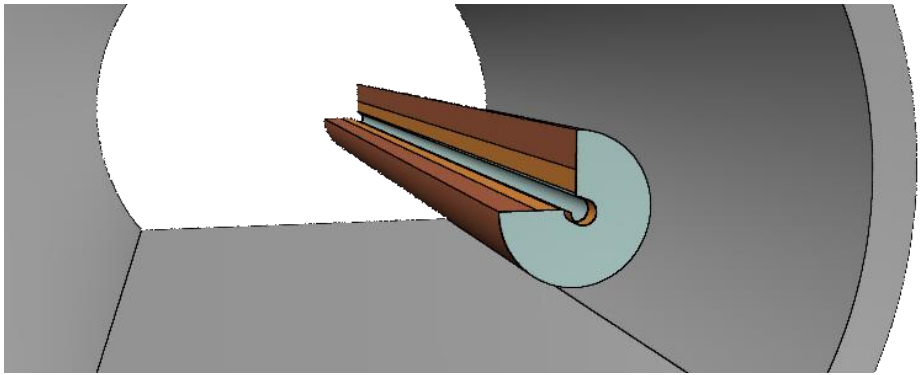
User routines: exercise

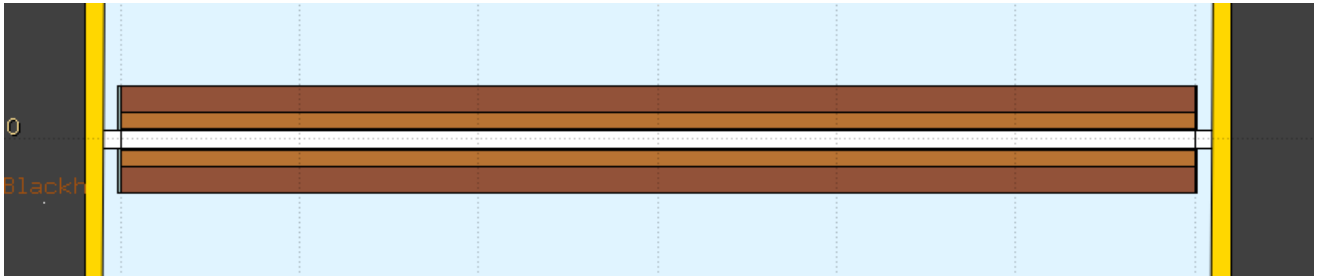
How to tailor FLUKA to non-standard user needs

Introduction and scope

- Case study
  - Toy model electron/positron accelerator

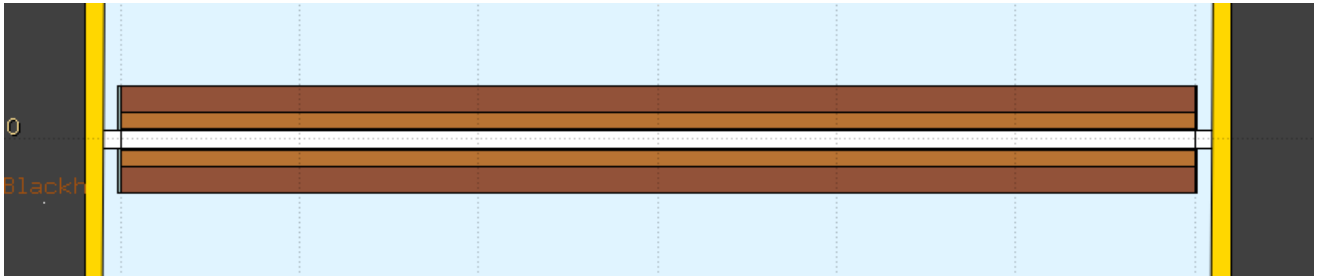


- User routine usage
  - **`usrmed.f`** to reinject particle from one side to the other
  - **`fluscw.f`**: momentum/energy fluence



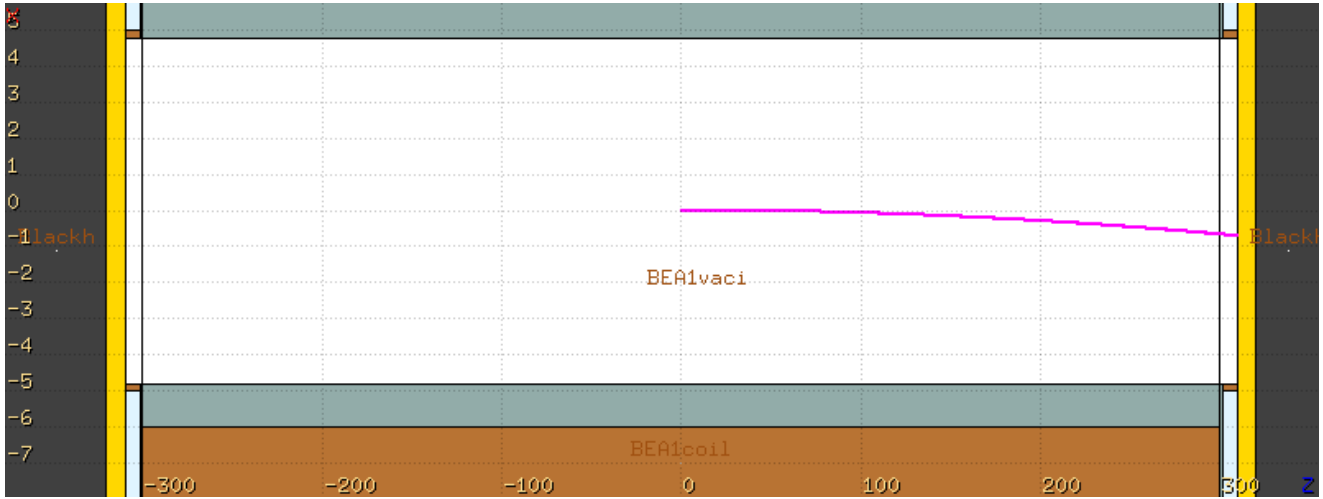
Case study

- We want to bend a beam of 100 GeV positrons with a series of dipoles. As arbitrary value, we consider the magnetic field to be 0.5 T. Each magnet is 6 meters long
- A geometry consisting of a dummy magnet inserted in a tunnel is provided
- We do not want to simulate this geometry by using lattices and transformations (i.e. having many singular elements), but by exploiting the symmetry and reinjecting the particles from one side to the other



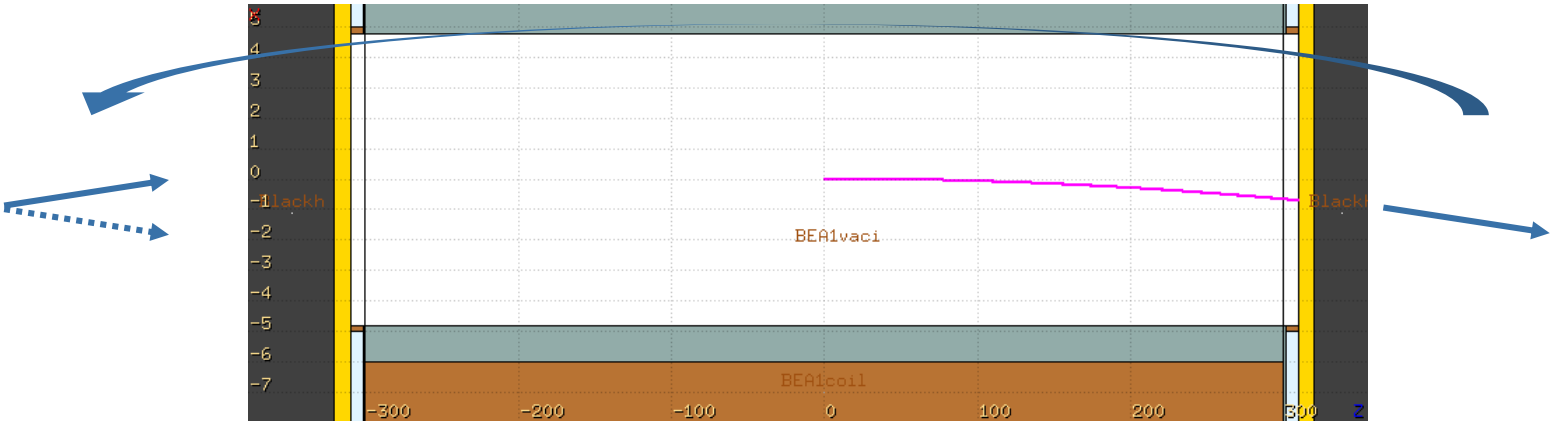
Problem 1: trivial

- Check if the geometry is correctly implemented: at which angle does the beam exit the magnet?
  - Derive the angle from the magnetic rigidity formula
  - Check this value in the geometry
  - Optional: can you use **`mgdraw`** to retrieve it? How accurate is it?



Problem 2: not so trivial

- Now we know how the beam behaves in the machine. We want to reinject it. You need to write a **`usrmed.f`** routine that:
  - Is activated only when the particle leaves the magnet. (Maybe you can activate it only in gold!)
  - Flips the beam position across the axis (z becomes –z)
  - Rotates the beam direction in the correct position



Problem 2: little hint

- If you want, you can proceed and start writing your routine from scratch. It can be a frustrating but certainly rewarding experience. But, in case you are lost, start from the one you have already.
- Once finished, try to shoot positrons whose momentum is larger or smaller than the nominal one. Where do they go?
- Optional: what happens if you shoot the ideal particle? Infinite tracking?! Try enabling the synchrotron radiation

```
      PARAMETER ( ALPHOU = ??? )
      PARAMETER ( TOLKIC = 0.1D+0 )
      IF( LFIRST ) THEN
*     It can be a good idea to initialize the sine
*     and cosine value for the rotation matrix
         SINALP = ???
         COSALP = ???
         LFIRST = .FALSE.
      END IF
```

Insert the angle of deflection when the particle leaves the magnet. (This is **half** of the total angle of deflection from the bending magnet!)

Initialise the matrix element of the rotation

Problem 2: little spoiler

```
      PARAMETER ( ALPHOU = ??? )
      PARAMETER ( TOLKIC = 0.1D+0 )
      IF( LFIRST ) THEN
*     It can be a good idea to initialize the sine
*     and cosine value for the rotation matrix
         SINALP = ???
         COSALP = ???
         LFIRST = .FALSE.
      END IF
```

Insert the angle of deflection when the particle leaves the magnet. (This is **half** of the total angle of deflection from the bending magnet!)

Initialise the matrix element of the rotation

```
*     Position part: flip the z coordinate
      XX_OLD = XX
      ZZ_OLD = ZZ
      XX = ???
      ZZ = ???
```

Just change the sign for z!

```
*     Direction part: rotate the vector (it is a
*     simple rotation matrix)
      TXX_OLD = TXX
      TZZ_OLD = TZZ
      TXX     = ???
      TZZ     = ???
```

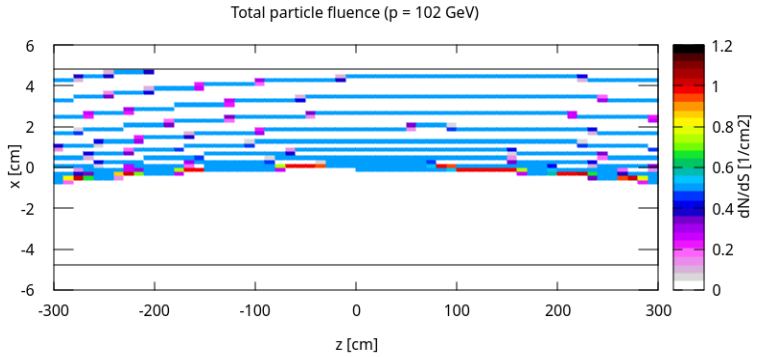
Write the rotation matrix:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

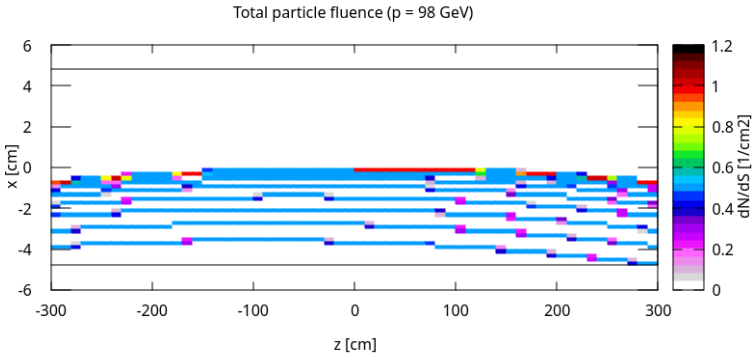
Problem 2: what to expect

- A USRBIN has been provided in the inputfile. Try plotting the total particle fluence in both cases
- Optional: can you generate similar plots with USRDUMP in Geoviewer?

p > 100 GeV



p < 100 GeV

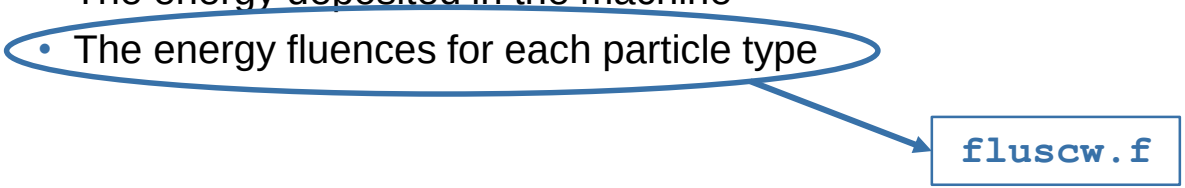


Problem 3: scoring energy fluence

- Now you have the equivalent of an infinite series of dipoles.

- Imagine that, for some reason, when a bunch is in the middle of the magnet, your magnets lose part of their field and they go from 0.5 to 0.4 T. Oh no! The positrons will impact on the machine.

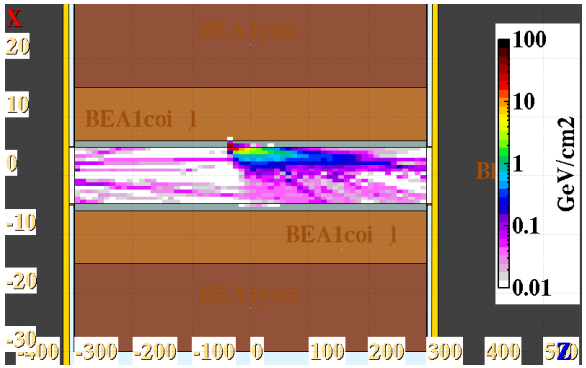
- We want to score several quantities, but mainly:
  - The energy deposited in the machine
  - The energy fluences for each particle type



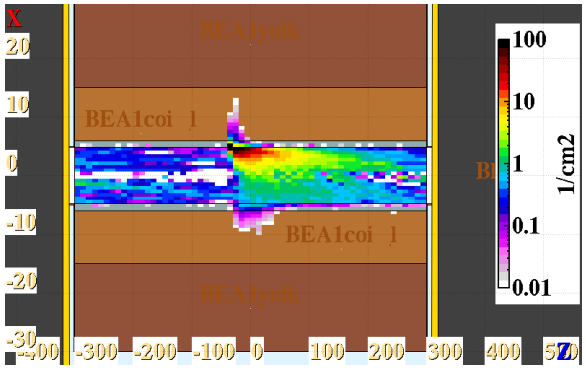
Problem 3: what to expect

- In the input file, you have 2 sets of USRBIN scoring: .23 and .24
- For all the scoring in .24 we just want to score the particle fluence
- In case of the scoring in .23, we want to score the particle fluence weighted by their kinetic energy
- You need to write an appropriate **`fluscw.f`** user routine and activate it via USERWEIG card

Photon kinetic energy fluence: ϕE



Photon fluence: ϕ



Problem 3: tips

```
*----------------------------------------------------------------------*
*
      INCLUDE 'scohlp.inc'
      INCLUDE 'paprop.inc'
*
      FLUSCW = ONEONE
      LSCZER = .FALSE.
*     Put an IF condition to change FLUSCW only for the correct
*     detectors (use JSCRNG )
      IF ( ??? ) RETURN
*     Ignore heavy ions:
      IF ( IJ .LE. -6 ) THEN
         FLUSCW = ZERZER
         RETURN
      ENDIF
*     Calculate the kinetic energy:
      ???
*     Apply the weight:
      FLUSCW = ???
      LSCZER = .FALSE.
      RETURN
*=== End of function Fluscw ===========================================*
      END
```

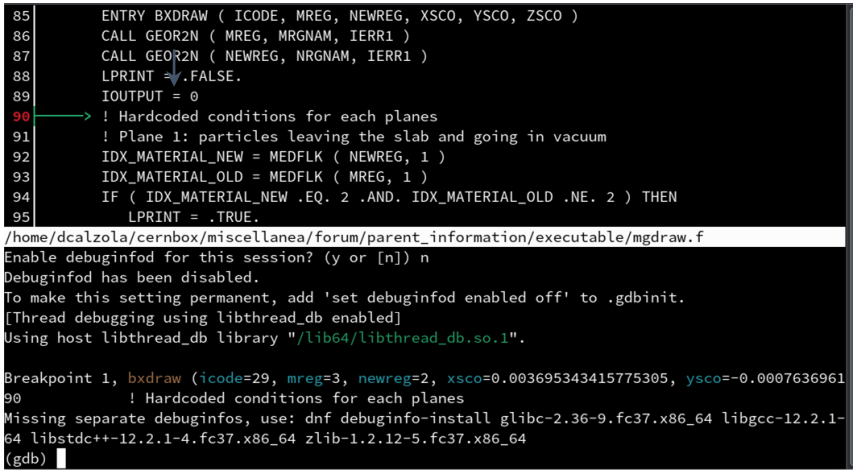
Debugging (example)

```
[me@localhost myFolder]$ /usr/local/fluka/bin/rfluka -g cgdb -e executable/fluka.x -N0 -M1 test.inp
```

```
New UI allocated
(gdb) b 86
Breakpoint 1 at 0x407b60: file mgdraw.f, line 90.
```

```
(gdb) r
Starting program: myFolder/executable/fluka.x myFolder/test.inp
```

We ask for a breakpoint on line 86, it accepted it on line 90



Debugging window: in the top part there is the source, while instruction are given in the bottom. You can:

- **[s]**tep to the next instruction
- go to the **[n]**ext line
- **[b]**ack**[t]**race: prints a stack trace, listing each function and its arguments
- **[p]**rint variable value
- …and much more!


FLUKA

Most important commons

- Always to be added:
  - **dblprc.inc** contains as parameters most commons physical and mathematical constants. Here lies the implicit declaration for variables: IMPLICIT DOUBLE PRECISION (A-H,O-Z)
  - **dimpar.inc** dimensions of the most important arrays
  - **iounit.inc** logical input and output unit numbers (1 to 19 are reserved)

- Few tips:
  - Use DBLPRC parameters when possible
  - Pay attention to typos in numerical constants! With implicit declaration (as of today): TWOTHI = ⅔, TWOTHR = 0
  - If you are using a common block, do not redefine an existing variable within your routines

Other important commons

The most important commons can also be found in the FLUKA manual (13.1).
A few selected ones are:

- **BEAMCM** properties of primary particles as defined by BEAM and BEAMPOS
- **EMFSTK** electromagnetic stack (for e+/- and photons)
- **SOURCM** user variables and information for a user-written source
- **FHEAVY** stack of heavy secondaries created in nuclear evaporation
- **FLKMAT** material properties
- **RESNUC** properties of the current residual nucleus
- **FLKSTK** main FLUKA particle stack
- **TRACKR** TRACKs Recording (properties of the currently transported particle and its path)
- **PAPROP** particle properties (masses, charges, etc.)