

A decorative frame consisting of blue lines and corner markers. A vertical line on the left and a horizontal line at the top meet at a small circle in the top-left corner. A horizontal line below the title and a vertical line on the right meet at a small circle in the bottom-right corner.

Tracking in magnetic fields

Advanced FLUKA Course

Magnetic field tracking in FLUKA

FLUKA allows for tracking in **arbitrarily complex magnetic fields**. Magnetic field tracking is performed by **iterations** until a given accuracy when crossing a boundary is achieved.

Meaningful user input is required when setting up the parameters defining the tracking accuracy.

Furthermore, when tracking in magnetic fields FLUKA accounts for:

- The **precession of the mcs** final direction around the particle direction: this is critical in order to preserve the various correlations embedded in the FLUKA advanced MCS algorithm
- The **precession of** a (possible) particle **polarization** around its direction of motion: this matters only when polarization of charged particles is a issue (mostly for muons in Fluka)
- The **decrease of the particle momentum** due to energy losses along a given step and hence the corresponding decrease of its curvature radius. Since FLUKA allows for fairly large (up to 20%) fractional energy losses per step, this correction is important in order to prevent excessive tracking inaccuracies to build up, or force to use very small steps

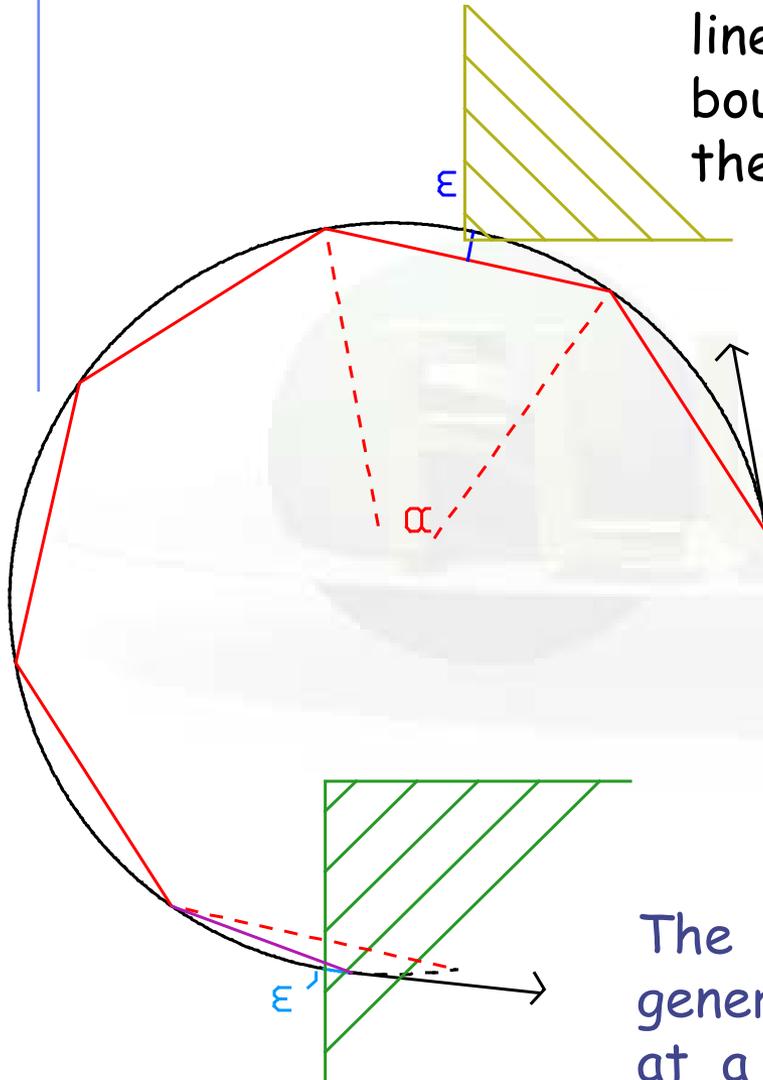
Magnetic field tracking in FLUKA

The true step (black) is approximated by linear sub-steps. Sub-step length and boundary crossing iteration are governed by the required tracking precision

The **red line** is the path actually followed, the **magenta segment** is the last substep, shortened because of a boundary crossing

- α = max. tracking angle (MGNFIELD)
- ϵ = max. tracking/missing error (MGNFIELD or STEPSIZE)
- ϵ' = max. bdrx error (MGNFIELD or STEPSIZE)

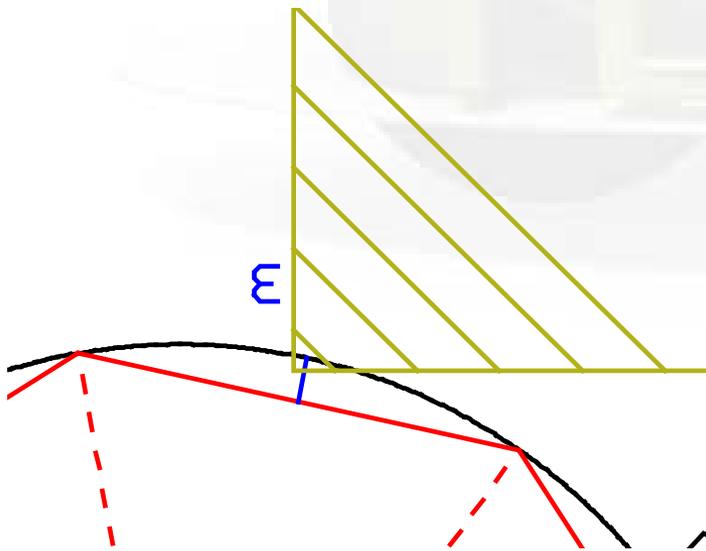
The end point is ALWAYS on the true path, generally NOT exactly on the boundary, but at a distance $< \epsilon'$ from the true boundary crossing (light blue arc)



Setting the tracking precision

MGNFIELD	α	ε	Smin	B_x	B_y	B_z
----------	----------	---------------	------	-------	-------	-------

- α largest angle in degrees that a charged particle is allowed to travel in a single sub-step. Default = 57.0 (but a maximum of 30.0 is recommended!)
- ε upper limit to error of the boundary iteration in cm (ε' in fig.). It also sets the tracking error ε . Default = 0.05 cm



IF α and /or ε are too large, boundaries may be missed (as in the plot).

IF they are too small, CPU time explodes..

Both α and ε conditions are fulfilled during tracking

→ Set them according to your problem

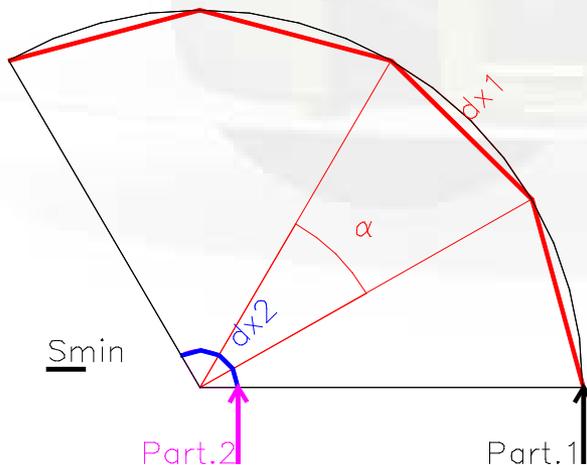
→ Tune ε by region with the STEPSIZE card

→ Be careful when very small regions exists in your setting : ε must be smaller than the region dimensions!

Setting the tracking precision

MGNFIELD	α	ε	Smin	B_x	B_y	B_z
----------	----------	---------------	------	-------	-------	-------

- **Smin** minimum sub-step length. If the radius of curvature is so small that the maximum sub-step compatible with α is smaller than Smin, then the condition on α is overridden. It avoids endless tracking of spiraling low energy particles. Default = 0.1 cm



Particle 1: the sub-step corresponding to α is $> S_{min}$ \rightarrow accept
Particle 2: the sub-step corresponding to α is $< S_{min}$ \rightarrow increase α

Smin can be set by region with the **STEPSIZE** card

Setting precision by region

STEPSIZE	Smin/ ϵ	Smax	Reg1	Reg2	Step
----------	------------------	------	------	------	------

- Smin: (if what(1)>0) minimum step size in cm
Overrides MGNFIELD if **larger** than its setting.
- ϵ (if what(1)<0) : max error on the location of intersection with boundary.
 - The possibility to have different "precision" in different regions allows to save cpu time
- Smax : max step size in cm. Default:100000. cm for a region without mag field, 10 cm with mag field.
 - Smax can be useful for instance for large vacuum regions with relatively low magnetic field
 - It should not be used for general step control, use EMFFIX, FLUKAFIX if needed

Possible loops in mag.fields

- Although rare, it is **PHYSICALLY** possible that a particle **loops for ever** (or for a very long time). Imagine a stable particle generated perpendicularly to a uniform B in a large enough vacuum region: it will stay on a circular orbit forever !
- Suppose now that the orbit enters in a non-vacuum region (here we can at least loose energy..) but the boundary is missed due to insufficient precision. This results again in a never-ending loop.

Luckily, it almost never happens. **Almost.**

Example: uniform dipole field

MGNFIELD	α	ε	Smin	B_x	B_y	B_z
----------	----------	---------------	------	-------	-------	-------

- Uniform magnetic fields can be defined through the **MAGFIELD** card without need of programming / compiling. In this case B_x , B_y , B_z are the components of the magnetic field, not the normalized vectors, i.e. $B_x \neq B_{TX}$, ...
- The field for a dipole of radius R [cm] should normally be set to (CS is the charge state, e.g. 1 for e^- or proton):

from **(BEAMCM)**

$$B[T] = 1D+13 * Pbeam[GeV/c] / (Clight[cm/s] * R [cm] * CS)$$

from **(DBLPRC)**

- For any other case, B_x , B_y and B_z need to be set to 0.0 and `magfld.f` must be customized and compiled.
- In all cases, magnetic field must be activated in the given region through **ASSIGNMA**, `WHAT(5) = 1.0`

The magfld.f user routine

This routine allows to define arbitrarily complex magnetic fields:
(uniform fields can be defined through the MGNFIELD card)

SUBROUTINE MAGFLD (X, Y, Z, BTX, BTY, BTZ, B, NREG, IDISC)

Input variables:

x,y,z = current position

nreg = current region

Output variables:

btx,bty,btz = cosines of the magn. field vector

B = magnetic field intensity (Tesla)

idisc = set to 1 if the particle has to be discarded

- All floating point variables are double precision ones!
- BTX, BTY, BTZ must be normalized to 1 in double precision
- Magfld .f is called only for regions where a magnetic field has been declared through ASSIGNMAT

magfld: example toroidal field

```
SUBROUTINE MAGFLD ( X, Y, Z, BTX, BTY, BTZ, B, NREG, IDISC )
```

Standard FLUKA includes : KEEP THEM

```
INCLUDE '(DBLPRC)'  
INCLUDE '(DIMPAR)'  
INCLUDE '(IOUNIT)
```

```
★ INCLUDE '(NUBEAM)'
```

```
IF ( NREG .EQ. NRHORN ) THEN  
  RRR = SQRT ( X**2 + Y**2 )  
  BTX = -Y / RRR  
  BTY = X / RRR  
  BTZ = ZERZER  
  B = 2.D-07 * CURHOR / 1.D-02 / RRR  
END IF
```

This gives a versor \perp radius
in a plane \perp z axis

In this case, the cosines are
automatically normalized.
Otherwise, user **MUST**
ensure that
 $BTX2+BTY**2+BTZ**1=ONEONE$**

B intensity depending
on R and current

USEFUL TIP

This is a user defined include file, containing for example
`COMMON /NUBEAM/ CURHORN, NRHORN,`
It can be initialized in a custom **usrini.f** user routine, so
that parameters can be easily changed in the input file

magfld: example solenoid...

Different fields in different regions:

```
IF ( NREG .EQ. NRHORN ) THEN
```

```
.....  
ELSE IF ( NREG .EQ. NRSOLE ) THEN
```

```
  BTX = ZERZER
```

```
  BTY = ZERZER
```

```
  BTZ = ONEONE
```

```
  B  = SOLEB
```

```
ELSE IF ( NREG .EQ. NRMAP ) THEN
```

```
★ CALL GETMAP ( X, Y, Z, BTX, BTY, BTZ, B)
```

```
ELSE
```

```
  WRITE ( LUNOUT, *) 'MGFLD, WHY HERE ?
```

```
  WRITE ( LUNOUT, *) NREG'
```

```
  STOP
```

```
END IF
```

This gives a perfect solenoid field

Intensity calculated at initialization

Get values from field map

Add a bit of protection.

The user can add **more routines**, they have to be included in the linking procedure

Always :

include the three standard FLUKA INCLUDEs

use FLUKA defined constants and particle properties for consistency

Possible, not explained here : call C routines

magfld.f multipoles

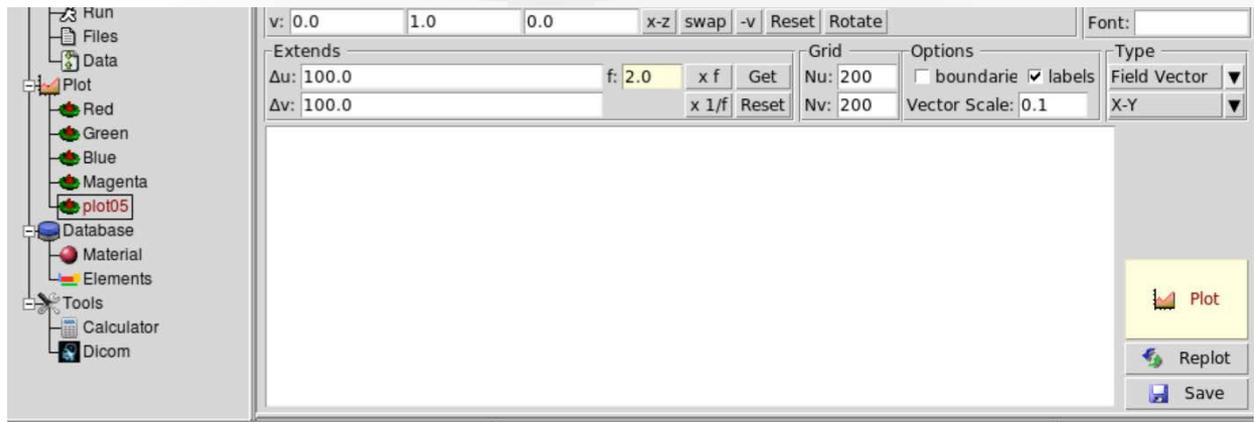
Magnetic scalar potential for magnet with $2 \times m$ poles, e.g. $m=1 \rightarrow$ dipole, $m=2 \rightarrow$ quadrupole, etc.:

$$\Phi_m = r^m \{A_m \cdot \cos(m \cdot \theta) + B_m \cdot \sin(m \cdot \theta)\}$$

Coordinates can then be transformed to Cartesian and B_m computed as:

$$B_m = -\nabla\Phi_m$$

This includes normal and skew configurations, e.g. for normal quadrupole $\rightarrow A_2=0$



You can visualize the magnetic field (intensity or vector) map with Flair (not Geometry Editor)

magfld.f multipoles - quadrupoles

Quadrupoles

$$RRR = \text{SQRT} (X^{**2} + Y^{**2})$$
$$B = (K1 * 1D-04) * (BRho) * RRR$$

With

$K1 [m^{-2}]$ from MAD lattice

$BRho [T cm] = 1D+09 * Pbeam / Clight$

[GeV/c] from (BEAMCM)

[cm/s] from (DBLPRC)

Normal Quadrupoles

$$BTX = Y / RRR$$
$$BTY = X / RRR$$
$$BTZ = \text{ZERZER}$$

Skew Quadrupoles

$$BTX = -X / RRR$$
$$BTY = Y / RRR$$
$$BTZ = \text{ZERZER}$$

Remember: $\mathbf{Fm} = q \mathbf{v} \wedge \mathbf{B}$

change X/Y signs to reverse polarity

magfld.f multipoles - sextupole

$$BTX = 5.D-01 \times (K2 \times 1D+06) \times BRho \times (X^{**2} - Y^{**2})$$

$$BTY = (K2 \times 1D+06) \times BRho \times (X * Y)$$

$$BTZ = ZERZER$$

$$B = \text{SQRT}(BTX^{**2} + BTY^{**2})$$

$$BTX = BTX/B$$

$$BTY = BTY/B$$

With

$K2 [m^{-3}]$ from MAD lattice

$B0 * rho [T cm] = 1D+09 * Pbeam / Clight$

[GeV/c] from (BEAMCM)

[cm/s] from (DBLPRC)

Initialization/output routines

Very useful to **initialize** and propagate variables common to other user routines:

- **usrglo.f** knows **nothing** about the simulations, but can provide information to the other initialization stages. Can be called many times **USRGCALL**
- **usrini.f** knows **everything** about the problem. Here one can, for instance, use information about materials, regions etc. (can use names) **USRICALL**
- **usrein.f** is useful when doing **event-by-event** user scoring, it can for instance reset and reinitialize event-dependent user quantities **(no card)**

Associated **OUTPUT** routines:

- **usrout.f** called at the **end of the run** **USROCALL**
- **usreout.f** called at the **end of each event** **(no card)**

usrini.f : example

```
SUBROUTINE USRINI ( WHAT, SDUM )
```

```
INCLUDE '(DBLPRC)'  
INCLUDE '(DIMPAR)'  
INCLUDE '(IOUNIT)
```

Default declarations

```
.....  
DIMENSION WHAT (6)  
CHARACTER SDUM*8
```

```
.....  
CHARACTER MAPFILE(8)  
INCLUDE '(NUBEAM)'
```

Here we store our variables

```
IF ( SDUM .EQ. 'HORNREFL' ) THEN  
  NRHORN = WHAT (1)  
  CURHORN = WHAT (2)  
ELSE IF (SDUM .EQ. 'SOLENOID') THEN  
  SOLEB = WHAT (2)  
  NRSOLE = WHAT(1)
```

Here we initialize region numbers
And parameters for the magfld.f
routine

contnd

usrini.f: example contnd

```
ELSE
  MAPFILE=SDUM
  MYUNIT=21
  CALL OAUXFI ( MAPFILE, MYUNIT, 'OLD' , IERR)
  CALL READMAP(MYUNIT)
  CLOSE (21)
  NRMAP= WHAT (1)
END IF
RETURN
```

Use the SDUM field to read the name of the magnetic field map file

Open the field map

Call a user procedure that reads and stores the field map to be used by magfld.f

This usrini needs 3 cards to initialize all parameters, i.e.:

```
USRICALL MyHorn 150000. HORNREFL
USRICALL MySole 1.3 SOLENOID
USRICALL Mapped myflmap
```

The region **names** in the what's are automatically parsed and converted to region **numbers** by FLUKA
(same would happen with materials, scoring ..)

Roto-translation routines:

SUBROUTINE DOTRSF (NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)

...
SUBROUTINE DORTNO (NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)

...
SUBROUTINE UNDOTR (NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)

...
SUBROUTINE UNDRTO (NPOINT, XPOINT, YPOINT, ZPOINT, KROTAT)

...
DIMENSION XPOINT (NPOINT), YPOINT (NPOINT), ZPOINT (NPOINT)

...
The **DOTRSF** routine executes the **KROTAT_{th}** transformation as defined by **ROT-DEFI** on **NPOINT** points, defined by the **X,Y,ZPOINT** arrays, *with a (possible) translation* included

DORTNO does the same *without the translation* (eg for velocity vectors)

UNDOTR performs the *inverse* transformation, *with a (possible) translation* included

UNDRTO performs the *inverse* transformation, *without the translation*

Magfld.f & Roto-translation routines

Translate coordinates to magnet prototype, where B is easily defined, e.g. quadrupole aligned with Z axis at $X=0, Y = Y_0$:

```
SUBROUTINE MAGFLD( X, Y, Z, BTX, BTY, BTZ, B, NREG, IDISC)
[...]
```

INCLUDE 'LTCLCM'

```
[...]
```

XLOC = X; YLOC = Y; ZLOC = Z

CALL DOTRSF (1, XLOC, YLOC, ZLOC, MLATTC)

YLOC = YLOC - Y0

RRR = SQRT(XLOC**2+YLOC**2)

IF (RRR.gt.tol)

BTX = YLOC/RRR

BTY = XLOC/RRR

BTZ = ZERZER

B = k * RRR

```
[...]
```

Bring position to prototype

Transform Y to origin

Except if
RRR → 0

Apply 'standard' quad field expressions

Compute B modulus

Then magnetic field vector is then anti-rotated (not translated!) to the replica using the same rotation index:

```
CALL UNDRTO ( 1, BTX, BTY, BTZ, MLATTC )
```

Rotate B vector to replica orientation,

Randomizing B: ray-tracing

- A large fraction of accidents in accelerators with potential radiation protection consequences is linked to one or several **magnets having wrong field intensities** (operator settings, magnet short, power failure, error in polarity,...).
- The RP analysis of a machine should include the definition of the **phase space of mis-steered beams (ray-trace)**. **Collimators** are installed to shield rays that could lead to unacceptable exposure.
- **FLUKA 3D geometry and magnetic tracking capabilities** provide a powerful way to obtain the mis-steering ray-trace, especially when several magnets are involved

Randomizing B: ray-tracing, process

- 1) Check transport for nominal conditions
- 2) Set a new random value of B for every primary, e.g.:

```
DATA partnum / 2 /
```

```
...
```

```
IF ( partnum. ne . Numpar(1) ) THEN
```

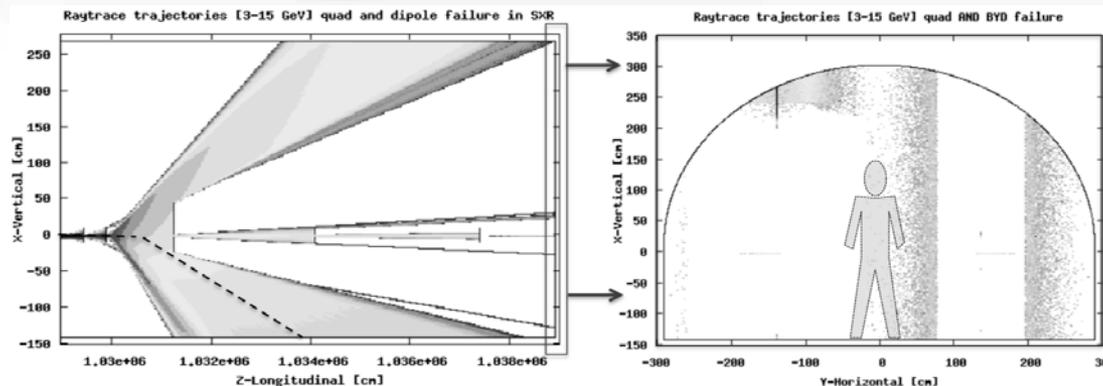
```
    B = ( FLRNDM() - 0.5D0 ) * BO
```

```
END IF
```

```
...
```

*Strength of dipole is randomized between $[-BO, BO]$
Mis-alignment, and other effects can be introduced similarly*

- 3) Set all materials to vacuum, except for collimators
- 4) Plot trajectories. Intensities $\sim k * \text{probability}$



"Monte Carlo simulation of beam mis-steering at electron accelerators", in proceedings of SATIF-11

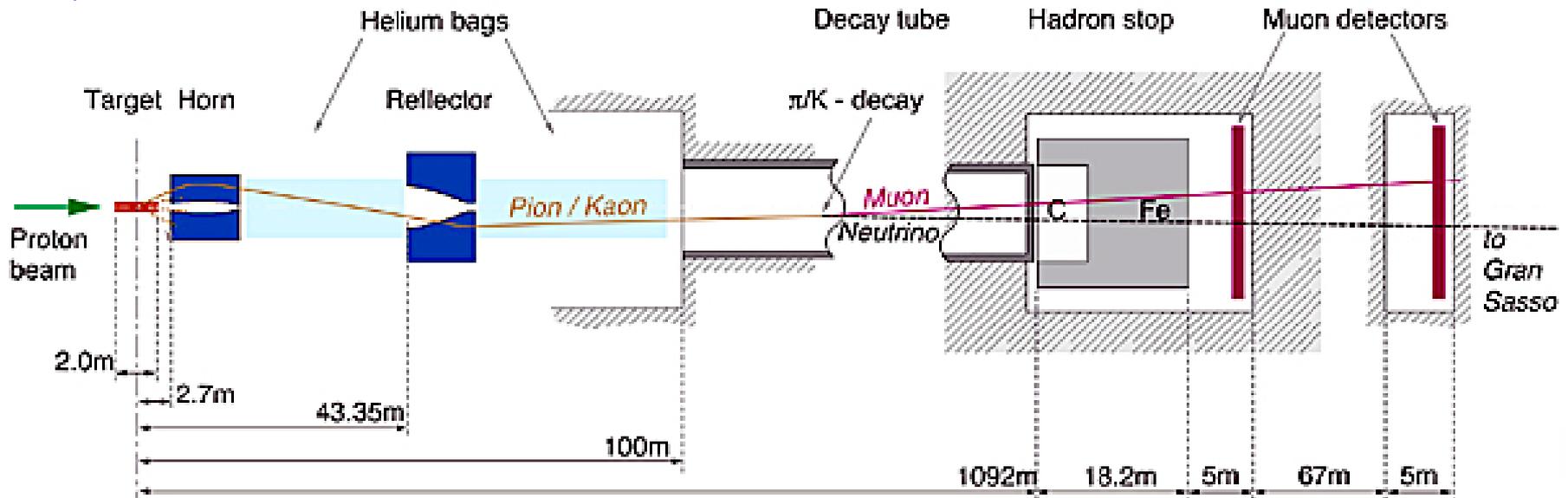
Ray-tracing in an XFEL dump-line

END



Example: magnetic field in CNGS

Cern Neutrino to Gran Sasso



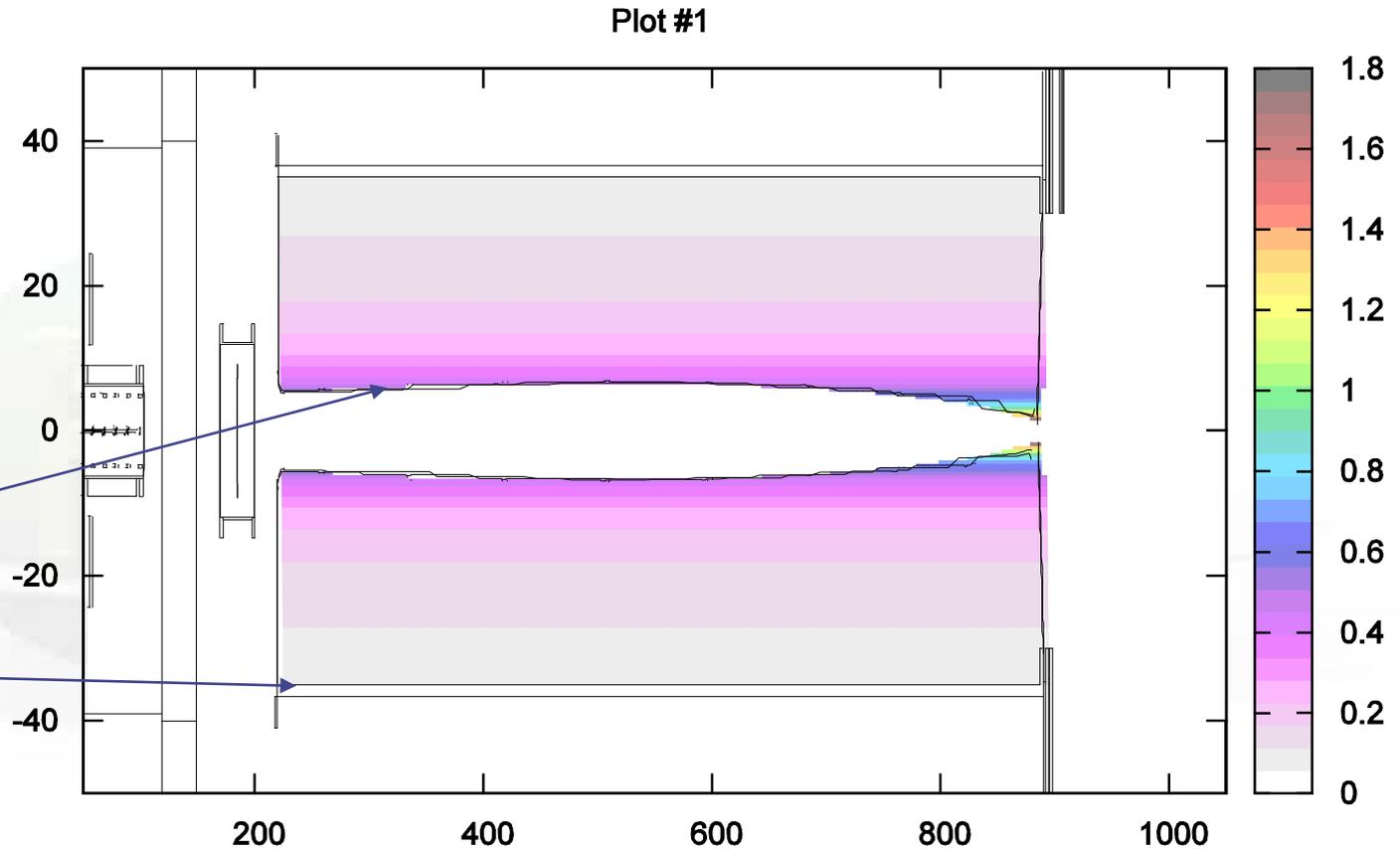
The two magnetic lenses (blue in the sketch) align positive mesons towards the Decay tunnel, so that neutrinos from the decay are directed to Gran Sasso, 730~km away
Negative mesons are deflected away
The lenses have a finite energy/angle acceptance

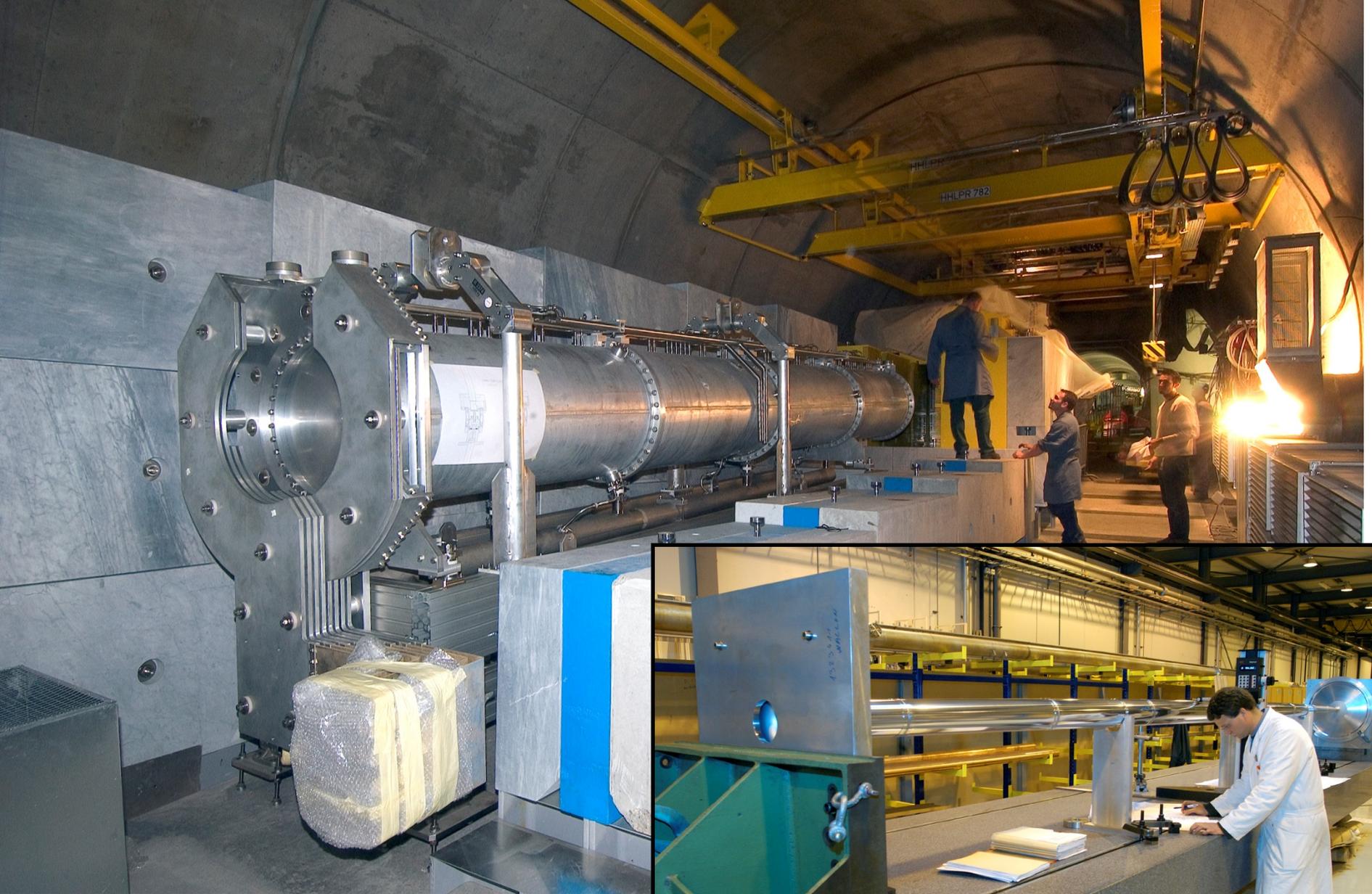
Example : the magfld.f routine

Magnetic field intensity in the CNGS horn

A current $\approx 150\text{kA}$, pulsed, circulates through the Inner and Outer

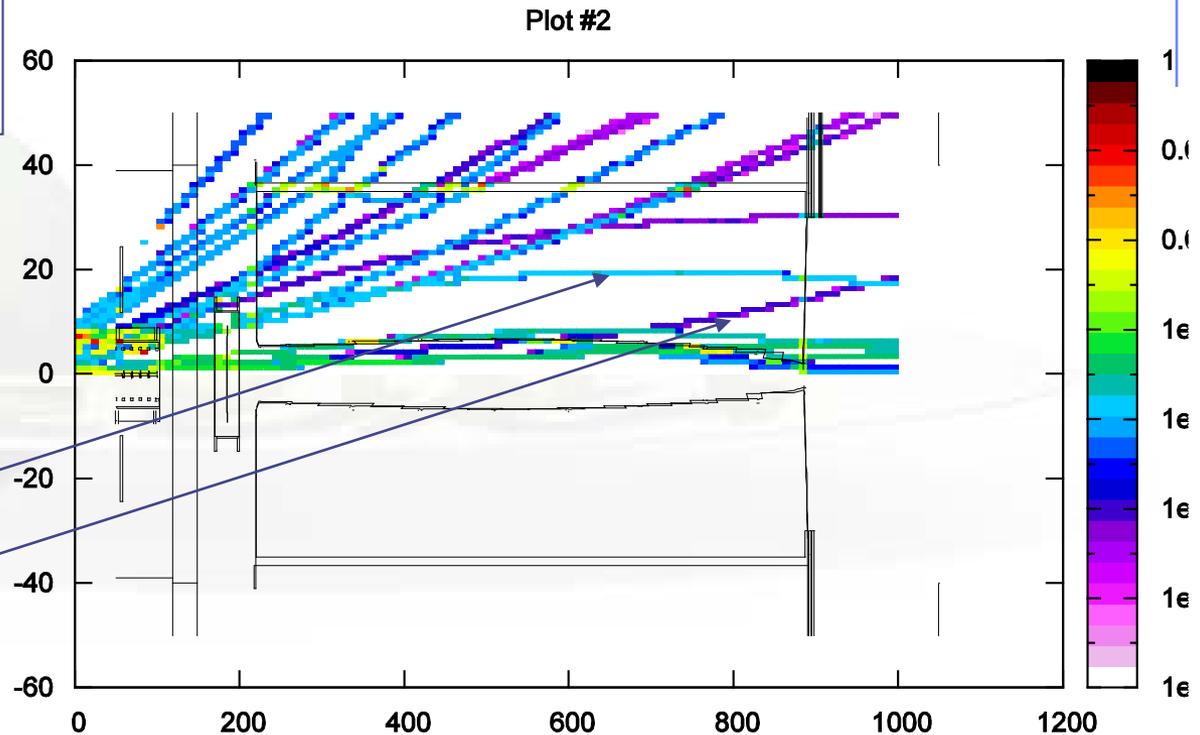
conductors
The field is toroidal,
 $B \propto 1/R$





magfld: results

charged particle tracks
in the CNGS geometry
1 event
USRBIN R-Z



Focused
De-focused
Escaping..many

usrglo.f :example

```
SUBROUTINE USRGLO ( WHAT, SDUM )
```

```
INCLUDE '(DBLPRC)'  
INCLUDE '(DIMPAR)'  
INCLUDE '(IOUNIT)
```

Default declarations

```
.....  
DIMENSION WHAT (6)  
CHARACTER SDUM*8  
INCLUDE '(NUBEAM)'
```

Here we store our variables

```
IF ( WHAT(1) .GT. ZERZER ) THEN  
  ROTTRG = WHAT(1)  
  LTGMISA = .TRUE.  
  TRATARG = ZERZER  
  IF ( WHAT(2) .GT. ZERZER ) TRATARG = WHAT(2)  
RETURN
```

Suppose we have a lattic.f routine

That rotates the target to simulate misalignment : here a flag and the rotation / translation amounts are set