntheta),
.      dzdthdth(nsd, ntheta),
.      dzdthdpsi(nsd, ntheta),
.      dbxdpsidpsi(nsd, ntheta),
.      dbxdthdth(nsd, ntheta),
.      dbxdthdpsi(nsd, ntheta),
.      dbydpsidpsi(nsd, ntheta),
.      dbydthdth(nsd, ntheta),
.      dbydthdpsi(nsd, ntheta),
.      dbzdpsidpsi(nsd, ntheta),
.      dbzdthdth(nsd, ntheta),
.      dbzdthdpsi(nsd, ntheta)

real dzdpsik, dzdthk, drdpsik, drdthk, xjacob

real ! Partial Derivatives computed
.      drdpsidpsik,
.      drdthdthk,
.      drdthdpsik,
.      dzdpsidpsik,
.      dzdthdthk,
.      dzdthdpsik,
.      dbxdpsidpsik,
.      dbxdthdthk,
.      dbxdthdpsik,
.      dbydpsidpsik,
.      dbydthdthk,
.      dbydthdpsik,
.      dbzdpsidpsik,
.      dbzdthdthk,
.      dbzdthdpsik

real dbdpsik, dbdthk,
.      dbxdpsik, dbxdthk,
.      dbydpsik, dbydthk,
.      dbzdpsik, dbzdthk

```

```

real psik, thetak, theprm, psi1, bmodk, bxk, byk, bzk,
.   psi2, theta2, secondOeq1, secondOeq2
real xk(nxmx), yk(nymx)
real bmod_vmeck

c*** 3-D arrays:
real psi(nxmx, nymx, nphimx),
.   psi_dx(nxmx, nphimx, nymx),
.   FULL_psi_dx(nxmx, nphimx*nfp, nymx),
.   bmod_dx(nxmx, nphimx, nymx),
.   FULL_bmod_dx(nxmx, nphimx*nfp, nymx),
.   rho(nxmx, nymx, nphimx),
.   thetap(nxmx, nymx, nphimx)

real bxn(nxmx, nymx, nphimx),
.   bxn_dx(nxmx, nphimx, nymx),
.   FULL_bxn_dx(nxmx, nphimx*nfp, nymx),
.   byn(nxmx, nymx, nphimx),
.   byn_dx(nxmx, nphimx, nymx),
.   FULL_byn_dx(nxmx, nphimx*nfp, nymx),
.   bzn(nxmx, nymx, nphimx),
.   bzn_dx(nxmx, nphimx, nymx),
.   FULL_bzn_dx(nxmx, nphimx*nfp, nymx),
.   bmod(nxmx, nymx, nphimx)

character*1 trans

complex b, zi

complex arg, xil(0:100), xilp(0:100)
complex bl(100)

real
.   rhomax, rhoant,
.   dthetant, dphiant

integer nmodesx, nmodesy, nmodesphi,
.   nnodex, nnodey,
.   nnodephi, i, j, k, l,
.   jequat, iflag, liw, lw, nrhs, icenter, np, nfp

integer noderho

integer n, m, nphi

real
.   btor,
.   rt, awally, awallz, signbz,
.   awright,
.   rhoplasm,
.   qavg0, ymax, phimax, psiwall,
.   rhonorm, xiota0,
.   rwleft, rwright, xwleft, xwright, psiright
real dxdrho(nxmx, nymx), dzdrho(nxmx, nymx)

real
.   q, teedge, xlnlam,
.   xmax, qe,i3, pi,
.   costh, sinth, radius, rnx, rny, rnphi

```

```

real capr(nxxmx),
.  xprime(nxxmx), x(nxxmx), dx

real rhon(nrhomax), wdotilavg(nrhomax), wdoti2avg(nrhomax),
.  wdoteavg(nrhomax), drho
real xnavg(nrhomax), qhat

real theta(nxxmx, nymx), theta0(nxxmx, nymx),
.  bx, by, bz,
.  bx_fred, by_fred, bz_fred,
.  bzeta(nxxmx, nymx),
.  dxdth(nxxmx, nymx), xntau(nxxmx, nymx),
.  xiota(nxxmx, nymx), qsafety(nxxmx, nymx)

real bphi, bth, br

real rhomt0t(nxxmx, nymx), rhome, rhomil, rhomi2, rhomi3
real xnupi(nxxmx, nymx), prod

real dbxdx, dbydx, dbzdx,
.  dbxdy, dbydy, dbzdy,
.  dbxdphi, dbydphi, dbzdphi

real wdote(nxxmx, nymx), wdotil(nxxmx, nymx),
.  wdoti2(nxxmx, nymx), wdoti3(nxxmx, nymx), wdot(nxxmx, nymx)
real fye, fyi1, fyi2, fyi3, fy

real fype(nxxmx, nymx), fypil(nxxmx, nymx),
.  fypi2(nxxmx, nymx), fypi3(nxxmx, nymx), fyp(nxxmx, nymx)

real denom

real p, pil, pi2, pit, pi3, pe

real
.  yprime(nymx), y(nymx), dy

real phiprimec(nphimx), phicourse(nphimx),
.  phiprime(nphimx), phi(nphimx), phi0(nphimx),
.  phitot(nphimx*nfp),
.  dphi, dphic, phistart, phi_nfp(nphimx)

integer RelErrorCount, BxErrorCount, ByErrorCount, BzErrorCount
integer vecindex(2)

write(*,*)"Entered subroutine transform." ! KATE
totaltime=0.0
c--set default values of input data:

nmodesx = nmodesmax
nmodesy = mmodesmax
nmodesphi = lmodesmax
phistart = 0.0
rt = r0
rwleft = rmin !.70
rwright = rmax !2.5
ymax=0 ! default--later this is set to 2*awally
awally = max(abs(zmin),abs(zmax))
signbz = 1.0000E+00
qavg0 = 1.0

write(*,*)"rt=",rt
write(*,*)"rwleft=",rwleft," rwright=",rwright

```

```

write(*,*)"awally=",awally

nrhs = 1

c-----nmodesx=number of modes used in the x direction
c-----nmodesy=number of modes used in the y direction
c-----nmodesphi=number of modes used in the phi direction
c-----nnodecx = number of radial mesh points used for wdot calculation
c-----nnodecy = number of vertical mesh points used for wdot calculation
c-----phistart = starting position in phi for VMEC magnetic field
c-----rt= major radius of torus
c-----rwleft = major radius of the left conducting wall
c-----rwright = major radius of the right conducting wall
c-----awally = vertical location of the conducting wall
c-----ymax = radius in vertical (y) direction- in default it is set to awallx

if(abs(signbz).lt.1.0e-05)signbz = +1.0

c-----qavg0 is the rotational transform on axis

nnodex = nmodesx
nnoderho = nnodex

nky2 = nmodesy / 2
nky1 = - nmodesy / 2 + 1
nnodey = nmodesy

nphi2 = nmodesphi / 2
nphi1 = - nmodesphi / 2 + 1
nnodephi = nmodesphi

jequat = nnodey / 2
icenter = nnodex / 2
if (qavg0 .ne. 0.0) xiota0 = 1./qavg0
q = 1.6e-19
teedge = 400.0
qhat = qavg0
teedge = teedge * q
qe = -q
zi = cmplx(0.0,1.0)
pi = 3.141592654
xlnlam = 20.0
xmax = rwright - rwleft
awright = rwright - rt
ymax = 2.0 * awally

xwleft = rwleft - rt
xwright = rwright - rt

write(*,*)"xwleft=",xwleft," xwright",xwright
write(*,*)"About to define x mesh..." ! KATE
write(*,*)"nnodex=",nnodex,"nnodey=",nnodey ! KATE

*
* -----
* Define x mesh: x(i), xprime(i), capr(i)
* -----
c-- xprime: 0 to xmax
c-- x(i) : -xmax / 2.0 to xmax / 2.0
dx = xmax / nnodex
do i = 1, nnodex
xprime(i) = (i-1) * dx + dx / 2.0
c-- Note: the code gives slightly smoother results with dx/2.0 added
x(i) = xprime(i) + xwleft

```

```

    capr(i) = rt + x(i)
end do

write(*,*)"About to define y mesh..." ! KATE
write(*,*)"nnodex=",nnodex,"nnodey=",nnodey ! KATE

*
* -----
* Define y mesh: y(j), yprime(j)
* -----
c-- yprime: 0 to ymax
c-- y(j) : -ymax / 2.0 to ymax / 2.0
    dy = ymax / nnodey
    do j = 1, nnodey
        yprime(j) = (j-1) * dy + dy / 2.0
c-- Note: the code gives slightly smoother results with dy/2.0 added
        y(j) = yprime(j) - awally
    end do

c    np = 1
c    np = nfp
    phimax = 2.0 * pi / nfp

write(*,*)"About to define phi mesh..." ! KATE
write(*,*)"nnodephi=", nnodephi ! KATE

*
* -----
* Define phi mesh: phi(k), phiprime(k)
* -----
c-- phiprime: 0 to phimax
c-- phi(k) : -phimax / 2.0 to phimax / 2.0
c-- phi0(k): 0 to phimax / 2.0 and 0 to -phimax / 2.0
    dphi = phimax / nnodephi
    do k = 1, nnodephi
        phiprime(k) = (k-1) * dphi
c    1 + dphi / 2.0
c-- Note: the code gives worse results with dz/2.0 added
        phi(k) = phiprime(k) - phimax / 2.0

        if(phi(k) .le. 0.0) phi0(k) = phi(k) + phimax / 2.0
        if(phi(k) .gt. 0.0) phi0(k) = phi(k) - phimax / 2.0

        phi_nfp(k) = phiprime(k) * nfp
    end do

do i = 1, nnodex
    do j = 1, nnodey
        if(x(i) .ne. 0.0 .or. y(j) .ne. 0.0)then
            theta0(i,j) = atan2(y(j), x(i))
            if(theta0(i,j) .ge. 0.0) theta(i,j) = theta0(i,j)
            if(theta0(i,j) .lt. 0.0) theta(i,j) = theta0(i,j)+ 2.* pi
        end if
    end do
end do

write(*,*)"About to define rho mesh..." ! KATE
c-- Define rho mesh: rhon(n)
c-- rhon: 0 to rhomax
    rhomax = 1.0
    drho = rhomax / (nnoderho - 1)
    do n = 1, nnoderho
        rhon(n) = (n-1) * drho
    end do

```

```

end do

*
* -----
* Stellarator flux surfaces from VMEC
* -----

*
* -----
* VMEC mesh
* -----
psi_lim = float(ns)

dpsi = psi_lim / float(ns - 1)
dtheta = 2.0 * pi / float(ntheta - 1)
c dzeta = 2.0 * pi / float(nzeta) / float(nfp)

do j = 1, ns
  psi_vmec(j) = (j-1) * dpsi
end do

do i = 1, ntheta
  theta_vmec(i) = (i-1) * dtheta
end do

RelErrorCount=0
BxErrorCount=0
ByErrorCount=0
BzErrorCount=0
do k = 1, nnodephi

*
* -----
* Calculate derivatives from VMEC
* -----
do jvmec = 1, ns
  do ivmec = 1, ntheta
    jk = jvmec
    ik = ivmec

    call deriv_psi(capz_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dpsi, dzdpsi(jk, ik))
    call deriv_psi(capr_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dpsi, drdpsi(jk, ik))
    call deriv_psi(bmod_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dpsi, dbdpsi(jk, ik))
    call deriv_psi(bx_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dpsi, dbxdpsi(jk, ik))
    call deriv_psi(by_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dpsi, dbydpsi(jk, ik))
    call deriv_psi(bz_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dpsi, dbzdpsi(jk, ik))

    call deriv_theta(bmod_vmec, nsd, ntheta, nzeta,
      jk, ik, k, ns, ntheta, dtheta, dbdth(jk, ik))

```

```

        call deriv_theta(bx_vmec, nsd, nthetad, nzetad,
            jk, ik, k, ns, ntheta, dtheta, dbxdth(jk, ik))

        call deriv_theta(by_vmec, nsd, nthetad, nzetad,
            jk, ik, k, ns, ntheta, dtheta, dbydth(jk, ik))

        call deriv_theta(bz_vmec, nsd, nthetad, nzetad,
            jk, ik, k, ns, ntheta, dtheta, dbzdth(jk, ik))

        dzdth(jk, ik) = dzdth_vmec(jk, ik, k)
        drdth(jk, ik) = drdth_vmec(jk, ik, k)
    end do
end do
***** Partial derivatives (KATE)
! Partial derivatives (capr)
do jvmec = 1, ns
    do ivmec = 1, ntheta
        jk = jvmec
        ik = ivmec
        call deriv_psi2(drdpsi, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, drdpsidpsi(jk, ik))
        call deriv_psi2(drdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, drdthdpsi(jk, ik))
        call deriv_theta2(drdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, drdthdth(jk, ik))

        ! Partial derivatives (capz)
        call deriv_psi2(dzdpsi, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dzdpsidpsi(jk, ik))
        call deriv_psi2(dzdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dzdthdpsi(jk, ik))
        call deriv_theta2(dzdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dzdthdth(jk, ik))

        ! Partial derivatives (bx)
        call deriv_psi2(dbxdpsi, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbxdpsidpsi(jk, ik))
        call deriv_psi2(dbxdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbxdthdpsi(jk, ik))
        call deriv_theta2(dbxdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbxdthdth(jk, ik))

        ! Partial derivatives (by)
        call deriv_psi2(dbydpsi, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbydpsidpsi(jk, ik))
        call deriv_psi2(dbydth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbydthdpsi(jk, ik))
        call deriv_theta2(dbydth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbydthdth(jk, ik))

        ! Partial derivatives (bz)
        call deriv_psi2(dbzdpsi, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbzdpsidpsi(jk, ik))
        call deriv_psi2(dbzdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbzdthdpsi(jk, ik))
        call deriv_theta2(dbzdth, nsd, nthetad,
            jk, ik, ns, ntheta, dpsi, dbzdthdth(jk, ik))

    end do
end do

*
* -----
* Create a table of values from VMEC
* -----
*

```





```

drdpsidpsik = drdpsidpsi(jk,ik)
drdthdthk = drdthdth(jk,ik)
drdthdpsik = drdthdpsi(jk,ik)

dzdpsidpsik = dzdpsidpsi(jk,ik)
dzdthdthk = dzdthdth(jk,ik)
dzdthdpsik = dzdthdpsi(jk,ik)

dbxdpsidpsik = dbxdpsidpsi(jk,ik)
dbxdthdthk = dbxdthdth(jk,ik)
dbxdthdpsik = dbxdthdpsi(jk,ik)

dbydpsidpsik = dbydpsidpsi(jk,ik)
dbydthdthk = dbydthdth(jk,ik)
dbydthdpsik = dbydthdpsi(jk,ik)

dbzdpsidpsik = dbzdpsidpsi(jk,ik)
dbzdthdthk = dbzdthdth(jk,ik)
dbzdthdpsik = dbzdthdpsi(jk,ik)
! end Partial

xjacob = dzdpsik * drdthk - dzdthk * drdpsik

if(abs(xjacob) <= .0001*(abs(dzdpsik * drdthk) +
.   abs(dzdthk * drdpsik))) then
.   psi1 = psik
.   psi2 = psik
.   theprm = thetak
.   theta2 = thetak

else
.   psi1 = psik + 1.0 / xjacob
.   * (drdthk * (y(j) - yk(j))
.   - dzdthk * (capr(i) - xk(i)))

.   theprm = thetak + 1. / xjacob
.   * (dzdpsik * (capr(i) - xk(i))
.   - drdpsik * (y(j) - yk(j)))

.   psi1= psi1-psik
.   theprm= theprm-thetak

.   secondOeq1=.5*drdpsidpsik*psi1*psi1 +
.   drdthdpsik*psi1*theprm +
.   .5*drdthdthk*theprm*theprm
.   secondOeq2=.5*dzdpsidpsik*psi1*psi1 +
.   dzdthdpsik*psi1*theprm +
.   .5*dzdthdthk*theprm*theprm

.   psi2=psik + (1.0 / xjacob)
.   * (drdthk * (y(j) - yk(j)-secondOeq2)
.   - dzdthk * (capr(i) - xk(i)-secondOeq1))

.   theta2=thetak + (1.0 / xjacob)
.   * (dzdpsik * (capr(i) - xk(i)-secondOeq1)
.   - drdpsik * (y(j) - yk(j)-secondOeq2))

.   psi1=psi1+psik
.   theprm=theprm+thetak

end if

if(abs(psi1-psi2)/abs(psi1) > .5) then
.   psi2=psi1
.   if(psi1<1) RelErrorCount=RelErrorCount+1

```

```

endif
if(abs(theprm-theta2)/abs(theprm) > .5) then
    theta2=theprm
endif

psil=psi2
theprm=theta2
!psil=psil
!theprm=theprm

bmod_vmeck = bmodk + dbdthk * (theprm - thetak)
                + dbdpsik * (psil - psik)

bx_fred = bxk + dbxdthk * (theprm - thetak)
            + dbxdpsik * (psil - psik)
bx = bxk + dbxdthk * (theprm - thetak)
            + dbxdpsik * (psil - psik)
            + dbxdpsidpsik*(psil-psik)*(psil-psik)
            + dbxdthdpsik*(psil-psik)*(theprm-thetak)
            + dbxdthdthk*(theprm-thetak)*(theprm-thetak)

by_fred = byk + dbydthk * (theprm - thetak)
            + dbydpsik * (psil - psik)
by = byk + dbydthk * (theprm - thetak)
            + dbydpsik * (psil - psik)
            + dbydpsidpsik*(psil-psik)*(psil-psik)
            + dbydthdpsik*(psil-psik)*(theprm-thetak)
            + dbydthdthk*(theprm-thetak)*(theprm-thetak)

bz_fred = bzk + dbzdthk * (theprm - thetak)
            + dbzdpsik * (psil - psik)
bz = bzk + dbzdthk * (theprm - thetak)
            + dbzdpsik * (psil - psik)
            + dbzdpsidpsik*(psil-psik)*(psil-psik)
            + dbzdthdpsik*(psil-psik)*(theprm-thetak)
            + dbzdthdthk*(theprm-thetak)*(theprm-thetak)

if(abs(bx_fred-bx)/abs(bx_fred) > .2 .AND. psil<1) then
    BxErrorCount=BxErrorCount+1
endif
if(abs(by_fred-by)/abs(by_fred) > .2 .AND. psil<1) then
    ByErrorCount=ByErrorCount+1
endif
if(abs(bz_fred-bz)/abs(bz_fred) > .2 .AND. psil<1) then
    BzErrorCount=BzErrorCount+1
endif

bmod(i, j, k) = sqrt(bx**2 + by**2 + bz**2)

!if(psil<=1) then
!    write(*,*) "bmod(",i,",",j,",",k,")=",bmod(i,j,k)
!endif

bxn(i,j,k) = bx / bmod(i,j,k)
byn(i,j,k) = by / bmod(i,j,k)
bzn(i,j,k) = bz / bmod(i,j,k)

```

```

        psi(i,j,k) = psil / psi_lim
        thetap(i,j,k) = theprm
        rho(i,j,k) = sqrt(psi(i,j,k))
    end do
end do

write(*,*)"Finished creating table of values." ! KATE
write(*,*)"RelErrorCount (>.5) = ", RelErrorCount
write(*,*)"BxErrorCount (>.2) = ", BxErrorCount
write(*,*)"ByErrorCount (>.2) = ", ByErrorCount
write(*,*)"BzErrorCount (>.2) = ", BzErrorCount

rt = capr_vmec(1,1,1)

c----switch the phi and y coordinates and use dx polar
do i = 1, nnodex
    do j = 1, nnodey
        do k = 1, nnodephi
            psi_dx(i,k,j) = psi(i,j,k)
            bmod_dx(i,k,j) = bmod(i,j,k)
            bxn_dx(i,k,j) = bxn(i,j,k)
            byn_dx(i,k,j) = byn(i,j,k)
            bzn_dx(i,k,j) = bzn(i,j,k)
        enddo
    enddo
enddo

do l = 0, nfp-1
    phitot(1+l*nnodephi:nnodephi+1*nnodephi)=
        +phi(:)+l*(2*pi/(nfp))
enddo

do l = 0, nfp-1
    FULL_psi_dx(:,1+l*nnodephi:nnodephi+1*nnodephi,:)=
        psi_dx(:, :, :)
    FULL_bmod_dx(:,1+l*nnodephi:nnodephi+1*nnodephi,:)=
        bmod_dx(:, :, :)
    FULL_bxn_dx(:,1+l*nnodephi:nnodephi+1*nnodephi,:)=
        bxn_dx(:, :, :)
    FULL_byn_dx(:,1+l*nnodephi:nnodephi+1*nnodephi,:)=
        byn_dx(:, :, :)
    FULL_bzn_dx(:,1+l*nnodephi:nnodephi+1*nnodephi,:)=
        bzn_dx(:, :, :)
enddo

write(*,*)"Switched coordinates and made FULL mesh." ! KATE

dxname=trim(dxname)
dxname=dxname(5:(len(dxname)-4))
c----dx calls for 3 dimensional output
write(*,*)"Calling dxPolar for PSI..." ! KATE
call dxPol_scalar_dp
    (psi_dx(1:nnodex,1:(nnodephi),1:nnodey),
    capr(1:nnodex)
    ,phitot(1:(nnodephi)), y(1:nnodey),
    'PSI2nd'//dxname)
call dxPol_scalar_dp
    (FULL_psi_dx(1:nnodex,1:(nnodephi*nfp),1:nnodey),
    capr(1:nnodex)

```

```

.          ,phitot(1:(nnodephi*nfp)), y(1:nnodey),
.          'FULL_PSI2nd'//dxname)
write(*,*)"Calling dxPolar for BMOD..." ! KATE
call dxPol_scalar_dp
.          (bmod_dx(1:nnodex,1:(nnodephi),1:nnodey),
.          capr(1:nnodex)
.          ,phitot(1:(nnodephi)), y(1:nnodey),
.          'BMOD2nd'//dxname)
call dxPol_vector_dp
.          (bxn_dx(1:nnodex,1:(nnodephi),1:nnodey),
.          -bzn_dx(1:nnodex,1:(nnodephi),1:nnodey),
.          byn_dx(1:nnodex,1:(nnodephi),1:nnodey),
.          capr(1:nnodex)
.          ,phitot(1:(nnodephi)), y(1:nnodey),
.          'VEC2nd'//dxname)
call dxPol_vector_dp
.          (FULL_bxn_dx(1:nnodex,1:(nnodephi*nfp),1:nnodey),
.          -FULL_bzn_dx(1:nnodex,1:(nnodephi*nfp),1:nnodey),
.          FULL_byn_dx(1:nnodex,1:(nnodephi*nfp),1:nnodey),
.          capr(1:nnodex)
.          ,phitot(1:(nnodephi*nfp)), y(1:nnodey),
.          'FULL_VEC2nd'//dxname)

310 format(1p6e12.4)
309 format(10i10)

* -----
* stop parallel environment
* -----
c   call blacs_gridexit( icontxt )
c   call blacs_exit(0)

write(*,*)"total time for search=",totaltime
stop
end subroutine transform

c
c*****
c
subroutine deriv_psi(f, jd, id, kd, j, i, k, jmax, imax,
.                  dpsi, dfdpsi)

implicit none

integer jd, id, kd, j, i, k, jmax, imax
real f(jd, id, kd), dpsi, dfdpsi

if(j .ne. 1 .and. j .ne. jmax)
.   dfdpsi = (f(j+1, i, k) - f(j-1, i, k)) / (2.0 * dpsi)
if(j .eq. 1)
.   dfdpsi = (f(j+1, i, k) - f(j, i, k)) / dpsi
if(j .eq. jmax)
.   dfdpsi = (f(j, i, k) - f(j-1, i, k)) / dpsi

return
end

c
c*****

```

c

```
subroutine deriv_psi2(f, jd, id, j, i, jmax, imax,
.                   dpsi, dfdpsi)

implicit none

integer jd, id, j, i, jmax, imax
real f(jd, id), dpsi, dfdpsi

if(j .ne. 1 .and. j .ne. jmax)
.   dfdpsi = (f(j+1, i) - f(j-1, i)) / (2.0 * dpsi)
if(j .eq. 1)
.   dfdpsi = (f(j+1, i) - f(j, i)) / dpsi
if(j .eq. jmax)
.   dfdpsi = (f(j, i) - f(j-1, i)) / dpsi

return
end
```

c

C\*\*\*\*\*

c

```
subroutine deriv_theta(f, jd, id, kd, j, i, k, jmax, imax,
.                   dtheta, dfdth)

implicit none

integer jd, id, kd, j, i, k, jmax, imax, ipl, iml
real f(jd, id, kd), dtheta, dfdth

ipl = i + 1
iml = i - 1
if(i .eq. 1) iml = imax - 1
if(i .eq. imax) ipl = 2
dfdth = (f(j, ipl, k) - f(j, iml, k)) / (2.0 * dtheta)

return
end
```

c

C\*\*\*\*\*

c

```
subroutine deriv_theta2(f, jd, id, j, i, jmax, imax,
.                   dtheta, dfdth)

implicit none

integer jd, id, j, i, jmax, imax, ipl, iml
real f(jd, id), dtheta, dfdth

ipl = i + 1
iml = i - 1
if(i .eq. 1) iml = imax - 1
if(i .eq. imax) ipl = 2
dfdth = (f(j, ipl) - f(j, iml)) / (2.0 * dtheta)

return
end
```

c

C\*\*\*\*\*