Marylie Documentation Supplement: Beamline Design

C. Thomas Mottershead
phone: (805) 264-0583
email: ctmotters@gmail.com
July 8, 2023

Contents

1 Marylie Basics

Marylie is a large beam dynamics and analysis code from the University of Maryland, developed by Prof. Alex Dragt and co-workers since about 1984. It uses a Lie Algebraic formulation of the transfer map, hence the name (MARYland LIE algebra). The basic theory is outlined in the LieOpticSum note. Los Alamos has been involved from almost the beginning, especially the use of the code for design purposes. This note is meant to be a brief guide for new users of Marylie for beamline design. The current version of Marylie has some 42,000 lines of Fortran, in about 500

subroutines. It has many capabilities in beam dynamics simulation and analysis, as detailed in the 750 page user manual.

1.1 The Marylie Input File

Marylie is driven by a master input file divided into 8 segments, each marked with a # sign:

`#comment`: Optional text describing the purpose of the present run.
`#beam` : Kinematic data for this run.
`#menu` : The pool of user defined names for instances of typecode commands with their associated parameters.
`#xdata` : Supplemental text data, alternative to external files. (new feature)
`#lines` : Beam-lines defined by (possibly nested) sequences of user-named elements.
`#lumps` : Instruction sequences to construct and save a map.
`#loops` : Sequential lists of all elements in the specified lines.
`#labor` : The program to be executed, written in the language of usernames defined in the other segments - #menu, #lines, #loops or #lumps.

1.2 TypeCodes and Keys

There are some 238 commands or typecodes that cause some action when invoked, typically calling a subroutine to do some calculation. The user gives a unique name to each instance of a typecode command to be used, including its associated parameters. Sequences of these usernames in the `#labor` segment then serve as a program defining the calculations to be done. Unused (and unknown) typecode commands can generally be ignored. They are in five basic catagories:

1. Beamline elements such as drifts, bends, and magnetic multipoles. The parameters specify their length, strength, fringe fields, etc. These compute the transfer map of the element, and concatenate it onto the accumulating working map.

2. Codes that make use the current map, analysing it, tracking particles through it, printing it, computing inverses, normal forms, tunes, chromaticities, etc.

3. Codes for adjusting the beamline element parameters to give the map desired characteristics. These design procedures are the main subject of this note.

4. User routines that do whatever the user writes them to do, providing great flexibility.

5. Random elements, user routines, and parameter sets (used for error studies).

Appendix A has a complete list of the typecode names, with the number of parameters required for each. The items computed from the typecode commands - map elements and measures, user routine outputs, etc. - are saved for future use, usually in a common block. There are 49 "keys" defined for reference to these computed values. Keys for array elements have associated indices. Appendix B has the full list.

2 Defining The Design Problem

The design process consists of adjusting the parameters of a beamline - mainly the length and strength of the elements - to give it desired properties, e.g. $R(1,2) = 0.0$ for x-plane focus. Any computable characteristic of the beamline - elements of the transfer map, secondary quantities derived from it, or something computed by a user supplied subroutine - may be selected as a design aim. The desired beamline properties are the target values (*e.g.* 0.0) for the chosen aims (*e.g.* $R(1,2)$).

2.1 Design Aims

The typecode `'aim'` reads these aim and target definitions as text of the form:

```
aim = target
```

Any number of aim and target definitions may be put on the same line, with blanks or commas as delimiters. Input is terminated by a # sign. This text (not case sensitive) may be read from an external file, or from the new `#xdata` segment of the main input file.

2.2 Internal Text Data

The `#xdata` segment is a new convenience feature allowing the design process to be fully defined in the master input file. The commands `aim`, `vary`, and `sq` read a few lines of text defining the design aims and variables. The source is specified by their second parameter: $p(2) = \pm N$. If $p(2) = +N$, the source is an external file on logical unit N. If $p(2) = -N$, the text is copied instead from block N of the `#xdata` segment, which has the form:

```
#xdata
    ixd>:   N1
       lines of supplemental text (e.g.  R(1,2)=0.0) for virtual unit N1
    ixd>:   N2
       more arbitrary text for virtual unit N2
```

The keyword " `ixd>:`" followed by a numeral N is the delimiter marking the start of virtual file N. The `#xdata` block will hold up to 512 text lines, about 8 pages. The text lines may be up to 80 characters long. The virtual unit numbers (N1, N2, ...) are arbitrary, and do not need to be in sequence. Other Marylie commands may use this feature in the future.

2.3 Design Variables

The parameters of the defined beamline must then be adjusted to give the specified design aims their target values. Typical choices are the length or strength of selected beamline elements. The typecode `'vary'` selects a parameter as a design variable by reading the text

```
username npar
```

where username is the name given in #menu to the selected element, and `npar` = parameter number (1 to 6) to be varied. The text is not case sensitive, and may be read from either an external file or the new `#xdata` segment of the main input file. Any number of variable definitions may be on the same line, delimited by blanks or commas. Input is terminated by a # sign.

If the first parameter of `vary` is -1, a dependent variable may be attached to one of the primary independent variables, by entering on separate lines after the first # sign:

```
depvar   npar
 idv    slope
```

Here the first line is the username and parameter of the new dependent variable, and the second line specifies the definition sequence number (`idv`) of the independent variable to which it is slaved with the derivative `slope`. Since the number of dependent variables is otherwise unlimited, their input must always be terminated with another # sign (or end-of-file).

For example, a Russian quadruplet made of 4 quadrupoles with usernames QA, QB, QC, QD has the constraint that QA and QD have the same gradient with opposite signs, as do QB and QC. To maintain this symmetry while varying the two independent gradients, the `vary` input would be:

```
QA  2     QB  2   #
QD  2
 1   -1.0
QC  2
 2   -1.0
  #
```

The 2 after the names selects the gradient (parameter 2) for variation. Setting `slope` = -1.0 insures that if QA and QD have equal and opposite gradients to begin with, that symmetry is maintained throughout the variation. QC is slaved to QB in the same manner.

2.4 Labor Iterations

The `#labor` segment is a program specifing the calulations to be done, written in the language of usernames defined in the `#menu` segment, with subroutines defined in the `#lines` segment. Up to two (possibly nested) iterative processes (do loops) may be defined (eg. `fit,scan`). The typecodes `(bip,tip)` mark the begining and end of the "inner" process, and `(bop,top)` that of the "outer". The begin points have one parameter: `ntimes` = the maximum iterations allowed. The number of times a loop is actually executed depends on what it contains - e.g typecode `fit` quits when it either converges or fails. The applicable loop number is the fifth parameter of both the `aim` and `vary` typecodes that can define independent lists of aim and variable selections for each process. A third list may be defined for printouts (`wsq`).

3 The Design Process

A point design is a location in variable space for which the computed design aims take on their target values.

3.1 Fitting the Aims

Marylie uses an adaptive multidimensional iterative interpolation algorithm (AMDII) to adjust the selected variables to fit the aims. The k^{th} computed aim is a function $a_k(\vec{x})$ on the space $\vec{x}$ of selected

variables. A target value t_k specifies a contour level hypersurface $a_k(\vec{x}) = t_k$ of this function. A second aim-target choice defines another such hypersurface. For a problem defined by N choices of aims, targets, and variables, we seek a common root $\vec{x}_r$ that zeros all the error components $f_k(\vec{x}_r) \equiv a_k(\vec{x}_r) - t_k = 0$ for $k = 1, N$. This root lies in the intersection of all the contour level hypersurfaces. It may not be unique if some of the hypersurfaces are coincident. Or it may not exist if some of the hypersurfaces do not intersect.

The process begins with user specified feeler steps along each of the axes to compute the initial numerical derivatives. The hypersurface defined by each target equation is approximated by a hyperplane fitted to the N+1 best points so far. The process is most easily visualized in two dimesions: Each approximate plane intersects the zero plane in a line, and the crossing point of these lines is taken as the next estimate of the root. The exact functions are then recalculated at the estimated root, the new point replaces the worst previous point, and the root estimate is repeated. If all goes well, these simple multidmensional interpolation steps can rapidly converge to machine precision when they bottom out on roundoff.

The typecode `fit` does one iteration of this process. Calculation of the relevant aims must preceed it in the `labor` sequence. It can then update the selected variables. The first parameter is ±KFIT; the sign determines which loop selections to use - positive for inner, negative for outer. The other parameters specify feeler step size, error tolerance, print output, etc. See the Command Appendix for more detail.

If the new point is not the best yet, and KFIT > 1, an interim target point is selected halfway between the real target and the best point yet, and the process restarted. This adaptive targeting may be repeated up to KFIT times to allow up to KFIT-1 halvings of the reach toward the ultimate target point, rather than simply giving up on the first error increase. If the smaller step succeeds, a bigger step towards the real target is attemped. This moving target capability makes the search for a solution much more robust.

3.2 Scanning the Fit

Once an initial point design is found, other parameters of the beamline can be scanned through a specified range to explore the response of this design point. Sometimes the design variables themselves may be scanned without fitting to see if the aims *ever* hit the targets. This can be done manually, or by using the Marylie scan feature that allows any selected parameter to be incremented, and any desired inner computation repeated any number of times in an outer loop. Such parameter surveys provide clues to the range of the possible, and aid in the search for the best design, which could also need to satisfy secondary desiderata such as total length or maximum quadrupole gradients. The response of any such computable quantity to the scan parameter can be tabulated in the scan loop. An example of a Marylie file with this "scanning-the-fit" procedure is given below.

The constraint hypersurfaces generally move with the scan parameter, and may no longer intersect beyond some limit. The final inner loop fit aims should be printed for each step of the outer scan loop to verify convergence of the fit process. In this case, any convergence failures mark the valid boundaries of the scan range.

Convergence failures in general provide clues to the nature of the parameter space being studied. For example, if KFIT = 10 in the fit process, we are allowing up to 10 retargeting cuts. If all 10 cuts are tried, we are seeking a solution only 1/1024 the way from the current point to the original target. If even that fails, it serves as a warning that something is seriously wrong with the selection

of variables and aims. Maybe there is no solution at all to the question being asked, or there is a chasm of no solution blocking the path from the initial guess to the final target. Or maybe the chosen fit variables have no effect on the chosen aims. The game is to read the convergence failures to feel out the structure of parameter space, which could require a lot of trial and error.

3.3 Standard File Names

One step of the design cycle may use several input and output files. These are read from or written to the fortran unit numbers specified in the typecode parameters. The choice of unit numbers and file names is actually arbitrary, but it is convenient to standardize. I give each step in the design cycle a unique case name, e.g. `"case"`, with the Marylie input file named `"case.mip"` . The other associated input and output files are given the same name, with different suffixes, which can be set by a unix script. My standard file assignments were

```
-----------------------------------INPUT FILES ----------------------------
case.sca  fort.3   SCan Aims for wsq  (outer loop).    Or use ixd>:  3 in #xdata
case.ssv  fort.4   Select Scan Variables (outer loop). Or use ixd>:  4 in #xdata
case.aim  fort.7   Inner Loop Aims for fit.            Or use ixd>:  7 in #xdata
case.var  fort.8   Inner Loop Variables for fit.       Or use ixd>:  8 in #xdata
case.sq   fort.9   Select Quantities for wsq (loop 3)  Or use ixd>:  9 in #xdata
case.sv   fort.10  Select Variables  for wsq (loop 3)  Or use ixd>: 10 in #xdata
case.mip  fort.11  Marylie main InPut file
-----------------------------------OUTPUT FILES ---------------------------
case.out  fort.12  Marylie main OUTput file
case.dis  fort.14  output particle set DIStribution from tracking (input on 13)
case.tmo  fort.16  Transfer Map Output file
case.next fort.17  next Marylie main input file, containing fit solution.
case.wcl  fort.20  loop contents dump
case.env  fort.24  ENVelope plot file
case.olp  fort.26  Outer Loop Print wsq (aims and variables)
case.wsq  fort.28  Write Selected Quantities (aims and variables)
case.ilp  fort.30  Inner Loop Print of aims and variables (used in fit)
case.msc  fort.34  Marylie Sine Cosine basis rays, used for Poster rayplots.
case.geom fort.35  Geometry file for Poster beamline layouts
```

3.4 User Arithmetic

Multiple calls to `user9` allows the construction of elaborate algebraic combinations of existing computed quantities. These new constructs can also be used as design goals. Results are available using the key `u(n)` for the n^{th} entry in `ucalc`.

- There are 3 possible arithmetic operations: weighted sums, ratios, and products.
- The indices are used directly for the two single index quantities - Lie polynomial coefficients $f(k)$, and previously calculated ucalc entries $u(k)$.
- The double index items such as `R(i,j)` (R-matrix elements), and `tm(j,k)` (Taylor map components) are handled by packing the indices: set $k = 10i + j$ to reference $R(i, j)$. So

e.g. `R(1,2)` would be written as $R(12)$, i.e. $k = 12$. Likewise `tm(36,5)` ($= U_{5124}$) is indexed by $k = 365$ and called $T(k)$ below.

- All values of the constants (a, b) are allowed, and they may be both stored in the `ucalc` buffer (for job = 0) and retrieved from it (for job = -N). This allows building more complicated formulas by repeated calls to `user9`.
- Take $a = +1, b = -1$ to compute differences.
- To store map components themselves in `ucalc` use $a = +1, b = 0$.
- The following job ID sequence was chosen to be orderly with maximum backward compatibility:

job = -N	set job = +N and take constants from ucalc:	$a = u(ia)$, $b = u(ib)$
job = 0	store the constants in `ucalc` for future use:	$u(k) = a$, $u(m) = b$.
job = 1	weighted sums of R-matrix elements:	$u(n) = aR(k) + bR(m)$
job = 2	weighted sums of Lie coefficients:	$u(n) = af(k) + bf(m)$
job = 3	weighted sums of previous `ucalc` entries:	$u(n) = au(k) + bu(m)$
job = 4	weighted sums of Taylor coefficients:	$u(n) = aT(k) + bT(m)$
job = 5	ratios of Lie coefficients:	$u(n) = af(k)/[b + f(m)]$
job = 6	ratios of Taylor coefficients:	$u(n) = aT(k)/[b + T(m)]$
job = 7	ratios of previous `ucalc` entries:	$u(n) = au(k)/[b + u(m)]$
job = 8	ratios of R-matrix elements:	$u(n) = aR(k)/[b + R(m)]$
job = 9	products of Lie coefficients:	$u(n) = af(k)[b + f(m)]$
job = 10	products of Taylor coefficients:	$u(n) = aT(k)[b + T(m)]$
job = 11	products of previous `ucalc` entries:	$u(n) = au(k)[b + u(m)]$
job = 12	products of R-matrix elements:	$u(n) = aR(k)[b + R(m)]$
job = 13	RMS of the two ucalc entries:	$u(n) = \sqrt{u(k)^2 + u(m)^2}$
job = 14	reciprocal of variable k in selection loop m:	$u(n) = 1/x(k, m)$

4 Example: Focusing a Magnifier

This Marylie input file example (`rqmag.mip`) focuses a 3X magnifier in both planes. The quadrupole gradients in `#menu` are initial guesses. The `next` command after the fit writes a copy of this master file containing the new fitted values, for future use. Text after an exclamation point is a comment.

```
#comment
 Russian Quadruplet Magnifier Focus Example.  23 GeV protons.
  Mass =  938.2796   K.E.=    23000.0000 Mev   P=    23.919877 Mev/c
  Charge=  1.     Brho =   79.78812 Tesla-meters
  Beta (v/c)= 0.99923156   Gamma=    25.513147   Eta=    25.493542
#beam
  79.788121020000006
  24.513147629999999
  1.0000000000000000
  1.0000000000000000
#menu
 next     pmif
```

```
  1    17    2
 clear    iden
 begin    bip
   1000
 endfit   tip
    0
 beamin   usr8
  0.01  0.0   0.002  0.01  0.0   0.002   ! 1 cm X 2mrad beam
 aim      aim
   2   -7   0    1    1    3    ! select  aims from    ixd>:  7
 vary     vary
  -1   -8   0   0   1   3      ! select  variables from ixd>: 8
 fit      fit
  10    0    0   0.02   0   3
 qa       quad                          !
  0.6     50.0      1.0   1.0      !
 qb       quad                          !   Initial guesses
 0.9     -60.0      1.0   1.0     !
 qc       quad                          !
  0.9      60.0     1.0   1.0     !
 qd       quad                         !
  0.6     -50.0     1.0   1.0    !
 focal    drft
   1.0
 throw    drft                         ! Long final drift
   9.0
 gap      drft
  0.6
 center   drft
   1.4
 mapsav   tmo
   16
 linout   usr1
   1   1   20   1    3    0
 aper     ps9
  0.01   0  0  0  0  0
 fin      end
#xdata
 ixd>:  7
     R(1,2)=0.0  R(3,4) = 0.0  R(1,1) = -3.0 #     ! x and y focus, with -3X magnification
 ixd>:  8
       qa  2   qb  2  throw 1 #     ! Vary Russian Quadruole Gradients, and final drift
       qd  2
       1  -1.0
       qc  2
       2  -1.0
       #
#lines    !a Russian quadruplet magnifier
```

```
 russ
     1*aper        1*focal       1*qa          1*gap         1*qb       &
     1*center      1*qc          1*gap         1*qd          1*focal    &
     1*throw
#lumps
#loops  ! Unroll the full layout
 layout
     1*russ
#labor
    1*clear  ! clear the buffers
    1*russ   ! compute initial map
    1*aim    ! select aims
    1*vary   ! select variables
    1*begin  ! start of inner loop
    1*clear  ! clear the map
    1*russ   ! compute map of line
    1*fit    ! update variables
    1*endfit ! end of inner loop
    1*next   ! save the results. If you are here, the fit loop is finished
    1*clear  ! start over
    1*russ   ! compute final map
    1*mapsav ! save it to file 16
    1*beamin ! choose input beam moments
    1*layout ! expand the line
    1*linout ! report final design
    1*fin    ! all done
```

4.1 Resulting Output

File `rqmag.out`: First echo the aims, variables and targets. Then do 16 fit iterations that bounce off the bottom (machine precision). Quit on first error increase below 10^{-9}, and report final fitted beamline and resulting output beam. No target cuts were needed. Convergence was easy.

```
Marylie 2014 version with #xdata
           7  lines read into mltext common.
  1: id=  7  R(1,2)=0.0  R(3,4) = 0.0  R(1,1) = -3.0 #
  2: id=  8  qa  2   qb  2  throw 1 #
  3: id=  8  qd  2
  4: id=  8  1  -1.0
  5: id=  8  qc  2
  6: id=  8  2  -1.0
  7: id=  8  #

  Aims selected :
 No.     item        present value        target value
-----------------------------------------------------
  1    r(1,2) =      -1.61645789          0.00000000
  2    r(3,4) =       1.13230270          0.00000000
```

```
 3    r(1,1) =      -2.70692910         -3.00000000

  Variable #menu elements selected:
 No.  Element    Type     Parameter   Present value.
-------------------------------------------------------------
 1    qa         quad         2       50.000000000000
 2    qb         quad         2      -60.000000000000
 3    throw      drft         1       9.0000000000000

  Dependent #menu elements selected:
 No.  Element    Type     Parameter   Present value       IDV  Slope
-------------------------------------------------------------------------
 1   qd         quad       2      -50.000000000000        1  -1.00000
 2   qc         quad       2       60.000000000000        2  -1.00000
 Iter   1 Error= 1.6165E+00,  Step= 1.0000E+00,  SubErr= 1.6165E+00 @cut=    1
 -------------------------------------------------------
                   :
                   :
 -------------------------------------------------------
 Iter  15 Error= 2.2204E-15,  Step= 7.1607E-15,  SubErr= 2.2204E-15 @cut=    1
 -------------------------------------------------------
         16 = iter final value.
 Quit on iteration    16 for reason 2: Best effort: Hit bottom at full reach
 Final values with reach =    1 are:

  Aims selected :
 No.     item        present value        target value
-----------------------------------------------------
 1    r(1,2) =      2.220446049E-15      0.00000000
 2    r(3,4) =       0.00000000          0.00000000
 3    r(1,1) =      -3.00000000         -3.00000000

  New values for parameters:
 No.  Element    Type   Parameter   Present value         IDV  Slope
-------------------------------------------------------------------------
 1   qa         quad       2       54.299888446587
 2   qb         quad       2      -59.244687227295
 3   throw      drft       1       9.9915286067318
 4   qd         quad       2      -54.299888446587        1   -1.0000
 5   qc         quad       2       59.244687227295        2   -1.0000

  Maximum error is     2.220446E-15
  Maximum allowed was   0.00000


  Line has   6 drifts and   4 quads
  Beam Line Summary
   n  name     type   aper[cm] length[cm]    Gradient     PoleTip[g]   path[cm]
```

```
  1  focal    drift    1.00   100.000      0.0000000        0.00000   100.000
  2  qa        quad    1.00    60.000     54.2998884     5429.98884   160.000
  3  gap      drift    1.00    60.000      0.0000000        0.00000   220.000
  4  qb        quad    1.00    90.000    -59.2446872    -5924.46872   310.000
  5  center   drift    1.00   140.000      0.0000000        0.00000   450.000
  6  qc        quad    1.00    90.000     59.2446872     5924.46872   540.000
  7  gap      drift    1.00    60.000      0.0000000        0.00000   600.000
  8  qd        quad    1.00    60.000    -54.2998884    -5429.98884   660.000
  9  focal    drift    1.00   100.000      0.0000000        0.00000   760.000
 10  throw    drift    1.00   999.153      0.0000000        0.00000  1759.153
Total length =        17.59153[m],  Sum|GL| =   171.800303[T]
```

5 Beamline Rayplot Checks

A first order rayplot is a useful check on the solution just found, especially to verify that it is a ground state solution rather than an excited state. This uses `user1` to generate the sinelike and cosinelike basis rays (Dave Carey's notation), and a graphics package like POSTER to plot them.

5.1 Generating Basis Functions

To generate basis rays, put the beamline in a `#loop` to unroll any included lines to a flat array. With this named loop in the `#labor` segment, run Marylie `usr1` to slice the drifts and quadrupoles and generate a file of basis rays for the beam line of interest. So far this can only be a sequence of drifts, quads, and aperture settings. The aperture is taken to be p(1) of PS9 (Parameter Set 9), and applies to all following elements until reset by another aperture command Save on fort.34, renamed to case.msc (MarylieSineCosine). The USR1 parameters should be:

1. JOB = 1 to slice the elements and fill in the sincos buffer for future use.
2. MENV = 1 to write the sincos buffer to the .MSC file (lun=34)
3. MINCUT = minimum no. of slices/element (at least 7 slices for short elements like quads.)
4. STEP = nominal thickness of one slice. First try nslice=length/step, but it must be at least mincut. Actual final dz = length/nslice.
5. ISEND = 3 as usual for both jof and jodf output.
6. JTRAN = 0 for no translation files (unused option).

Content of the resulting .MSC file:

`line 1`= NEP, NFLG, WX, WY, BRHO, BETA, GAMM1:, where

NEP = points needed to draw quad blocks, NFLG = 0 is not used, (WX,WY) are the x,y achromatic correlation coefficients, and BRHO,BETA,GAMM1 are the usual kinematic parameters.

`lines 2,nep+1` = z, radius, G[T/M], n, typecode, name = beamline data
(at each element edge) for drawing quadrupole boxes

`lines nep+2,EOF` : z,r,cx,sx,cy,sy: cosine and sine basis rays at each z-point.

The achromatic correlation coeffients are computed from the Taylor map terms
WX = -tumat(12,1)/tumat(17,1) and
WY = -tumat(21,3)/tumat(24,3)

==========

5.2 POSTER Ray Plots

POSTER is a script driven program to generate PostScript plots from a variety of data types. The Mac OS displays PS plost as pdf's, which may be saved for inclusion in documents like this one. Appendix C lists all of the commands. The followng script sets plot scales, loads color tables, specifies number and spacing of rays in both position and angle, and draws the ray plot:

```
#page 1  6 606 360  782 .75 0.75  0  0
 #font  Helvetica
 #text Pathlength [m]
  0 0 0  300  380 0 14 0
 #area  81  574  421  720  1  3030  10  1
   0 0.0
17.60 -2.5  2.5  2 1 0 0
 #font Helvetica
 #text  Russian Quadrupole 3X Magnifier, 23 Gev/c protons.
  0 0 1 150  740 0 16 0
 #text Transverse position [cm]
  0 0 0  40 600  90 14 0
 #bbox
 #load 6 0.5 0.5             ! load 6 colors
 dot 1 0  1  1 0 2.5  20  0 0 0
 dot 2 0  1  1 0 2.5  20  0 0.7  0.0
 dot 3 0  1  1 0 2.5  20  1 0 0
 dot 4 0  1  1 0 2.5  20  0 0 1
 dot 5 0  1  1 0 2.5  20  0.5 0.4 0.0
 dot 6 0  1  3 0 1.5  20  1.0 0.9 0.7
 #rays  #file RQMag.msc
 2  4  2 1 -6 0.40 0.25  !    nrl,  nsa,  job,  ka,   kb,  delta,  dphi
 #text x-plane rays
 5 1 20 285 710 0 10 12
 0.00 mR
 0.25 mR
 0.50 mR
 0.75 mR
 1.00 mR
-----------------------------------------------------
Parameters:
         Rays to generate:
            nrl  = no. of ray locations to generate (on each side of axis)
            nsa  = no. of scattering angles to generate (each side of zero)
            job = 2 to generate achromatic rays using (wx,wy) from .msc file.
            job = 3 to generate plain uncorrelated rays (wx = wy = 0.0)
```

```
     signs: If job > 0 use same color key for both signs of scattering.
               If job < 0 use next color key for negative scattering angles.
View plane:
   ka = 1 for x-axis,
   ka = 2 for y-axis
Quad box flag:
   kb = 0 for none,
   kb = +K for inside boxes, where K=color key to use for the box.
   kb = -K for outside boxes.(K is skipped for the ray color sequence.)
Ray steps:
   delta= position step in [cm]
   dphi = scattering angle step [mR]
```