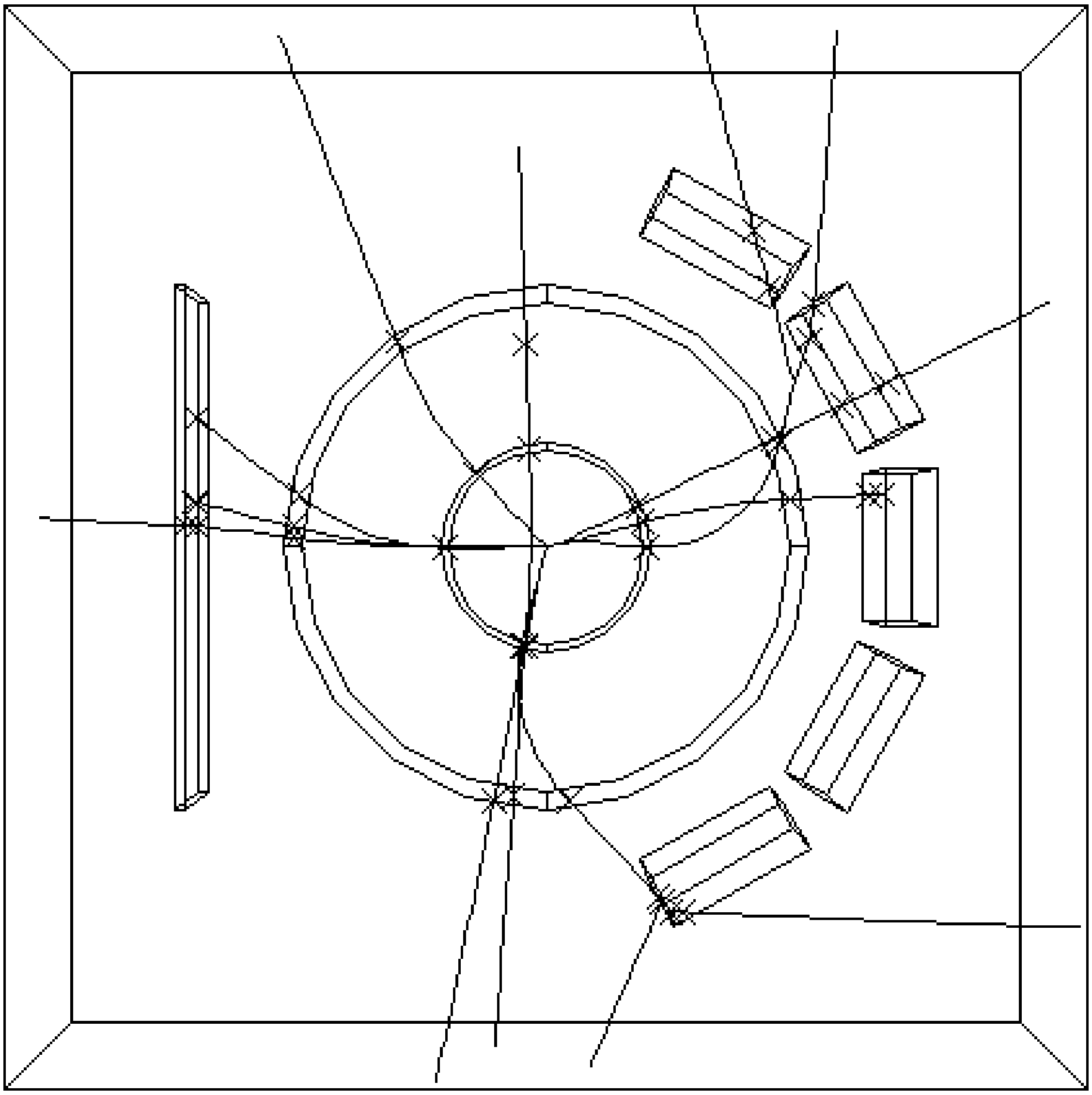
The "ABC" Particle Physics Simulator

David Ambrose
PHY 381C
May 1, 1997

Abstract

This project involved the creation of a *Monte Carlo* simulation package for experimental nuclear and high-energy physicists, designed for performing quick calculations, and for learning about particle physics in general.



1 Introduction

1.1 Motivation

With the growing size and complexity of today's nuclear and high-energy physics experiments, physicists have come to rely on computer simulations for both detector design and analysis of data. When each channel of detector information may cost thousands of dollars, simulations can determine the exact positioning and dimensions of detector elements to maximize acceptance and reduce expenses. Furthermore, computer simulations provide a means for calculating the performance of detector apparatus and understanding the results of the experiment. For example, a given detector may have many parameters, each of which affects the efficiency and acceptance for certain particles or events of interest. With the particles themselves undergoing potentially hundreds of possible interactions within the apparatus, analytically determining some global detector performance, such as efficiency, might prove impossible.

However, if we generate individual particles propagating through the apparatus in a step-by-step fashion, and apply random, but well known, independent physical processes to the particle at each step, then we can determine global detector performance over many such particles. The term *Monte Carlo* simulation derives from this use of independent, random events. The physics applied to the particle at each step, as it "swims" through the detector, might include curvature due to magnetic fields, energy loss, or decay into even more particles. In this manner, computer simulations also become an excellent tool for learning about particle physics, because they allow a student to visualize the complex behaviours of particle interactions.

Of course, the accuracy of *Monte Carlo* simulations depends on how well we approximate the physics at each step, and the variety of possible interactions increases with desired accuracy. To realistically simulate particles passing through matter, we might need to include higher-order processes such as pair-production, δ-rays, or hadronic showers, and the descriptions of the detector materials themselves can become quite complex. This added realism, while attainable [1, for example], makes particle physics simulators increasingly large, and cumbersome to use, thus limiting their effectiveness for students.

I have created the particle physics *Monte Carlo* "ABC", which applies a small set of fundamental physical processes common to most nuclear and high-energy physics experiments and, therefore, has a broad range of uses. The code has a structure designed for ease of use and quick results, so that an experimentalist can perform simple estimates and checks of more complex apparatus, while a student can design and test his or her own particle detector experiments as a means of learning about particle physics. In doing so, I have also learned much about the requirements and limitations of *Monte Carlo* simulation, which has an ever-increasing importance to my field of interest.

1.2 Project goals

The "ABC" simulator approximates the following physical processes, which are applicable to most particles in their transport through media:

- particle decay
- curvature in magnetic fields
- energy loss
- multiple scattering

Since this small set of processes and their approximations limit the potential usefulness of the simulation, I have attempted to modularize the code to accommodate additional user-defined processes, if needed, or improvements to the above approximations.

Simulation programs for complex experiments, with hundreds or thousands of different detector objects, most easily define their apparatus within the code itself. Unfortunately, any changes to the system require code editing and recompiling. For a student *wanting* to move objects around

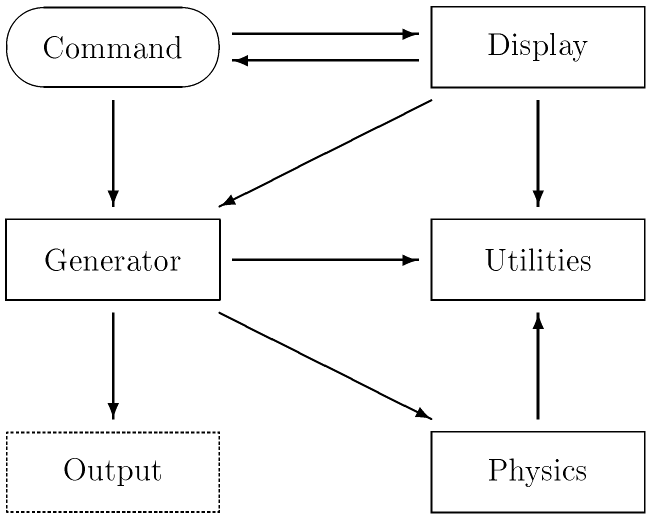


Figure 1: Code outline for the "ABC" simulator

to study effects, this *dead-time* might prove frustrating. I have therefore structured "ABC" to run from a command-line level, accepting input from the keyboard or script files for quick adjustments to the system, thus remaining independent of specific experiments.

Visualization of particle trajectories throughout the apparatus provides tremendous insight into both detector performance and particle interactions. "ABC" creates a graphics window where the user can view detectors and particles during the simulation. For a more quantitative analysis, the user can output particle hit information from each detector, for input to a plotting package, for instance.

Most importantly, I have strived to keep the code simple and compact, so that a user can quickly retrieve, compile, and run the program. I provide an example apparatus script, with default particle and material definitions, so the user can immediately run the simulation, or edit these files for specific experiments and needs. To this end, I have made the simulator code available on:

"http://wwwrel.ph.utexas.edu/Public/Students/ambrose/abc"

The text tile $abc.README$ (see appendix C) describes how to install and run the "ABC" simulator.

1.3 General code outline

Figure 1 diagrams the structure of the "ABC" program. Upon startup, the user enters the $\boxed{\text{Command}}$ routine, which reads and lists data (e.g. objects, particles, and other definitions) from the command-line. Additionally, this routine can set process flags and constants that might need adjustment for particular situations. From the command-line, the user can call the $\boxed{\text{Display}}$ routine, which opens the graphics window and draws the apparatus. The user may adjust view angles and scales, as well as toggle settings for axes, color, or perspective. The drawing package can also return to the command-line for more data input.

The $\boxed{\text{Generator}}$ routine performs the *Monte Carlo* simulation, producing particles from a pre-defined list of intitial conditions, and "swimming" the tracks through the apparatus. Particles intersecting detector objects produce hits which are sent to standard $\boxed{\text{Output}}$ for subsequent analysis. At each track step, $\boxed{\text{Physics}}$ routines apply physical processes (e.g. energy loss, decay) to tracks, updating the current step or producing new particles. Finally, the $\boxed{\text{Utilities}}$ routine contains various functions used by the other routines, such as vector rotations, line-object intersection, and random number generation.

2 The $\boxed{\text{Command}}$ Routine

2.1 Data input commands

Single-letter commands, followed by a series of parameters, control all input of data to the simulator. For example, to create colors for particles and objects in the $\boxed{\text{Display}}$ routine, the "c" command takes the form:

```
c <num> <red> <green> <blue>
```

where the (RGB) input parameters run from $(0.0, 1.0)$. The color number (num) references the color, with 0 reserved for "invisible" objects and particles. Blank spaces, commas, or tabs may delimit the parameters.

The "p" command defines particles with a number, name, and color, as well as charge, mass (in GeV/c^2), and lifetime ($c\tau_0$ in $meters$): [1]

```
p <num> <name> <color> <charge> <mass> <cτ0> <decay>
  ...<mode> <branch>
```

For decaying particles with non-zero "decay" mode, additional lines provide the final state particles for each decay process, along with the branching ratio (or decay fraction) for that mode.

Each object within the apparatus contains a single material defined with the "m" command:

```
m <num> <name> <A> <Z> <ρ> <X0>
```

Along with the name and number, "m" assigns the material properties of atomic mass (A) and number (Z), density (ρ, in g/cm^3), and radiation length (X_0, in cm). The $zero^{th}$ material denotes a vacuum, or empty object.

The *universe*, or $zero^{th}$ object, sets the spatial extents of the apparatus. The "u" command defines the universe, a rectangular object, with given color and material:

```
u <color> <mat> <x(min,max)> <y(min,max)> <z(min,max)>
```

To describe other objects within the universe, the "o" command provides the object number, name, shape, color, and material:

```
o <num> <name> <shape> <col> <mat> <det> <B_typ> <B_mag>
  ...<pos(x,y,z)> <siz(1,2,3)> <rot(α,β,γ)>
```

A detector flag (0 or 1) defines whether the object produces hit output for the tracks. Two parameters denoting type and shape describe magnetic field B within the object. Finally, "o" inputs the position, size, and rotation angles of the object with respect to the universe (or lab) frame. To handle the possibility of objects "within" other objects, numbering follows a hierarchy, with higher object numbers inside lower numbered objects (hence, the "universe" must equal object 0). This aides in stepping particles through the apparatus, as discussed in section 5 below.

An additional input command "g" sets the initial conditions for particles created by the $\boxed{\text{Generator}}$ routine:

```
g <num> <prt> <flag> <pos(x,y,z)> <mom(x,y,z)> <Δp>
```

This command gives each generator a number and particle type, along with initial position and momentum vectors. A generator flag value greater than zero will force the particle to decay through that particular mode. If the flag equals -1, a decay is forced through a random mode (according to branching ratio), while a value of -2 suppresses decays. Also, by specifying a non-zero Δp, the generator will randomly vary the magnitude of the particle's initial momentum.

[1] Unless otherwise noted, "ABC" uses units of $meters$ for length and time ($c\tau$), Tesla for B-fields, GeV for mass, momentum, and energy (with $c \equiv 1$), and $degrees$ for angles. Exceptions include materials units, which follow those typically found in tables (g, cm).

2.2 Script files

Because of the tedious nature of these command-line inputs, and to avoid repeating common entries, the "i" command inputs a list of instructions from a script file:

```
i <filename>
```

where *filename* has the file suffix $*.s$ assummed. Script file commands share the same format with command-line entries. The routine ignores blank lines, as well as anything after the pound sign "#" (the comment symbol).

Upon startup, Command automatically reads the default script file *abc.s*, which contains common color, particle, and material definitions. Using "i", this script may in turn call other scripts, up to three files deep, which define detector apparatus or additional particles and materials needed. By default, *abc.s* calls the script *example.s*, which provides a template for users to create their own experiments, and also allows the user to immediately run "ABC" after retrieving and compiling the code (see appendix A).

2.3 Control commands

From Command, the user can list current definitions for colors, generators, materials, objects, or particles with the "l" command:

```
l <c,g,m,o,p>
```

Similarly, "z" zeroes the current definitions for selected data:

```
z <c,g,m,o,p>
```

if, for example, the user wants to input a new apparatus from different script files.

To produce simulated particles for the Generator routine, the command-line accepts an integer number denoting the generator type, followed by an optional number of multiples. Hit information for the particles appears on standard output as particles enter detector objects during generation. The "f" command controls the format of this output:

```
f <0,1,2>
```

where 0 deactivates output, 1 produces single-line data for input to plotting packages (the default), and 2 formats the hit data for easier reading from the screen. With format type 1, the user should redirect the standard output to the desired file upon starting the simulator:

abc $>$ *filename.dat*

Command-line prompts, lists, and warnings for "ABC" appear on standard error.

The Command routine also controls flags and other simulator constants that might need adjustment for particular apparatus. The "s" command shows the current settings for variables. To toggle the process flags for particle decay, energy loss, magnetic fields, and multiple scattering, simply enter the name for the tag:

```
dectag
lostag
magtag
mlttag
```

Similarly, adjust constants for the simulator by entering the variable name, followed by the desired value. For example:

```
maxstp 0.05
```

resets the maximum step length for tracks to $5cm$. The main routine for "ABC" (*abc.c*) sets default values for these parameters, and defines the input data arrays for the simulator (see appendix B).

Entering "?" at the command-line produces the following list of possible commands:

```
Commands: (type '?' for help, 'q' to quit)

     # - generate particle # [n times]
     c - define color
     d - display
     f - set output format
     g - define generator
     i - input script file
     l - list data
     m - define material
     o - define object
     p - define particle
     s - show parameters
     u - define universe
     z - zero data
```

3 The $\boxed{\text{Display}}$ Routine

3.1 Strategy

The drawing package uses the *OpenGL Utility Toolkit* (GLUT) library to create a single *X-Window* diplaying the detector apparatus and simulated particles. $\boxed{\text{Display}}$ renders the 3-dimensional lab objects to the screen through a series of vector rotations. First, the routine transforms object frame points (such as corners of boxes) to the lab frame, and then rotates these lab frame coordinates to the desired view frame, represented by three Euler angles. Before plotting the (x, y) view frame coordinates, the routine applies an optional perspective to the objects through the view frame z-component and a pre-selected view distance:

$$x_v{}' = x_v \, \frac{d_v}{d_v - z_v} \ , \ y_v{}' = y_v \, \frac{d_v}{d_v - z_v}$$

In this way, all of the actual drawing commands reduce to connecting two-dimensional points within the graphics window, for instance:

```
glBegin(GL_LINES);
   glVertex2d(x1,y1); glVertex2d(x2,y2);
   glVertex2d(x3,y3); glVertex2d(x4,y4);
glEND();
```

While this method does not take advantage of the 3-dimensional capabilities of *OpenGL*, $\boxed{\text{Display}}$ maintains the modularity of the graphics calls for the future addition of different packages (for example, *TekTronix* screens, or direct creation of pen-plotter and *PostScript* output).

3.2 Objects

$\boxed{\text{Display}}$ currently defines two “shapes” of objects: boxes and cylinders. The “o” command defines boxes (*shape* = *box*) centered at (x, y, z) lab frame coordinates, with the z-axis specified by the three Euler angles (α, β, γ) and dimensions:

1) = x-width

2) = y-width

3) = z-width

Similarly, cylinders (*shape* = *cyl*) have an axis of symmetry along the object frame z-axis, with dimensions:

1) = inner diameter

2) = outer diameter

3) = height

Each object requires three separate functions, which control the presence of the object during simulation:

i) drawing function

ii) point location (is point inside or outside?)

iii) line intersection on object

The drawing function (e.g. `drw_box`), located within $\boxed{\text{Display}}$, performs the rotations and graphics output described above. Two utility functions (see section 5) then determine whether particle tracks fall within the object and where their trajectories intersect the object boundaries. Figure 2 shows a view (from $\alpha = 46°, \beta = 45°, \gamma = 90°$) of boxes and cylinders at four different rotations in the lab frame, along with the lab frame (x, y, z)-axes. Two generated particles pass through the series of objects, producing a trail of hits (shown as crosses).

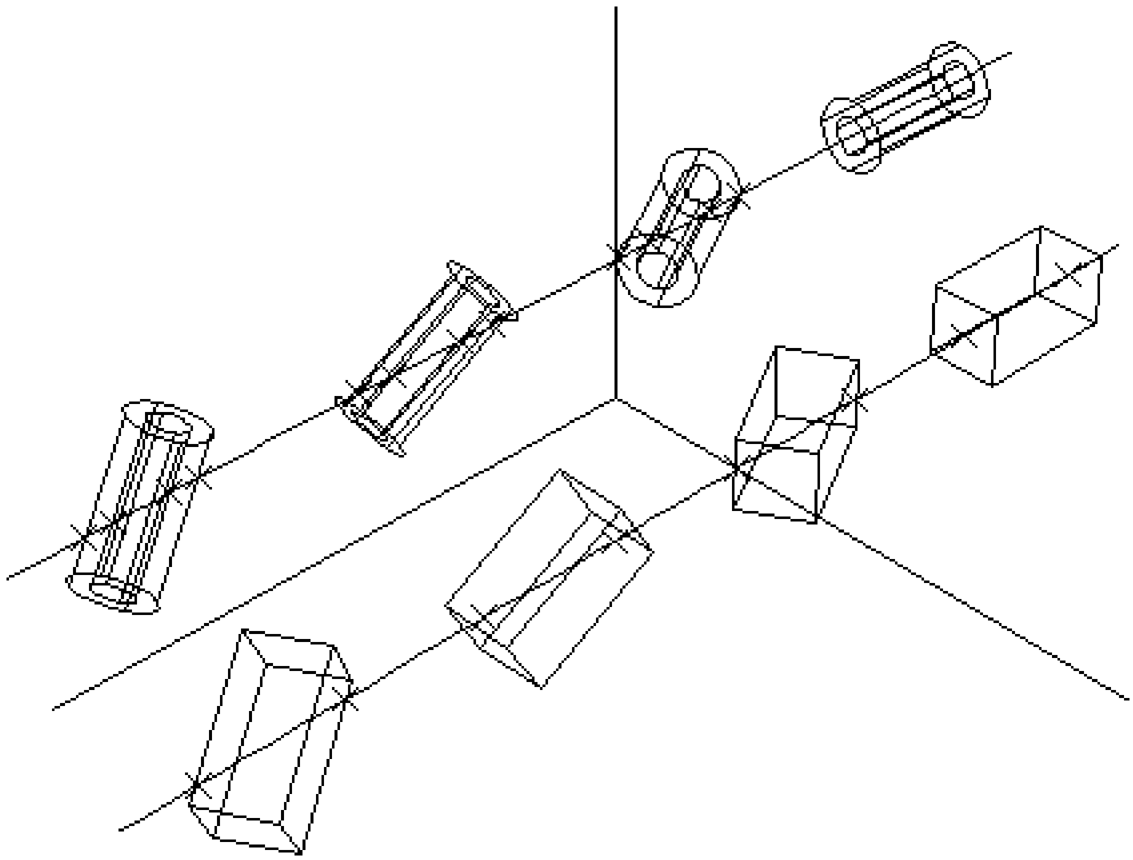


Figure 2: Example display of box and cylinder objects

3.3 Viewing commands

From the $\boxed{\text{Display}}$ routine window, single-letter commands control both the desired view parameters and the generation of new particles. Entering "?" provides a list (to the command-line terminal) of display package options:

```
Commands: (type '?' for help, 'q' to quit, <ESC> for prompt)

      # - generate particle # ('0'=10)
    i,o - zoom in with mouse (down on UL corner,up on LR), zoom out
    l,t - toggle tracks (lines/points),clear tracks/hits
    m,n - toggle background (B/W), color
u,j,h,p - toggle universe,axes,hits,perspective
v,x,y,z - view from angle,x-,y-,z-axis
      r - reset view

a/A,b/B,c/C - dec/inc view angle a,b,c
d/D,s/S,w/W - dec/inc view distance,scale,window

       <arrow> - move view left/right/up/down
<SHIFT><arrow> - rotate x-y view angle
```

For example, the number "3" activates the pre-defined generator defined as 3. Entering "a" or "A" will decrement or increment the view frame Euler angle α. To zoom in on a particular window area, enter "i" and then, using the mouse, click down on the upper-left corner and release on the lower-right corner of the desired region, or enter "o" to zoom back out to the previous setting.

The routine depicts particle trajectories as continuous lines, or toggling "l", as step-by-step points. Optionally, the particle hits for each detector may appear. The user can also control the drawing of lab frame axes or the rendering of the universe object.

4 The $\boxed{\text{Generator}}$ Routine

4.1 Overview

The $\boxed{\text{Generator}}$ represents the heart of the *Monte Carlo* simulation. From the initial data of particle type, position, and momentum via the "g" command, the routine "swims" the particle through the apparatus in a step-by-step fashion, applying various physical processes at each step. As described in the $\boxed{\text{Command}}$ section, a non-zero generator flag will force the particles to decay after the first step, while a non-zero Δp value causes a random deviation in initial particle momentum, thus simulating a beam of particles with a spread in energy.

A single step consists of an initial lab frame point, with a direction given by the particle's current momentum vector. The generator selects the length of the step starting at a maximum value specified with the *maxstp* parameter. The routine then decrements this step length depending on the radiation length X_0 of the material and the radius of curvature from magnetic fields (see section 6 below). If these lengths, times the variable increments (*incrad*, *incmag*, respectively), fall below *maxstp*, then the routine resets the current step length. To ensure that this "trial step" will span only a single material, with it's various physical effects, the routine checks for the crossing of object boundaries using the object location and intersection utilities, and truncates the step at the object border, accordingly (see section 5).

After determining this "trial" step point from the step length and direction, $\boxed{\text{Generator}}$ applies physics to the particle to form the final step point, which may or may not alter the particle's position, momentum, and energy. The routine stores the step values in 4-vector form, with the 4-position and 4-momentum having the notation [2, (11.16) and (11.54)]:

$$\mathbf{x} = (ct, x, y, z) \ , \ \mathbf{p} = (E/c, p_x, p_y, p_z)$$

where the final step time (ct_f) derives from the initial step time, step length (dx), and the particle's velocity [3, (34.1)]:

$$ct_f = ct_i + dx/\beta \quad \Rightarrow \quad \beta \equiv v/c = p/E$$

Upon entrying *and* exiting a detector object, the $\boxed{\text{Generator}}$ dumps this particle 4-position and 4-momentum to standard output, along with the generator, particle, and detector number.

One exception to the above procedure occurs when stepping across object boundaries. When the "trial" step exits the current object or enters a new one, the routine suppresses any deviations to the final step position (but not momentum!) from the physics processes, to ensure that the particles cross object boundaries in a clean fashion. This approximation introduces possible error in the stepper proportional to the step length, but greatly simplifies the object intersection utility function (see section 5).

4.2 Outline

To apply the above strategy for simulating particles, the outline for the $\boxed{\text{Generator}}$ routine becomes:

- Initialize first step with input position/momentum. If Δp selected, vary momentum magnitude with normal deviate. If generator flag set, force appropriate decay.
- Loop over tracks from current to total number. Swim tracks until momentum falls below threshold, or decay occurs:
  - Fill initial step data, determine object location
  - Choose step size from object field strength and radiation length X_0
  - Take trial step, checking for intersection with new object or exit of current one, and truncate
  - Apply physics processes to final step point:

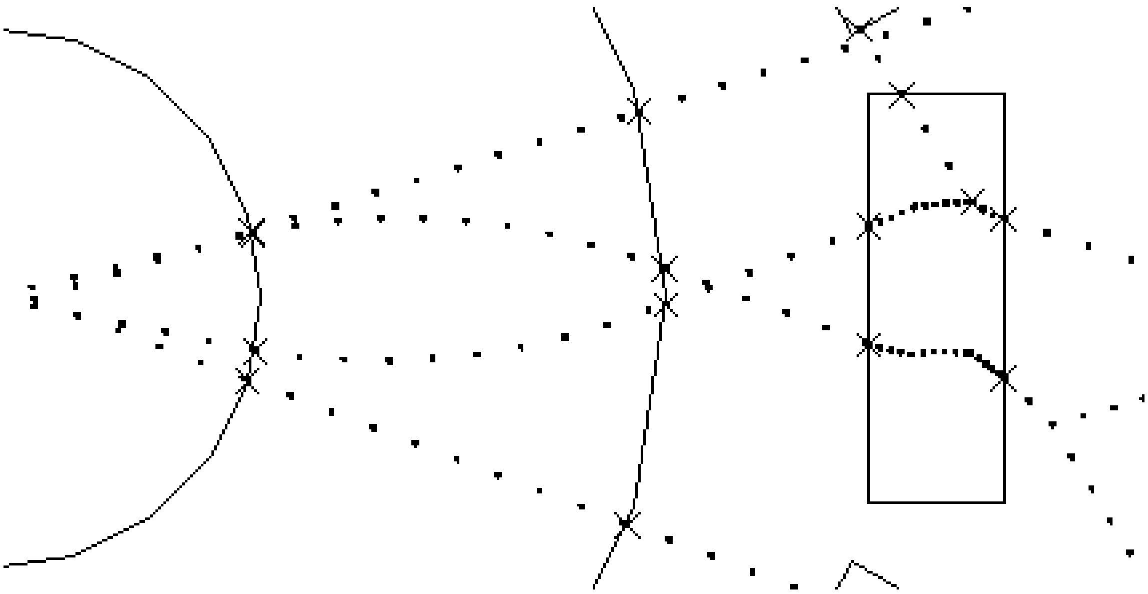


Figure 3: Example generator event showing the decay $K_L \rightarrow \pi^+\pi^-\pi^0$

  * curvature in magnetic field
  * multiple scattering
  * energy loss
  * particle decay (add new tracks to queue)
- Output position/momentum to *stdout* if entering or leaving detector object and fill hit array for $\boxed{\text{Display}}$

Figure 3 shows an example generator event, forcing a K_L meson decay at the origin of a cylindrical detector with outerlying aluminum slabs. The track setting from $\boxed{\text{Display}}$ depicts each particle step as the pions emerge from the origin (here, the K_L has been given some initial momentum along the x-axis). This particular decay creates three pions, one of which (the π^0), immediately decays into two photons, which travel straight through the magnetic field of the cylindrical object and leave the apparatus. The charged pions bend within the field, and enter the aluminum, where $\boxed{\text{Generator}}$ reduces the step length to accomodate the higher density material. After scattering within the slabs, the π^+ and π^- eventually decay into $\mu\nu$ pairs. As each particle crosses the object boundaries (which have all been defined as detectors), they leave "hits" denoted with a cross.

5 The $\boxed{\text{Utility}}$ Routines

5.1 Vector manipulation

"ABC" defines object frame coordinates with respect to the lab frame by a single translation to the object origin (x_o, y_o, z_o), followed by successive rotations about the object axes given by three *Euler* angles (α, β, γ). These rotations form a triple matrix product according to [4, (4.87)]:

$$\mathbf{A}(\alpha,\beta,\gamma) = \begin{pmatrix} +\cos\gamma\cos\beta\cos\alpha - \sin\gamma\sin\alpha & +\cos\gamma\cos\beta\sin\alpha + \sin\gamma\cos\alpha & -\cos\gamma\sin\beta \\ -\sin\gamma\cos\beta\cos\alpha - \cos\gamma\sin\alpha & -\sin\gamma\cos\beta\sin\alpha + \cos\gamma\cos\alpha & +\sin\gamma\sin\beta \\ +\sin\beta\cos\alpha & +\sin\beta\sin\alpha & +\cos\beta \end{pmatrix}$$

The utility function `uti_rot` applies this procedure to lab coordinates, first subtracting the object origin and then multiplying by the rotation matrix $\mathbf{A}$. The reverse of this procedure transforms object values back to the lab frame.

Similarly, a 4-vector $(x_0, \mathbf{x})$ in a reference from K transforms to a frame K' (moving with velocity β with respect to K) through matix multiplication. Using vector notation, this *Lorentz* transformation takes the form [2, (11.19)]:

$$\begin{aligned} x_0' &= \gamma\,(x_0 - \beta\cdot\mathbf{x}) \\ \mathbf{x}' &= \mathbf{x} + \frac{(\gamma-1)}{\beta^2}(\beta\cdot\mathbf{x})\,\beta - \gamma\beta x_0 \end{aligned}$$

The utility function `uti_bst` applies this *Lorentz* boost to 4-vectors, transforming from $(K \to K')$ or $(K' \to K)$.

5.2 Object functions

As stated in section 3, two utility functions operate on objects. The first routine, `uti_loc`, determines whether a given lab frame point falls inside an object, outside, or on the object boundary within the specified tolerance `mintol`. The procedure becomes straightforward after transforming the lab coordinates to the object frame, as outlined above. The routine compares the object frame coordinates with object dimensions, returning $(-1, 0, +1)$ if the point lies inside, on the border, or outside the object. The $\boxed{\text{Generator}}$ routine determines the current object of a given step point by looping through all defined objects, finding the highest number where `uti_loc` < 1.

To truncate trial generator steps at object boundaries, the utility function `uti_int` determines the intersection point on an object for a line, defined by two lab frame points. To maintain generality for any step size, the routine must decide between several possible solutions of the line on the object border, depending on the shape, since the step may enter, exit, or span a given object. For example, a *box* object has six sides, and therefore six possible solutions. After transforming the line to the object frame, `uti_int` reduces this set to three (x, y, z)-axis sides, by taking into account the locations of the endpoints, and whether the step enters, exits, or spans the object. To accomplish this, the $\boxed{\text{Generator}}$ passes `uti_int` a "preferred" point, initial or final, nearest the desired solution. The routine then calculates the three (x, y, z) solutions, picking the one with coordinates constrained by the object dimensions, and nearest the preferred step point. For example, defining the line equation with the direction cosines:

$$\begin{gathered} \Delta x = x_f - x_i, \quad \Delta y = y_f - y_i, \quad \Delta z = z_f - z_i \\ c_x = \Delta x/d, \quad c_y = \Delta y/d, \quad c_z = \Delta z/d \;\Rightarrow\; d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \end{gathered}$$

the possible solution on the x-border (with object x-width x_w) becomes:

$$\begin{aligned} x &= \pm\frac{x_w}{2} \\ y &= y_i + (x - x_i)\frac{c_y}{c_x} \\ z &= z_i + (x - x_i)\frac{c_z}{c_x} \end{aligned}$$

Cylindrical object line intersection follows a similar stategy, except that the presence of an inner diameter border complicates matters slightly. Through knowledge of the line endpoints and "preferred" point, the routine reduces the solution set to the z-border (as with boxes), and two (x, y)-border solutions found by solving the quadratic equation for the line through a circular surface of radius r:

$$x = -\frac{b}{2} \pm \frac{\sqrt{b^2 - 4c}}{2} \quad \Rightarrow \quad a = y_i - x_i \frac{c_y}{c_x}\,,\; b = \frac{2a(c_y/c_x)}{(c_y/c_x)^2 + 1}\,,\; c = \frac{a^2 - r^2}{(c_y/c_x)^2 + 1}$$

The routine then takes the *real* solution nearest the preferred point.

5.3 Random variable generation

Monte Carlo programs simulate random events through the use of random number generators. Actually, "ABC" requires two types of random numbers: uniform and normal. Uniform deviates (having an equal probability across a specified range) apply, for example, when choosing a random angle within a plane from $(0°, 360°)$. "ABC" uses the C $< math.h >$ funtion:

```
double drand48()
```

to generate uniform random numbers from $(0, 1)$, and then rescales this deviate to the appropriate range. Figure 4 shows `drand48()` values for $10k$ events. For the range $(0, 1)$, a line fit to the data gives a χ^2 over the number of degrees of freedom (ndf) very close to unity, as expected. Since the selection of particle decay modes uses this uniform variable, the high-end of the distribution becomes important for very rare modes. A second plot in Figure 4 for the range $(0.999, 1)$ also provides a fairly uniform distribution.

Statistical effects, such as particle scattering angles or measurement errors, follow a normal (or *Gaussian*) distribution, given by [5, def (4.7)]:

$$f(y) \;=\; \frac{e^{-(y-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

where μ and σ equal the distribution mean and standard deviation. "ABC" generates normal random numbers from two uniform deviates (r_1, r_2) using the *Box-Muller* method [6, (7.2.10)], which applies the fundamental transformation law of probabilities:

$$r_n \;=\; \mu + \sigma\, \sin(2\pi r_1)\sqrt{-2\ln r_2}$$

Figure 4 shows two examples of normal distributions for different values of (μ, σ), with corresponding *Gaussian* fits.

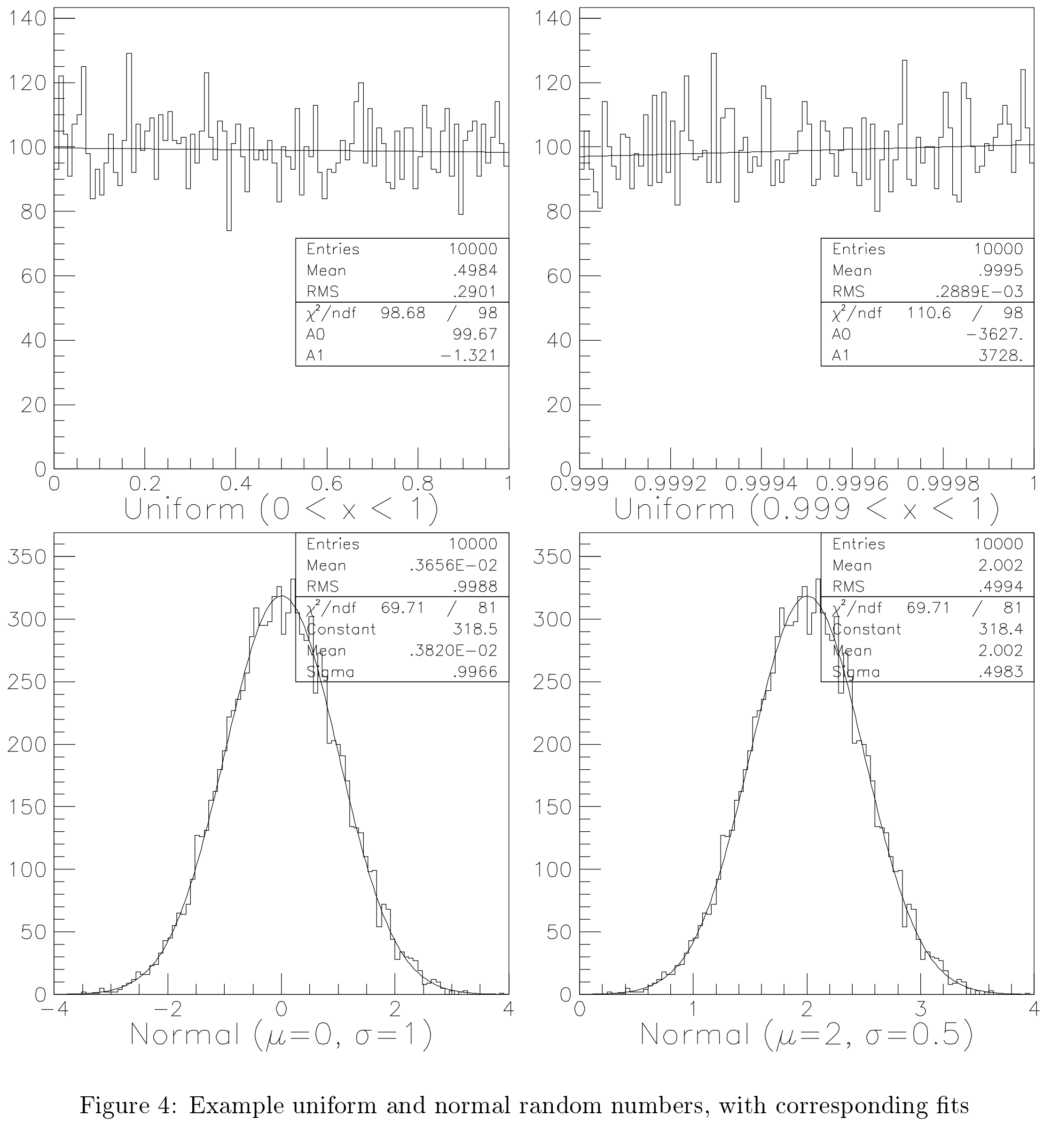


Figure 4: Example uniform and normal random numbers, with corresponding fits

6 The $\boxed{\text{Physics}}$ Routines

6.1 Overview

The $\boxed{\text{Physics}}$ package consists of individual subroutines which apply independent physical processes to the particle 4-position and 4-momentum at each generator step. Because of their independence, a user can easily define additional physics processes, if needed, adding their call to the $\boxed{\text{Generator}}$ routine. The current list of physics routines includes:

- particle decay
- magnetic field effects
- energy loss
- multiple scattering

"ABC" makes simple assumptions and approximations for these processes to reduce the need for user input, and maintain the scope of this project. These assumptions, therefore, represent the greatest source of error for the simulation, and also the area for most improvement. While the particle decay routine applies generally to all particles, the three other processes act only on charged particles, so that no particle interactions exist for neutral particles such as photons, neutrinos, or neutrons. Additionally, the energy loss algorithm only concerns ionization losses for heavy charged particles. Thus, "ABC" poorly approximates the behavior of electrons, where radiative loss (i.e. *bremsstrahlung*) dominates for relativistic energies. Possible extensions to the $\boxed{\text{Physics}}$ package might therefore include:

- photon interactions
  - photoelectric effect
  - Compton scattering
  - pair-production ($\gamma \to e^+e^-$)
- energy loss for electrons
  - ionization (modified Bethe-Bloch)
  - bremsstrahlung
  - Čerenkov radiation
- higher-order interactions, such as δ-ray production or hadronic showers

6.2 Applying particle decay

Unstable particles decay with a certain mean lifetime τ_0, which becomes dilated in the lab frame for particles with relativistic energies ($\tau = \gamma\tau_0, where\, \gamma = E/m$). The survival probability of the particle over a step x follows the well-known exponential decay law, which statistically takes the form of a Poisson distribution (with $x = 0$) [5, def (3.11)]:

$$P(x) = \frac{\lambda^x}{x!}e^{-\lambda} = e^{-\Delta t/\tau} \;\Rightarrow\; \lambda = \Delta t/\tau \;,\; \Delta t = t_f - t_i$$

The decay routine `abc_dec` tests for a particle decay at each generator step by calculating this suvival probability and choosing a uniform deviate (r_p). If ($r_p > P(x)$), `abc_dec` generates a decay, picking a random decay mode by summing the mode branching ratios (which need not sum to unity) and picking a second uniform random number from within these probabilities, producing a random mode proportional to the decay fraction.

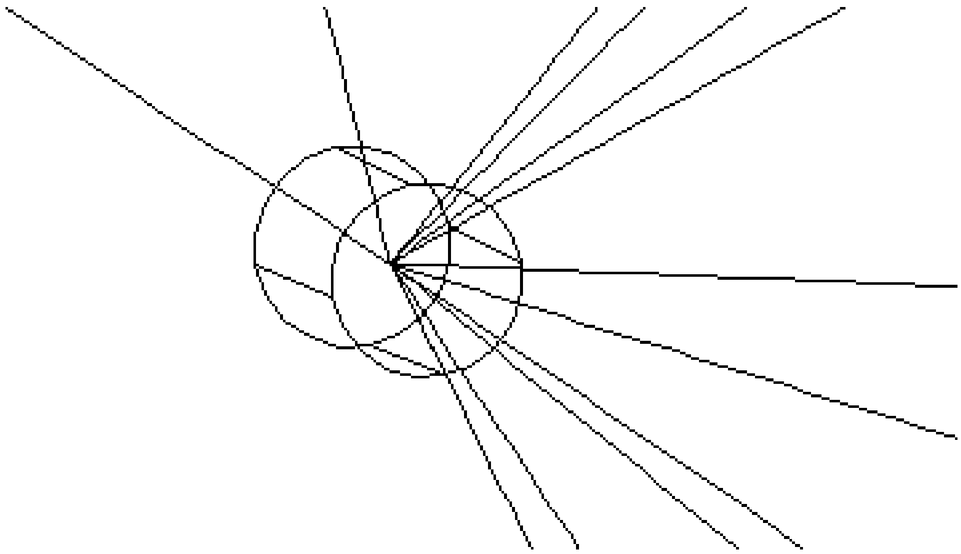


Figure 5: Testing particle decay with $(\mu \to e\nu\nu)$ and $(\pi \to \mu\nu)$

After selecting a decay mode, `abc_dec` must give each final state particle a random direction and momentum, whose probabilities derive from the partial decay rate for a mass m (in rest frame) into n bodies, a function of the matrix element $\mathcal{M}$ and phase-space $d\Phi_n$ [3, (34.10)]:

$$d\Gamma = \frac{2\pi^4}{2m} \mid \mathcal{M} \mid^2 d\Phi_n(P; p_1, p_2, \ldots, p_n)$$

Here, $\mathcal{M}$ describes the physics of the decay (including resonances or spin effects), while $d\Phi_n$ contains the kinematic constraints (conservation of energy and momentum). To generalize decays for all particles, `abc_dec` assumes a constant matrix element, producing decays uniform in phase-space. The routine then *Lorentz* boosts the final state particles back to the lab frame.

For two-body decays, applying conservation of 4-momentum pre-determines the final-state momenta, where:

$$\mid \mathbf{p_1} \mid = \mid \mathbf{p_2} \mid = \frac{[(m^2 - (m_1 + m_2)^2)(m^2 - (m_1 - m_2)^2)]^{1/2}}{2m}$$

The phase-space element then reduces to:

$$d\Gamma \sim \frac{\mid \mathbf{p_1} \mid}{m^2} d\Omega \;\Rightarrow\; d\Omega = d\phi_1 d(\cos\theta_1)$$

Thus, to choose a decay uniformly over the volume element $d\Omega$, `abc_dec` picks two uniform deviates, ϕ_1 from $(-\pi, \pi)$, and $\cos\theta_1$ from $(-1, 1)$.

Three-body decays form a unique production plane, where a *Dalitz* plot of any two final-state energies has a uniform distribution. Choosing two such energies E_i within the range $(m_i < E_i < (E - m_j - m_k))$, the phase-space becomes:

$$d\Gamma \sim dE_1\, dE_2\, d\Omega \;\Rightarrow\; d\Omega = d\alpha\, d(\cos\beta)\, d\gamma$$

Here, the Euler angles (α, β, γ) define the rotation of the production plane, which has a uniform orientation with respect to the center of mass frame. `abc_dec` therefore chooses three uniform deviates for $\alpha(-\pi, \pi)$, $\cos\beta(-1, 1)$, and $\gamma(-\pi, \pi)$.

The script file *tdec.s* (see Figure 5) tests the particle decay routine by creating a cylindrical detector, and generating $(\mu \to e\nu\nu)$ and $(\pi \to \mu\nu)$ decays from the origin. Plotting the survival probability for pions with momentum $0.1 GeV$, Figure 6 shows an exponential decay position with a measured slope of -0.1664, close to the expected result. The $\cos\theta$ distribution for electrons from the three-body muon decay (at rest) form a roughly uniform population across $(-1, 1)$. Finally, muons from pion decay, which become forward-constrained in the lab as pions exceed a momentum of $39 MeV$, have only a positive $\cos\theta$ distribution at $50 MeV$.

6.3 Applying magnetic fields

In uniform, static B-fields, charged particles follow helical paths with a radius of curvature [7, (2.1)]:

$$a \sim \frac{c\, p_\perp}{z\, B} \;\Rightarrow\; p_\perp \equiv \textit{perpendicular momentum}$$

The magnetic field routine `abc_mag` defines "dipole" fields for objects, with a uniform B along the z-axis. With this simple assumption, rotating step points to the object frame automatically separates the particle momentum into $p_\perp$ and $p_\parallel$ components. From the angle of curvature for step Δx ($a_c = \Delta x / a$) and the initial $p_\perp$ angle ($a_1 = \tan^{-1}(p_y/p_x)$), the object frame final step position and momentum become:

$$\begin{aligned}
x_2 &= x_1 + d\cos(a_1 - a_c/2) \;\Rightarrow\; d = 2a\sin(a_c/2) \\
y_2 &= y_1 + d\sin(a_1 - a_c/2) \\
p_{x2} &= p\cos(a_1 - a_c) \;\Rightarrow\; p = \sqrt{p_{x1}^2 + p_{y1}^2} \\
p_{y2} &= p\sin(a_1 - a_c)
\end{aligned}$$

The routine then rotates this corrected object frame point back to the lab. As stated above, `abc_mag` suppresses the positional correction when the step exits or enters a new object, to simplify the line intersection utility.

Figure 7 tests the magnetic field with the script file *tmag.s*. Protons with a spread in momentum enter two dipole boxes with equal and opposite fields, where they emerge separated, but parallel. A cylindrical magnet rotated upwards captures the protons, where all but a single high-momentum track follow spiral paths, emerging from the top of the object.

6.4 Applying particle energy loss

Charged particles heavier than electrons lose energy within a material primarily through inelastic collisions with atomic electrons, given by the Bethe-Bloch equation [3, (22.1)]:

$$-\frac{dE}{dx} = K\, z^2\, \frac{Z}{A}\, \frac{1}{\beta^2} \left[\frac{1}{2}\ln\frac{2m_e c^2\beta^2\gamma^2 T_{max}}{I^2} - \beta^2 - \frac{\delta}{2} - \frac{C}{Z}\right]$$

where the maximum kinetic energy transfer to electrons is:

$$T_{max} = \frac{2m_e c^2\beta^2\gamma^2}{1 + 2\gamma(m_e/m) + (m_e/m)^2}$$

To minimize the number of material parameters needed, the energy loss routine `abc_los` approximates the mean excitation potential for a medium with atomic number and mass (Z, A) by [8, (2.29)]:

$$I = \begin{cases} 12\, Z + 7 & (Z < 13) \\ 9.76\, Z + 58.8\, Z^{-0.19} & (Z \geq 13) \end{cases}$$

The Bethe-Bloch equation contains both low and high-energy corrections. The routine ignores the shell correction C, which applies when particles approach the velocity of the bound atomic electrons, since the $\boxed{\text{Generator}}$ sets a minimum particle momentum (*minmom* $\sim 10\, MeV$). Since charged particles tend to polarize the medium, atomic electrons far from the particle trajectory become shielded, reducing their effect on the energy loss. This *density* effect becomes important at

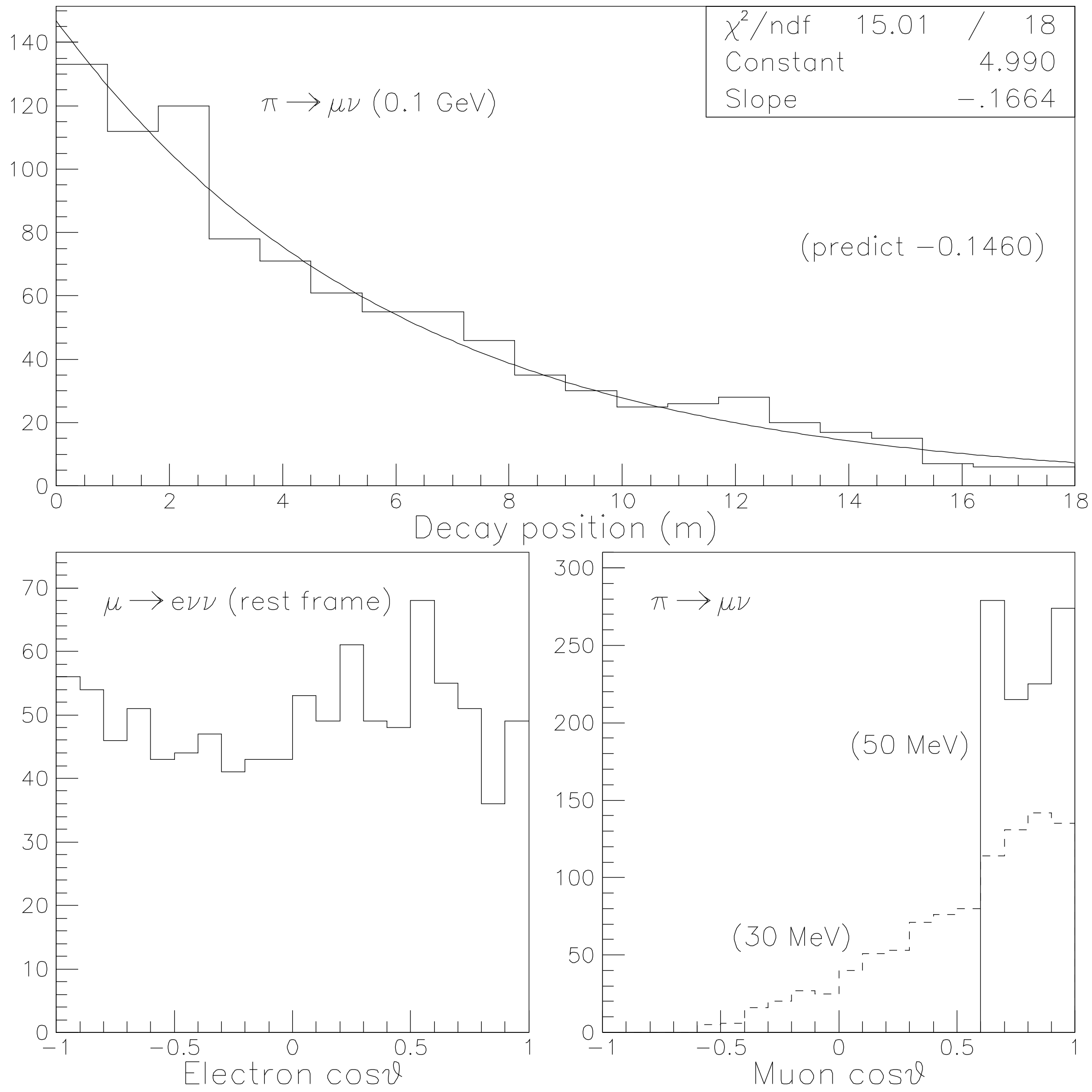


Figure 6: Decay position for $0.1GeV$ pions; electron and muon decay angles for $(\mu \rightarrow e\nu\nu)$ and $(\pi \rightarrow \mu\nu)$

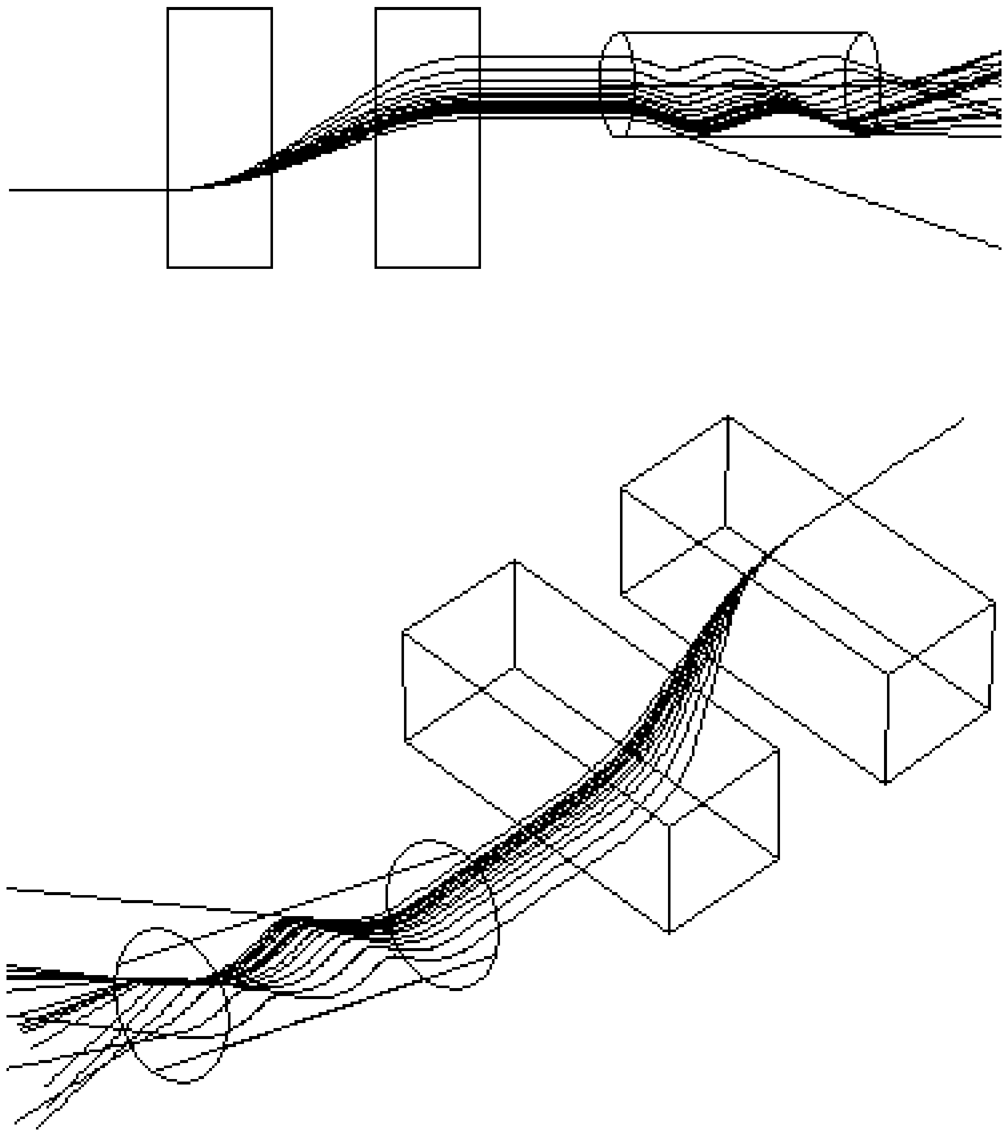


Figure 7: Testing the magnetic field routine with protons of varying momentum

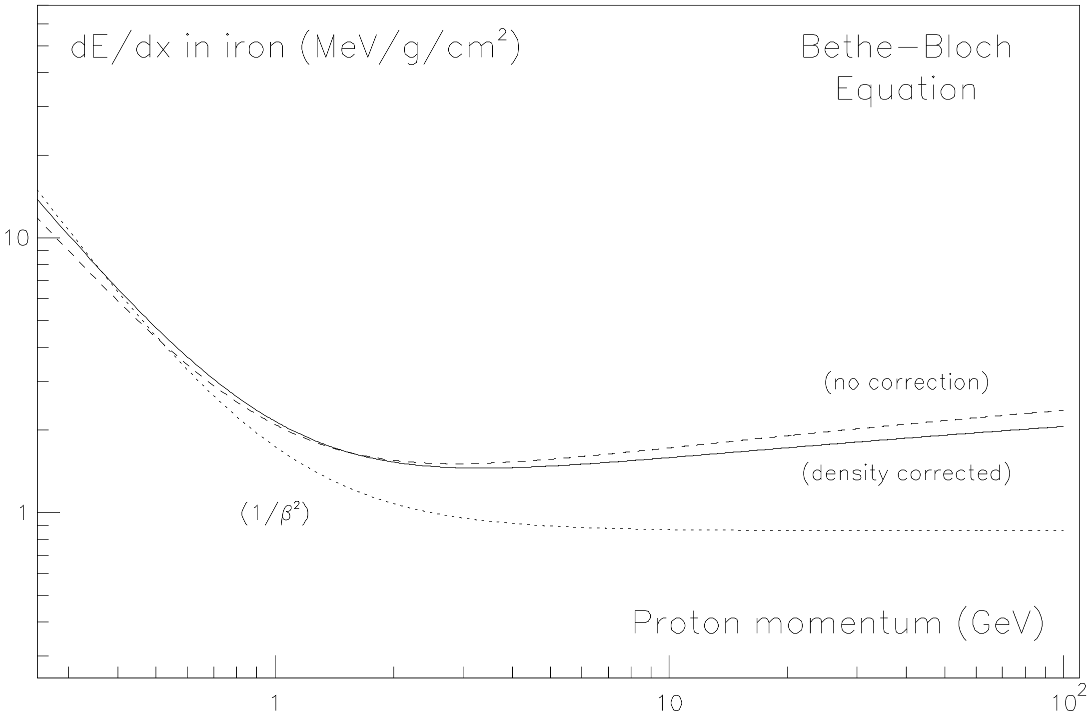


Figure 8: Bethe-Bloch approximation used by “ABC”

high energy, decreasing the logrithmic rise of the Bethe-Bloch formula. `abc_los` makes the following high-energy approximation to the density correction [3, (22.3)]:

$$\frac{\delta}{2} \sim \ln(\hbar\omega_p/I) + \ln\beta\gamma - 1/2 \;\Rightarrow\; \hbar\omega_p = 28.816\sqrt{\rho\, Z/A}$$

Figure 8 shows the resulting Bethe-Block values for protons in iron, with and without the density correction, along with a fitted $1/\beta^2$ dependence. As stated in [8, p.29], these approximations to the energy loss have an accuracy of a few percent for particle energies where ($\beta > 0.1$).

The `abc_los` routine determines the total energy loss for the step by integrating dE/dx across the step length. Since the dE/dx approximation itself has at least a few percent error, the integration does not represent the limiting factor in accuracy, so `abc_los` uses a fourth-order Runge-Kutta scheme [6, (16.1.3)], where the parameter `inclos` sets the increment size in fractions of step length. During the integration, the routine updates the particle momentum. If the momentum falls below threshold, `abc_los` truncates the step and ends the particle swimming. Testing this integration for muons in iron, Figure 9 plots the muon momentum versus particle position which, extrapolating to the x-axis, gives the particle range. Table 1 shows that this *Monte Carlo* range has an average error of about 5% of measured [3, p.133] values, as expected.

An additional error in the energy loss algorithm results from the exclusion of energy “straggling”. The actual particle energy loss at each step is statistical in nature, following a distribution modelled by the Landau and Vavilov theories [9, VII-2]. For high energies, this straggling varies the range of particles by about 5%.

6.5 Applying multiple Coulomb scattering

While charged particles lose energy as they collide with atomic electrons in passing through material, their direction of momentum remains fixed (assuming $m \gg m_e$). Conversely, particles suffer many small-angle *elastic* scatters with nuclei, where energy remains constant, but particle direction

Momentum(GeV)	*Range(m)*	*Expect(m)*	*Error(%)*
0.1	0.0128	0.0121	-5.8
0.5	0.3165	0.3370	+6.0
1	0.7320	0.7400	+1.0
5	3.6400	3.3800	-7.7
10	6.9200	6.7300	-2.8

Table 1: Comparison of simulated and measured muon range

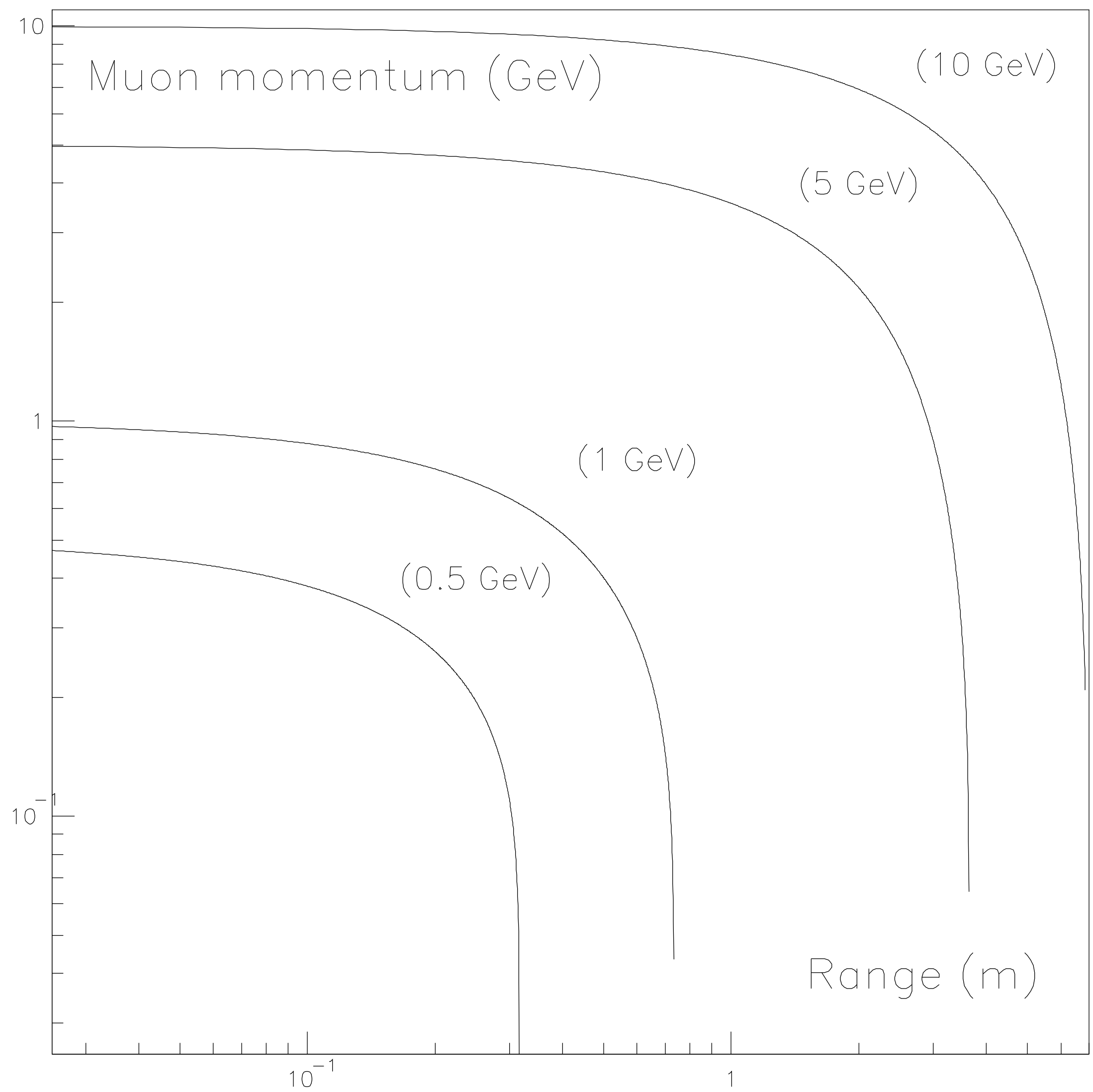


Figure 9: Muon momentum versus range, in iron

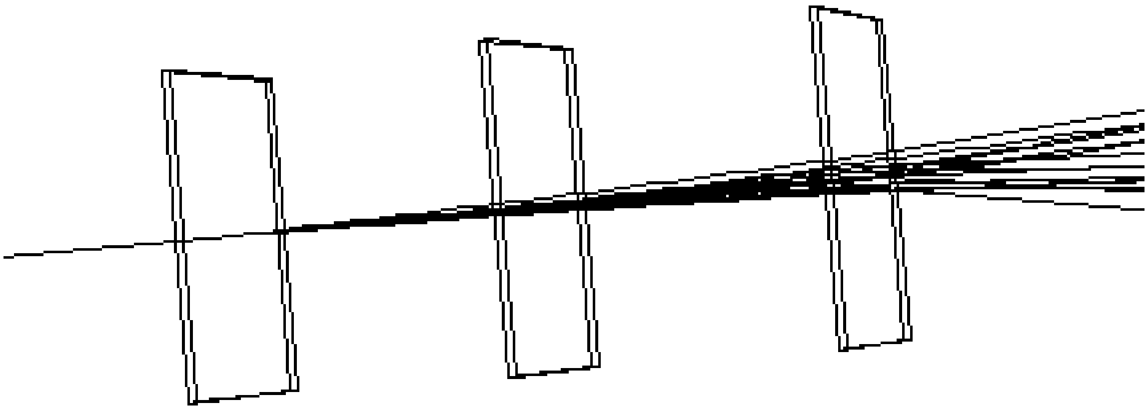


Figure 10: $0.5 GeV$ muons scattering through Al, Cu, and U slabs

changes. Physicists term this multiple Coulomb scattering, since most of the deflection occurs from Coulomb scattering with the nucleus. The multiple scattering routine `abc_mlt` uses the following approximation for the rms plane scattering angle (through pathlength x) [3, (22.8)]:

$$\theta_0 = \frac{13.6 MeV}{\beta\, c\, p}\, z\, \sqrt{x/X_0}\, \left[1 + 0.038\, \ln x/X_0\right]$$

The actual *Molière* distribution for the scattering angle contains complicated tails due to large angle scatters. The `abc_mlt` routine assumes that the scattering angle follows a Gaussian distribution, roughly valid for the central 98% of the curve, but providing an approximation no better than about 11% [3, p.134].

To simulate this Gaussian scattering, `abc_mlt` chooses two normal random variables from a distribution with ($\mu = 0, \sigma = \theta_0$), calculating a random space scattering angle ($\theta_s = \sqrt{\theta_1^2 + \theta_2^2}$) from the original particle direction. The routine then selects a random azimuthal angle ϕ_s about the particle direction, from $(-\pi, \pi)$, and rotates the final step momentum vector through (θ_s, ϕ_s).

Testing the multiple scattering routine for muons, Figure 10 shows muons incident on $5cm$ slabs of aluminum, copper, and uranium (defined with the script file *tmlt.s*). Figure 11 shows the muon scattering angle for the three materials at a muon momentum of 1.0 and $0.5 GeV$, so that cutting the momentum in half roughly doubles the mean scattering angle.

Two factors limit the accuracy of this algorithm using to the Gaussian approximation. First, the final step position remains fixed, with adjustment only to the momentum direction, for simplicity. Second, since the actual scattering distribution has larger tails than a Gaussian, successive steps of scatters do not correctly add in quadrature. The combined effect underestimates the true total scattering over the track length.

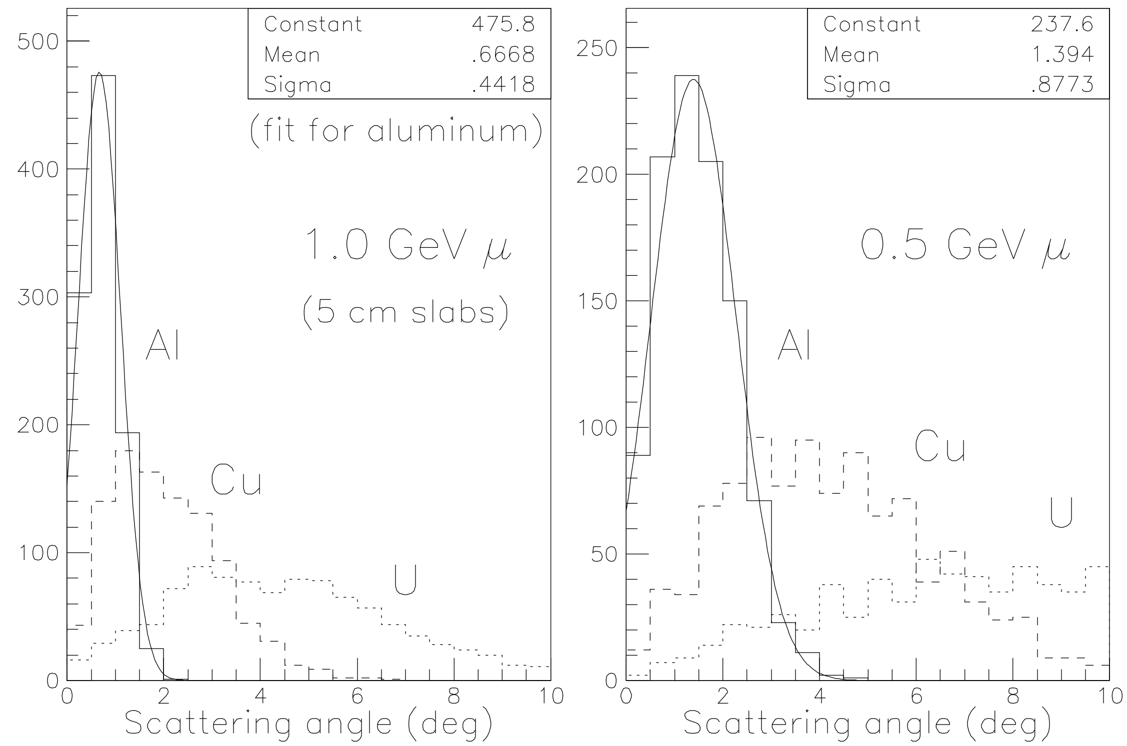


Figure 11: Testing multiple scattering for muons in Al, Cu, and U

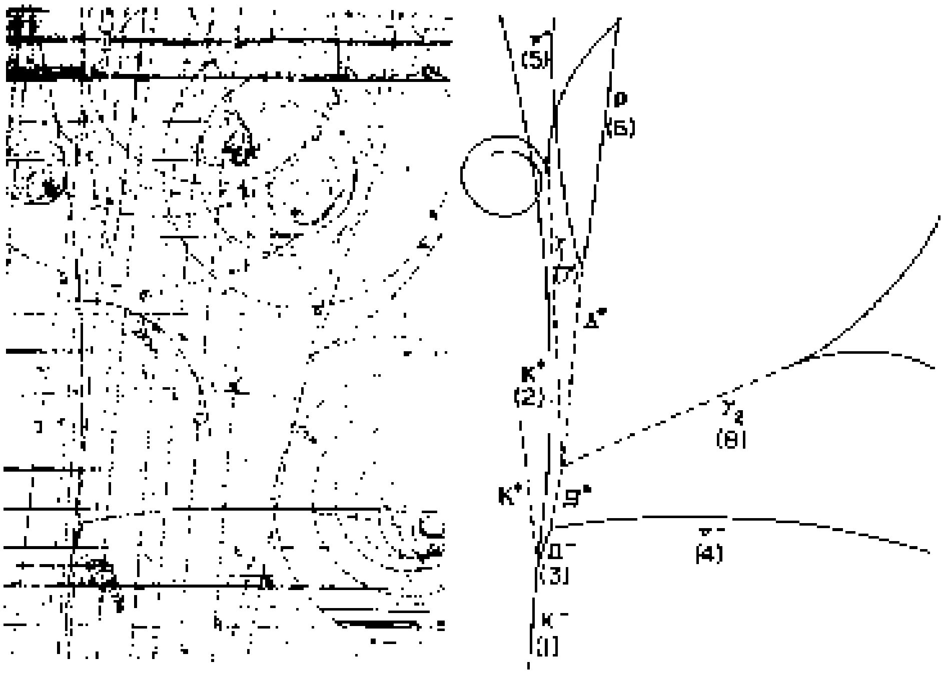


Figure 12: Discovery of the Ω^-, from Phys. Rev. Lett. **12**, 204 (1964)

7 "ABC" Examples

7.1 Creating an experiment

The file *example.s* provides a starting point for creating new experiments. This file defines a universe with $20m$ sides, and a cylindrical detector located at the origin, with a $0.2T$ field (see title page figure). Five vertical slabs of aluminum surround the cylinder at the positive (downstream) x-axis, while a thin slab of iron rests upstream. The *example.s* script (see appendix A.2) defines muon, pion, and proton generators starting from the origin, forcing decays for the μ and π. The script also defines a new particle, the "ABCino", which can decay into three pions, kaons, or sigmas. This illustrates the fact that simulator particles need not be real, nor particles! For example, to study the nuclear interaction ($np \to pp\pi^-$) for a neutron beam incident on liquid hydrogen (protons), one could create a "particle" with a mass equal to the center of mass of the np system, with a forced three-particle "decay" mode of $pp\pi^-$, and a generator momentum equal to that of the neutron beam.

7.2 Discovery of the Ω^-

Students might want to reproduce famous experiments that they read about in textbooks. For instance, the discovery of the Ω^- baryon appears in many books on particle physics, since Gell-Mann predicted its existence from the *Eight-fold Way* [10, p.36]. The experiment, performed at Brookhaven's AGS, reported this finding using the BNL $80in.$ hydrogen bubble chamber [11] (see Figure 12).

The script file *tomega.s* simulates this discovery by defining the bubble chamber as a *box*, containing LH_2 and a $0.4T$ field. The generator forces an Ω^- decay (assuming creation by $K^- + p \to \Omega^- + K^+ + K^0$), which produces the following chain of events in Figure 13:

$$\begin{aligned} \Omega^- \to \Xi^0\,\pi^- \ , \ \Xi^0 &\to \Lambda^0\,\pi^0 \\ \pi^0 &\to e^+\,e^-\,\gamma \\ \Lambda^0 &\to p\,\pi^- \end{aligned}$$

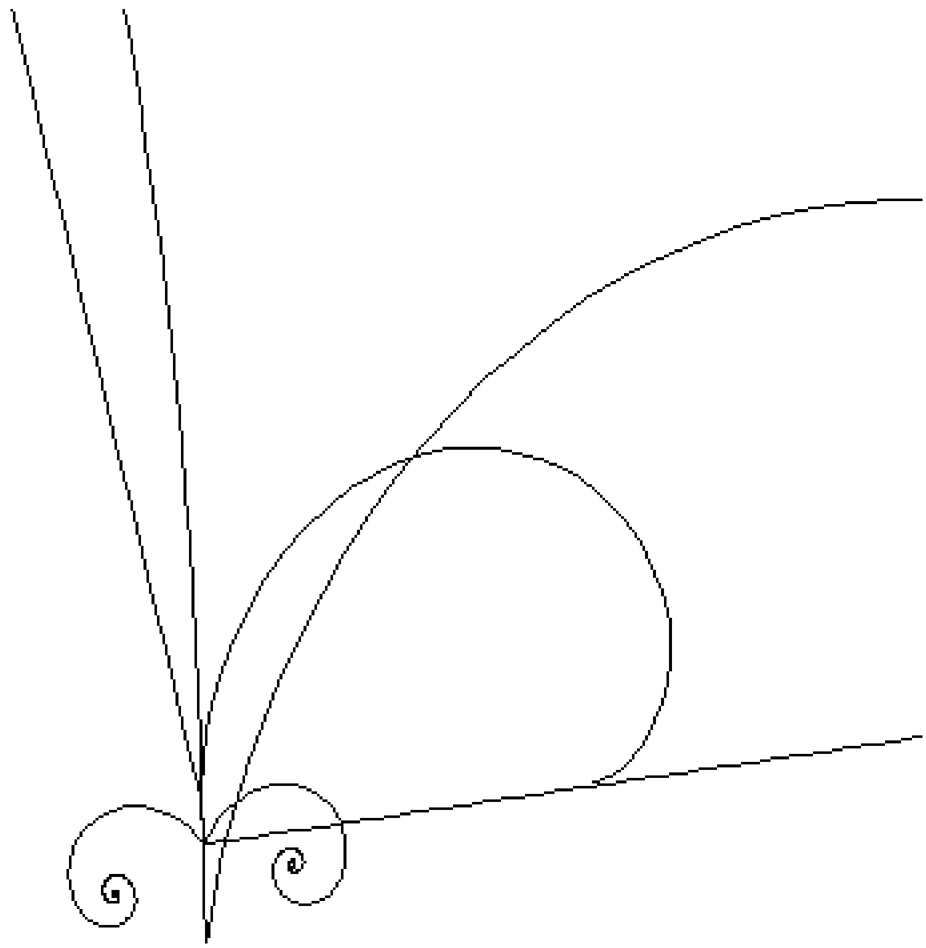


Figure 13: Example omega event

$$\pi^- \quad \rightarrow \quad \mu^- \, \nu$$

7.3 Detection of the η meson

To detect particular events of interest, experimentalists must distinguish these particles from the plethora of background events that exist in the typical experiment. "ABC" provides an easy means for studying these backgrounds for the apparatus in question. For example, to study η mesons through their two-body decay into $\gamma\gamma$, the script file *teta.s* defines a cylindrical *calorimeter*, which detects photons emerging from the origin (see Figure 14). One can "detect" the η by reconstructing the two-body decay mass from a single photon energy. However, the η also decays, with roughly equal probability, into three pions, and the π^0's themselves decay to $\gamma\gamma$, so many competing photons exist for the detector.

The experiment can remove photons from any decays containing charged pions by vetoing with pion detectors behind the calorimeter. That leaves the photons from the η-produced π^0's which, as Figure 15 shows, become conveniently separated from the $\gamma\gamma$ peak. However, if π^0's are directly created from the target in addition to η's, which seems likely, then misidentifying these photons as those from the η creates a roughly flat background which enters into the η peak. Thus, "ABC" warns of this potential contamination, from which the experimentalist may want to place a cut on the reconstructed π^0 mass.

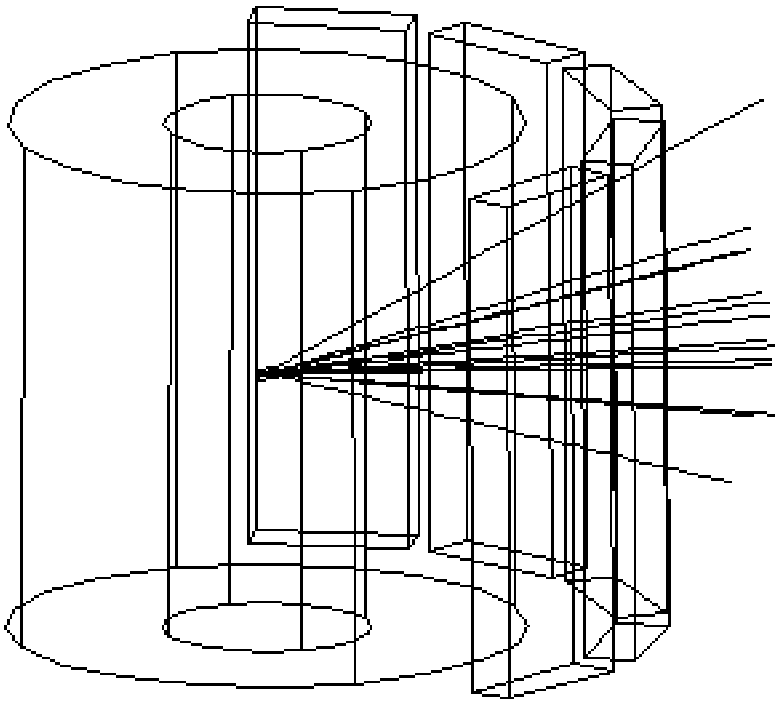


Figure 14: The η detector

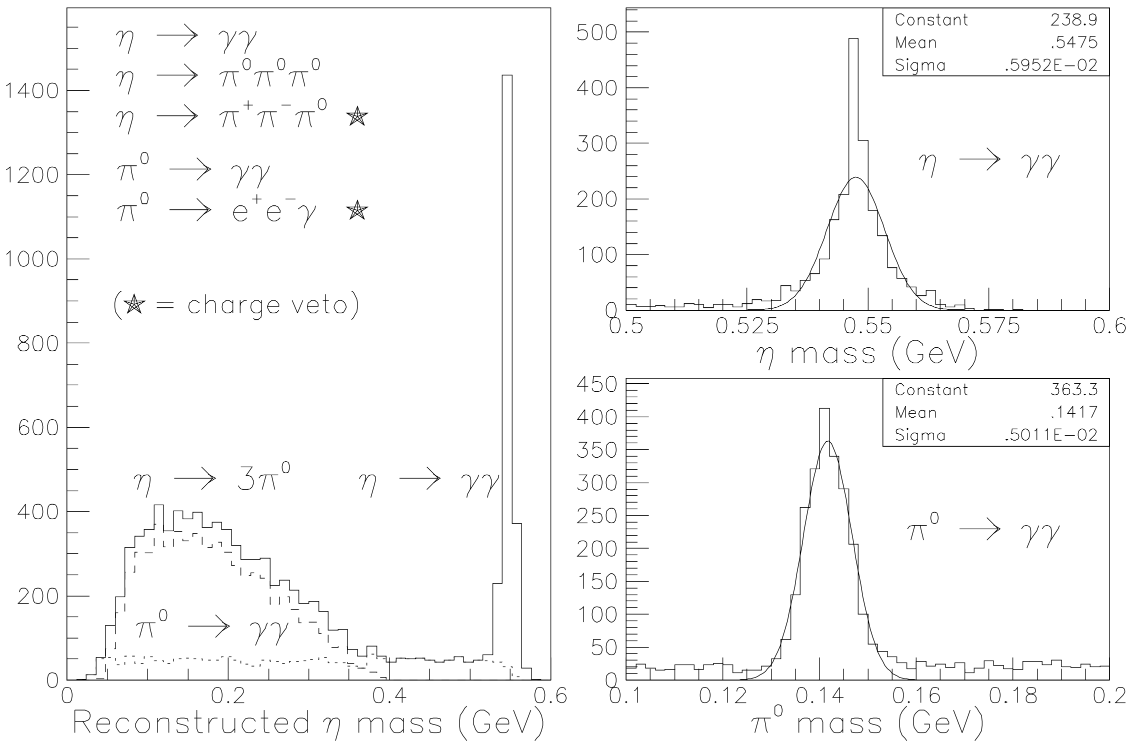


Figure 15: Plots showing the detection of the η from decay to $\gamma\gamma$, versus possible background events

A “ABC” Script Files

A.1 Default file *abc.s*

```
# Script file: abc.s
#
#    This is the default script file for the ABC simulator.  Command format
# is identical to command-line format.  Separate input parameters with blanks,
# commas, or tabs.  Everything after a '#' is ignored (comment symbol),as are
# blank lines.
#
#    This file is automatically read when starting 'abc' (unless deactivated
# with abctag=0).  Use it to define colors, particles, and materials common
# to all experiments.  Specific scripts for apparatus can be input from here
# (script files may call other scripts up to 3 deep).
#
#    The particle and material properties below were taken from the Particle
# Physics Booklet [Phys.Rev. D50,1173(1994)] and the GEANT 3.2.1 manual [CERN
# Program Library].

# Define colors: (color 0 pre-defined as black or white)
#   num  red   grn   blu
  c 01   1.0   0.0   0.0
  c 02   0.0   1.0   0.0
  c 03   0.0   0.0   1.0
  c 04   1.0   0.5   0.5
  c 05   0.5   1.0   0.5
  c 06   0.5   0.5   1.0
  c 07   1.0   0.5   0.0
  c 08   0.0   1.0   0.5
  c 09   0.5   0.0   1.0
  c 10   1.0   0.0   0.5
  c 11   0.5   1.0   0.0
  c 12   0.0   0.5   1.0
  c 13   1.0   0.0   0.1
  c 14   0.1   1.0   0.0
  c 15   0.0   0.1   1.0
  c 98   0.0   0.0   0.0 # black
  c 99   1.0   1.0   1.0 # white

# Define particles:
#   num/name     col  chrg  mass(GeV)  life(m)  decay
  p 01 photon    01   +0.0  0.0        0.0      0
  p 02 neutrino  02   +0.0  0.0        0.0      0
  p 03 electron  03   -1.0  0.000511   0.0      0
  p 04 positron  04   +1.0  0.000511   0.0      0
  p 05 muon-     05   -1.0  0.105658   658.654  1
       030202 1.0                               # e- nu nu
  p 06 muon+     06   +1.0  0.105658   658.654  1
       040202 1.0                               # e+ nu nu
  p 07 pion-     07   -1.0  0.139568   7.804    1
       000502 1.0                               # mu- nu
  p 08 pion+     08   +1.0  0.139568   7.804    1
       000602 1.0                               # mu+ nu
```

```
  p 09 pion0     09   +0.0  0.134974   25.1e-9  2
       000101 0.988                             # gamma gamma
       010304 0.012                             # gamma e- e+
  p 10 proton    10   +1.0  0.938272   0.0      0
  p 11 neutron   11   +0.0  0.939566   2.66e+11 1
       020310 1.0                               # nu e- p+ (beta decay)
  p 12 kaon-     12   -1.0  0.493646   3.713    6
       000502 0.6351                            # mu- nu
       000709 0.2117                            # pi- pi0
       070807 0.0559                            # pi- pi+ pi-
       030209 0.0482                            # e- nu pi0
       050209 0.0318                            # mu- nu pi0
       070909 0.0173                            # pi- pi0 pi0
  p 13 kaon+     13   +1.0  0.493646   3.713    6
       000602 0.6351                            # mu+ nu
       000809 0.2117                            # pi+ pi0
       080807 0.0559                            # pi+ pi+ pi-
       040209 0.0482                            # e+ nu pi0
       060209 0.0318                            # mu+ nu pi0
       080909 0.0173                            # pi+ pi0 pi0
  p 14 kaonL     14   +0.0  0.497671   15.51    7
       070402 0.1935                            # pi- e+ nu
       080302 0.1935                            # pi+ e- nu
       070802 0.1355                            # pi- mu+ nu
       080502 0.1555                            # pi+ mu- nu
       090909 0.2150                            # pi0 pi0 pi0
       080709 0.1239                            # pi+ pi- pi0
       000605 7.2e-9                            # mu+ mu-
  p 15 kaonS     15   +0.0  0.497671   0.0268   2
       000708 0.6861                            # pi- pi+
       000909 0.3139                            # pi0 pi0
  p 16 lambda    01   +0.0  1.115684   0.0789   2
       001007 0.6420                            # p pi-
       001109 0.0358                            # n pi0
  p 17 sigma-    02   -1.0  1.119744   0.0443   1
       001107 1.0                               # n pi-
  p 18 sigma+    03   +1.0  1.118937   0.0240   2
       001009 0.5164                            # p pi0
       001108 0.4836                            # n pi+
  p 19 sigma0    04   +0.0  1.119255   2.2e-11  1
       001601 1.0                               # lambda gamma
  p 20 xi-       05   -1.0  1.321330   0.0491   1
       001607 1.0                               # lambda pi-
  p 21 xi0       06   +0.0  1.314850   0.0871   1
       001609 1.0                               # lambda pi0
  p 22 omega     07   -1.0  1.672430   0.0246   3
       001612 0.686                             # lambda K-
       002107 0.234                             # xi0 pi-
       002009 0.080                             # xi- pi0

# Define materials: (material 0 pre-defined as vacuum)
#   num/name      A        Z      den(g/cm2)   rad(cm)
  m 01 air        14.610   7.30   1.205e-3     30423
  m 02 helium     4.0000   2.00   0.125        755.0
```

```
  m 03 aluminum   26.980   13.0   2.700         8.900
  m 04 iron       55.850   26.0   7.870         1.760
  m 05 copper     63.540   29.0   8.960         1.430
  m 06 lead       207.19   82.0   11.35         0.560
  m 07 uranium    238.03   92.0   18.95         0.320

# Input experiment scripts:
  i example
```

A.2 Example file *example.s*

```
# Script file: example.s
#
#    Example experiment script file for the ABC simulator.  First, a universe
# is defined as a box with 20m sides, and given a color 99 (white) and material
# 1 (air).  A new particle, the ABCino, is created and given several possible
# decay modes.  The experiment consists of a cylindrical detector with a small
# magnetic field.  Surrounding the cylinder are vertical slabs of aluminum,
# and a thin slab of iron in front.  Ten generators are defined, one for each
# decay mode of the ABCino, along with decays for kaons, muons, and pions.
# Finally, protons are generated in the backwards direction to test their
# effects in the iron.

# Define universe:
#   col mat xmin xmax ymin ymax zmin zmax
  u 99  01  -10  +10  -10  +10  -10  +10

# Define new particle:
#   num/name     col  chrg  mass(GeV)  life(m)  decay
  p 99 ABCino    01   +0.0  4.00       0.5      4
       070809 0.25                              # pi- pi+ pi0
       121314 0.25                              # K- K+ KL
       121315 0.25                              # K- K+ KS
       171819 0.25                              # S- S+ S0

# Define objects:
#   num/name   shp col mat det bty/mg  pos-(x,y,z)   siz-(1,2,3)   ang-(a,b,c)
  o 01 chamber cyl 01  01  1   d  .20  +0.0 +0.0  0  4.0 10. 10.     0  0  0
  o 11 block1  box 02  03  1   0  0.0  +3.5 +6.06 0  1.0 3.0 10.   +60 0  0
  o 12 block1  box 02  03  1   0  0.0  +6.06 +3.5 0  1.0 3.0 10.   +30 0  0
  o 13 block1  box 02  03  1   0  0.0  +7.00  0   0  1.0 3.0 10.     0  0  0
  o 14 block1  box 02  03  1   0  0.0  +6.06 -3.5 0  1.0 3.0 10.   -30 0  0
  o 15 block1  box 02  03  1   0  0.0  +3.5 -6.06 0  1.0 3.0 10.   -60 0  0
  o 20 plate   box 03  06  1   0  0.0  -7.0 +0.0  0  0.2 10. 10.     0  0  0

# Define generator:
#   num prt flg       pos-(x,y,z)        mom-(x,y,z)        delmom
  g 01  99  1      +0.0  +0.0  +0.0   0.50   0.00   0.00     0.00  # ABCino
  g 02  99  2      +0.0  +0.0  +0.0   0.50   0.00   0.00     0.00  #   dec2
  g 03  99  3      +0.0  +0.0  +0.0   0.50   0.00   0.00     0.00  #   dec3
  g 04  99  4      +0.0  +0.0  +0.0   0.50   0.00   0.00     0.00  #   dec4
  g 05  12  2      +0.0  +0.0  +0.0   1.00   0.00   0.00     0.00  # K- dec2
  g 06  13  3      +0.0  +0.0  +0.0   1.00   0.00   0.00     0.00  # K+ dec3
```

```
g 07  14  6       +0.0  +0.0  +0.0   1.00   0.00   0.00     0.00  # KL dec6
g 08  05  1       +0.0  +0.0  +0.0   0.00   0.00   0.00     0.00  # mu- decay
g 09  07  1       +0.0  +0.0  +0.0   0.50   0.00   0.00     0.00  # pi- decay
g 10  10  0       +0.0  +0.0  +0.0  -1.00   0.00   0.00     0.50  # proton
```

B The "ABC" Main Routine

```
/*
   Program: abc.c

      The ABC program simulates particle physics.  The user first defines
   objects and particles via the command line or script files (*.s), and then
   generates random particle events.  ABC swims the particles through the
   objects, applying physical processes (e.g. energy loss, decay) at each step.
   The user can then display the events or save detector object data to
   standard output.

      This main program defines common process tags and simulator constants,
   initializes the data arrays, and then calls the command input routine
   abc_com.  A default universe is defined as a 1-meter cube with zero color
   and material.
*/

#include "abc.h"

/* Process tags (0=off,1=on unless noted) */

int abctag=1;   /* read abc.s on startup */
int dattag=1;   /* detector output data (0=off,1=normal,2=formatted) */
int dectag=1;   /* particle decay */
int drwtag=1;   /* display drawing package */
int lostag=1;   /* energy loss */
int magtag=1;   /* magnetic fields */
int mlttag=1;   /* multiple scattering */
int trctag=1;   /* error tracing */
int vrbtag=1;   /* verbose state (0=off,1=normal,2=echo input) */

/* Simulator constants */

double inclos=0.01;       /* step increment for energy loss */
double incmag=0.1;        /* step increment for magnetic field */
double incrad=1.0;        /* step increment for radiation length */
double maxstp=0.33;        /* maximum step size */
double minstp=0.000001;   /* minimum step size */
double minmom=0.001;      /* minimum particle momentum */
double mintol=0.000001;   /* minimum edge tolerance */

/* Generator data */

int    genprt[maxgen];
int    genflg[maxgen];
double genxpo[maxgen];
double genypo[maxgen];
double genzpo[maxgen];
double genxmo[maxgen];
double genymo[maxgen];
double genzmo[maxgen];
double gendmo[maxgen];
```

```
/* Detector hit data */

int    hitnum;
int    hitprt[maxhit];
double hitxpo[maxhit];
double hitypo[maxhit];
double hitzpo[maxhit];

/* Track data */

int    trknum;
int    trkpnt[maxtrk];
int    trkprt[maxtrk];
double trktpo[maxtrk][maxpnt];
double trkxpo[maxtrk][maxpnt];
double trkypo[maxtrk][maxpnt];
double trkzpo[maxtrk][maxpnt];
double trkemo[maxtrk][maxpnt];
double trkxmo[maxtrk][maxpnt];
double trkymo[maxtrk][maxpnt];
double trkzmo[maxtrk][maxpnt];

/* Particle data */

int    prtnum[maxprt];
int    prtcol[maxprt];
char   prtnam[maxprt][maxnam];
double prtchr[maxprt];
double prtmas[maxprt];
double prtlif[maxprt];
int    prtmdn[maxprt];
int    prtmdp[maxprt][maxmdn];
double prtmdb[maxprt][maxmdn];

/* Material data */

int    matnum[maxmat];
char   matnam[maxmat][maxnam];
double matatm[maxmat];
double matatn[maxmat];
double matden[maxmat];
double matrad[maxmat];

/* Object data */

int    objnum[maxobj];
int    objcol[maxobj];
int    objmat[maxobj];
int    objdet[maxobj];
char   objbty[maxobj];
double objbmg[maxobj];
char   objnam[maxobj][maxnam];
char   objshp[maxobj][maxnam];
double objdim[maxobj][maxdim];
```

```
/* Color data */

double drwcol[maxcol][4];

/* Physical constants */

double kb=0.307075;              /* Bethe-Bloch constant (MeV*cm2/g) */
double me=0.51099906;            /* electron mass (MeV) */
double pi=3.141592653589793;     /* pi */

main(int argc,char **argv)
{
   int i;
   double a,x,y;

/* Initialize arrays */
   trknum = 0;
   hitnum = 0;
   for(i=0;i<maxcol;i++) drwcol[i][0] = 0;
   for(i=0;i<maxmat;i++) matnum[i] = 0;
   for(i=0;i<maxprt;i++) prtnum[i] = 0;
   for(i=0;i<maxtrk;i++) trkprt[i] = 0;
   for(i=0;i<maxgen;i++) genprt[i] = 0;
   for(i=0;i<maxobj;i++) objnum[i] = 0;

/* Setup default universe */
   strcpy(objnam[0],"universe"); strcpy(objshp[0],"box");
   objcol[0] = 0; objbmg[0] = 0.0; objmat[0] = 0;
   objdet[0] = 0; objbty[0] = '0';
   objdim[0][1] = 0.0; objdim[0][4] = 1.0; objdim[0][7] = 0.0;
   objdim[0][2] = 0.0; objdim[0][5] = 1.0; objdim[0][8] = 0.0;
   objdim[0][3] = 0.0; objdim[0][6] = 1.0; objdim[0][9] = 0.0;
   strcpy(matnam[0],"vacuum");
   matatm[0] = 0.0; matatn[0] = 0.0; matden[0] = 0.0; matrad[0] = 0.0;

/* Execute command input */
   fprintf(stderr,"\n");
   fprintf(stderr,"  Welcome to the ABC Particle Physics Simulator \n");
   com_hlp(); abc_com(argc,argv);
}
```

C "ABC" Installation

```
                The ABC Particle Physics Simulator

     The ABC program simulates particle physics.  The user first defines
  objects and particles via the command line or script files (*.s), and then
  generates random particle events.  ABC swims the particles through the
  objects, applying physical processes (e.g. energy loss, decay) at each step.
  The user can then display the events or save detector object data to
  standard output.

Code installation:

     From the directory /Students/ambrose/abc on wwwrel.ph.utexas.edu,
  retrieve all *.c, *.h, and *.s files, as well as this file (abc.README)
  and the Makefile:

      abc.c - Main program
      abc.h - Header file
      abc.s - Script file (default)

      abc_com.c - Command routine
      abc_dec.c - Particle decay
      abc_drw.c - Display routine (drawing package)
      abc_gen.c - Generator routine
      abc_los.c - Energy loss
      abc_mag.c - Magnetic field routine
      abc_mlt.c - Multiple scattering
      abc_uti.c - Utility routines

Compiling and running:

     Compile the routine with Makefile by simply typing "make".  The only
  special libraries that are needed deal with the graphics.  The OpenGL
  Utility Toolkit (GLUT) library, as well as the MesaGL libraries, must
  be linked to with the statement:

      -L/usr/localn32/lib -lglut -lMesaGLU -lMesaGL -lXmu -lXext -lX11 -lm

  Run the program by typing "abc".  By default, the script file abc.s will
  load the example script (example.s).  This call should be commented out
  (using a "#" at the last line of abc.s) if other scripts are desired.
  Unfortunately, I have only tested the program on einstein.ph.utexas.edu.
  Also, when running ABC on the einstein terminal, the mouse function (for
  zooming in views) does not work correctly, but views can be adjusted by
  scaling and moving.

                                                       - Dave Ambrose
                                                          (4/21/97)
```

References

[1] CERN Program Library, Application Software Group, *GEANT 3.2.1 - Detector Description and Simulation Tool*, (CERN, Geneva, 1993).

[2] J. D. Jackson, *Classical Electrodynamics*, 2nd Edition, (John Wiley & Sons, New York, 1975).

[3] Particle Data Group, K. Hikasa *et al.*, *Review of Particle Properties*, Phys. Rev. **D54**, S1 (1996).

[4] G. Arfken, *Mathematical Methods for Physicists*, 3rd Edition, (Academic Press, Inc., 1985).

[5] W. Mendenhall *et al.*, *Mathematical Statistics with Applications*, 4th Edition, (PWS-KENT Publishing Company, Boston, 1990).

[6] W. H. Press *et al.*, *Numerical Recipes in C*, (Cambridge University Press, 1992).

[7] D. H. Perkins, *Introduction to High Energy Physics*, 3rd Edition, (Addison-Wesley Publishing Company, Inc., 1987).

[8] W. R. Leo, *Techniques for Nuclear and Particle Physics Experiments*, (Springer-Verlag Berlin Heidelberg, 1987).

[9] L. G. Greeniaus, *TRIUMF Kinematics Handbook*, (September, 1987).

[10] D. Griffiths, *Introduction to Elementary Particles*, (John Wiley & Sons, New York, 1987).

[11] V. E. Barnes *et al.*, Phys. Rev. Lett. **12**, 204 (1964).