Simulation of muons energy loss in lithium hydride for the neutrino factory design study

Saurabh Kumar Vij
Delhi University, India
&
ENSTA ParisTech, France

20.08.2010
INDEX

Aim of the Project

I. Neutrino physics

II. Neutrino factory
   a) Proton driver
   b) Target system
   c) Muon front end
      i. Capture and drift
      ii. Bunching and rotation
      iii. Ionization cooling
   d) Muon acceleration system
   e) Muon decay ring

III. Energy loss simulations
   a) Motivation behind the simulations
   b) Geant4 framework
      i. Physics list
      ii. Primary Generator Action
      iii. Detector construction
   c) Simulation description

IV. Results and discussion

V. Conclusion

VI. Acknowledgement

VII. Glossary

IX. Annex

X. Reference
Aim of the Project
The project consisted of a study of the energy loss by muons passing through LiH with the Geant4 simulation package. Simulations have been done using lithium in different composition of $^6\text{Li}$ and $^7\text{Li}$ and then were compared with data from the Particle Data Group. The aim of the study was to try to understand the difference in the models used for the simulation of compounds and estimate the difference in energy loss when the properties of the compound are changed. The output from Geant4 simulations has been used as an input for the ROOT data analysis package.

I. Neutrino Physics
The Standard Model which base was setup by physicists S.L.Glashow, A. Salam and S.Weinberg in the late 1960’s describes the universe with the help of twelve particles (see figure 1), six quarks (up, down, top, bottom, charm and strange) and six leptons (electron, electron neutrino $\nu_e$, muon, muon neutrino $\nu_\mu$, tau and tau neutrino $\nu_\tau$) and three forces, the strong force, the weak force and the electromagnetic force. The carriers of these forces are called bosons. The weak force carriers are the $Z^0$ and the $W^\pm$ bosons, the photon $\gamma$ for the electromagnetic force and the gluon $g$ for the strong force. Gravitation is neglected as it is small in comparison to the other forces. The Standard Model also predicts a Higgs boson which has not been discovered yet and is responsible for the particles mass.

![Figure 1: The Standard Model of elementary particles](image-url)
The neutrino is neutral and is undergoing interactions through the electroweak interactions (the weak and electromagnetic forces have been unified in one force called electroweak interaction) such as radioactive decays or nuclear reactions taking place in the sun. For each particle there is also an antiparticle carrying the opposite charge of the particle.

An example of neutrino creation through the beta ($\beta^-$) decay process:

$$n^0 \rightarrow p^+ + e^- + \text{anti-}\nu_e$$

where $n^0$ stands for the neutron, $p^+$ for the proton, $e^-$ for the electron and anti-$\nu_e$ for the anti-neutrino $\nu_e$

According to the Standard Model of physics, neutrinos have no mass. But many years of solar, atmospheric, accelerators and reactors based neutrino experiments have demonstrated that neutrinos can change flavour and have a mass, thus challenging the very basic hypothesis of the Standard Model. This change of flavour is called neutrino oscillations e.g. an electron neutrino converting into a muon neutrino and vice versa. With the results of these experiments, we have been able to describe the neutrino flavour eigenstates ($\nu_e, \nu_\mu, \nu_\tau$) as a function of the mass eigenstates ($\nu_1, \nu_2, \nu_3$) and a mixing matrix called PMNS from the physicists who got the idea of this description (B. Pontecorvo, Z. Maki, M. Nakagawa, S. Sakata):

$$
\begin{bmatrix}
\nu_e \\
\nu_\mu \\
\nu_\tau
\end{bmatrix} = 
\begin{bmatrix}
U_{e1} & U_{e2} & U_{e3} \\
U_{\mu1} & U_{\mu2} & U_{\mu3} \\
U_{\tau1} & U_{\tau2} & U_{\tau3}
\end{bmatrix}
\begin{bmatrix}
\nu_1 \\
\nu_2 \\
\nu_3
\end{bmatrix}
$$

Where

$$
\begin{bmatrix}
U_{e1} & U_{e2} & U_{e3} \\
U_{\mu1} & U_{\mu2} & U_{\mu3} \\
U_{\tau1} & U_{\tau2} & U_{\tau3}
\end{bmatrix} = 
\begin{bmatrix}
c_{12}c_{13} & s_{12}c_{13} & s_{13}e^{-i\delta} \\
-s_{12}c_{23} - c_{12}s_{23}s_{13}e^{i\delta} & c_{12}c_{23} - s_{12}s_{23}s_{13}e^{i\delta} & s_{23}c_{13} \\
s_{12}s_{23} - c_{12}c_{23}s_{13}e^{i\delta} & -c_{12}s_{23} - s_{12}c_{23}s_{13}e^{i\delta} & c_{23}c_{13}
\end{bmatrix}
$$

Where $\delta$ = the CP phase violation term, $c_{ij} = \cos \theta_{ij}$ and $s_{ij} = \sin \theta_{ij}$.

$\theta_{ij}$ is called the mixing angle between states $i$ and $j$.

Additional phases are present in case the neutrino and the anti-neutrino are the same particles but they do not influence the neutrino oscillation, so they are not shown in the equation above.

The probability of measuring a particular flavour for a neutrino varies periodically as it propagates. The probability that a neutrino originally of a given flavour $\alpha$ will later be observed as having flavour $\beta$ is given by:
$P_{\alpha \rightarrow \beta} = \delta_{\alpha \beta} - 4 \sum_{i > j} \text{Re}(U_{\alpha i}^* U_{\beta i} U_{\alpha j}^* U_{\beta j}) \sin^2 \left( \frac{\Delta m_{ij}^2 L}{4E} \right) + 2 \sum_{i > j} \text{Im}(U_{\alpha i}^* U_{\beta i} U_{\alpha j}^* U_{\beta j}) \sin \left( \frac{\Delta m_{ij}^2 L}{2E} \right)$

$\Delta m_{ij}^2 \equiv m_i^2 - m_j^2$

Where $\Delta m_{ij}^2$ = square mass difference between two neutrino states $i$ and $j$

$\delta_{\alpha \beta} = \text{Kronecker delta}, \quad \delta_{\alpha \beta} = \begin{cases} 1 & \text{if } \alpha = \beta \\ 0 & \text{if } \alpha \neq \beta \end{cases}$

$E =$ energy of the neutrino

$L =$ length over which the neutrino propagated

As we can see from the probability equation, if the neutrinos have a mass, they can oscillate. Therefore in order to describe completely this phenomenon we need to know precisely the mixing angles $\theta_{12}, \theta_{13}$ and $\theta_{23}$, two of the squared mass differences $\Delta m_{12}^2, \Delta m_{23}^2$ and $\Delta m_{32}^2$ and the CP violation phase term $\delta$. Experiments have been able to measure $\Delta m_{21}^2$ and $|\Delta m_{31}^2|$ and the mixing angles $\theta_{12}$ and $\theta_{23}$. Their values from the combined data (Neutrino 2010 conference) are:

$\Delta m_{21}^2 = 7.65^{+0.23}_{-0.20} \times 10^{-5} \text{eV}^2$

$|\Delta m_{31}^2| = 2.40^{+0.12}_{-0.11} \times 10^{-3} \text{eV}^2$

$\sin^2 \theta_{12} = 0.304^{+0.022}_{-0.016}$

$\sin^2 \theta_{23} = 0.50^{+0.07}_{-0.06}$

We do not know yet the sign of $\Delta m_{31}^2, \theta_{13}$ and the phase violation term $\delta$. To be able to make such precise measurements an intense neutrino beam is required. One of the proposed schemes is the neutrino factory [4].
II. The Neutrino Factory

It is a facility which will generate an intense beam of neutrinos coming from muon decay and be able to send $10^{21}$ neutrinos per year (1 accelerator year accounting for $10^7$ s) to distant detectors located several kilometres away. If the neutrino beam is measured as it leaves the end of the muon accelerator and again as it comes out of the far side of the world, we can observe if the composition of the beam has changed and hence study neutrino oscillations.

A neutrino factory (see figure 2) is composed of
- a proton driver
- a target system
- a muon front-end
- a muon acceleration system
- two decay rings

These are described below:

a) Proton driver
This provides an intense 4MW beam of 5-15 GeV protons, which impinges on a Hg target to produce pions. This energy range was chosen from simulation as the optimum for pion production. The driver could be a high current H’ linac feeding an accumulator and a compressor ring, a series of rapid cycling synchrotrons, or a booster synchrotron injecting into a FFAG (Fixed Field Alternating Gradient).
b) Target system
It is generally accepted that the fixed solid target would work up to 1 MW of beam power, but at higher beam powers, a moving target is required, so a liquid Hg jet was chosen as the target for the production of pions. Pions emanate from the target in all directions, with a range of energies. As many as possible are captured in a solenoid channel, with the magnetic fields gradient decreasing from 20 to 1.5 T over a length of 12 m.

c) Muon front-end
The muon front-end (see figure 3) is made of a drift region, a bunch, a rotation and a cooling region.

Figure 3: Front-End of the neutrino factory design

i. Drift
The captured pions decay into muons in a drift region of 111 m length. Only 5% produced decay in useful muons for the neutrino factory accelerator system, this explains why so much proton beam power is needed at the production target. Muons are then bunched and rotated

ii. Bunching and rotation
An RF buncher then turns the muon pulse into trains of alternating $\mu^\pm$ bunches. The muons then pass through an RF phase rotation system that reduces the energy spread of the beam and increases the bunch length. The energy spread is required to be reduced to get a uniform beam with low spread in momentum and energy.

---

1 Words written in *italics* are explained in glossary
iii. Ionization cooling
The muon transverse emittance is too large for the subsequent accelerating structures, and it needs to be reduced by a process known as ionization cooling. The principle is as follows: particles are passed through liquid hydrogen (H$_2$) or lithium hydride (LiH) absorbers and loose momentum in all directions as they ionize the material. Thus the beam emittance is reduced. Now to maintain the longitudinal momentum, we use RF cavities along with the absorber material, these RF cavities increases the longitudinal momentum for the same amount that has been reduced during ionization, compensating for the energy losses in the LiH absorbers, thus providing a beam with the reduced transverse momentum and the emittance.

To demonstrate ionization cooling of muons for the first time, an experiment is under construction at RAL (Rutherford Appleton Laboratory) known as MICE (Muon Ionization Cooling Experiment) [5]. MICE will use a single 5.5 m cooling cell and measure the emittance reduction in the transverse and longitudinal phase. It will measure cooling performance over a range of beam momenta between about 150 and 250 MeV/c, for various absorber materials and magnetic field configurations.
d) Muon acceleration system

![Figure 5: Schematic diagram for the muons acceleration](image)

The half-life of the muons in the rest frame is only 2.2 μs, so there is a great need for the muons to be accelerated very rapidly, such that they can be accelerated to considerable energy before they decay into neutrinos. The 200-300 Mev muons from the cooling channel are first accelerated (see figure 5) by a linac to 0.9 GeV. These 0.9 GeV muons enter the RLA’s (Recirculating Linear Accelerator) in series, which are dog bone shaped and here the beam is recirculated until it reaches 12.6 GeV. This 12.6 GeV beam is then passed through an FFAG (Fixed Field Accelerating Gradient) ring for further acceleration to 25 GeV.

e) Muon decay rings

![Figure 6: Race-Track design for a Muon Decay Ring](image)

It is the part of the neutrino factory where the high energy muons decay to give an intense beam of neutrinos, which are then sent to the far away detectors. The design is based on racetrack shaped rings (see figure 6). With production straights of 600 m, these are over 750 m in total length, with *FODO* cells at the curve to provide focussing of the muon beam. Their lowest point penetrates at a depth of 435 m into the earth, making geological properties of the neutrino factory site important. Two such rings could be built, one for each baseline in separate tunnels, each ring handling either $\mu^+$ or $\mu^-$. The two beams would be switched periodically by reversing all magnet polarities in both rings. Baseline distances needed are 7500 km for one detector, and 3000–5000 km for the other.
III. Energy loss simulations

a) Motivation behind the simulations
As the process of muon ionization cooling necessitates a delicate interplay between optimizing absorber length and RF cavity voltage for the muon momentum transformation, different absorber materials are under study. Simulations have been done with materials such as liquid hydrogen (H₂) and lithium hydride (LiH), in order to choose the best absorber material.

Muons passing through material are losing energy primarily by ionization and atomic excitation. The mean rate of energy loss (or stopping power) is given by the Bethe Bloch [2] equation:

\[ -\left\langle \frac{dE}{dx} \right\rangle = K z^2 \frac{Z}{A} \frac{1}{\beta^2} \left[ \frac{1}{2} \ln \frac{2m_e c^2 \beta^2 \gamma^2 T_{\text{max}}}{I^2} - \beta^2 - \frac{\delta(\beta\gamma)}{2} \right] \]

Where
\[ K/A = 4\pi N_A r_e^2 m_e c^2 / A \]
\[ N_A = \text{Avogadro’s number} = 6.022 \times 10^{23} \text{ mol}^{-1} \]
\[ r_e = \text{classical electron radius} = 2.818 \times 10^{-15} \text{ m} \]
\[ m_e c^2 = \text{electron mass } \times c^2 = 0.511 \text{ MeV} \]
\[ A = \text{atomic mass of the material (in g mol}^{-1} \]
\[ Z = \text{atomic number of the material} \]
\[ z = \text{charge of the incident particle (multiple of elementary charge)}^2 \]
\[ I = \text{Mean excitation energy in eV} \]
As we can see from the equation above, depending on the material properties and the particle mass and energy, the energy loss in material is different.

\[ T_{\text{max}} \] is the maximum kinetic energy which can be imparted to an electron in a single collision. For a particle with mass M and momentum \( \beta\gamma Me \), it is given by

\[ T_{\text{max}} = \frac{2m_e c^2 \beta^2 \gamma^2}{1 + 2\gamma m_e/M + (m_e/M)^2} \]

As the particle energy increases, density effect corrections have to be taken into account and are described by the density effect function \( \delta(\beta\gamma)/2 \).

b) Geant4 framework:
To study the interaction of muons with LiH, Geant4 [1] version 9.3.p01 was used for the simulation. Geant4 provides a complete set of tools for all areas of detector simulation: geometry, tracking, detector response, run, event and track management, visualization and user interface. The multi-disciplinary nature of the toolkit requires that it supply an abundant set of

\[^2\] 1 elementary charge = 1.602176487 \times 10^{-19} \text{ C.}
physics processes to handle diverse interactions of particles with matter over a wide energy range. For many physics processes a choice of different models is available. There are many classes in Geant4 that are required to run a successful simulation but some of the modified are:

1) Physics List
2) Primary Generator Action
3) Detector Construction
4) The main() program

i) Physics List:
In this method, the user has to specify which particles, processes, and production cuts. It gives the freedom to decide which physics processes are to be included along with the prediction for production of certain particles. It deals with all the interactions that the particles can undergo. The processes used in this project are handled by the constructEM () method which handles the following processes:

- Ionization (for electrons and muons)
- Multiple Scattering and Bremsstrahlung (for electron)
- Gamma conversion (for gammas)
- Transportation (for muons)

ii) Primary Generator Action:
This method is used for generating the primary particles and allows to choose the direction, energy and initial position of the particle. It contains an object called G4ParticleGun () which lets the user decide the number of particles to be sent along with the momentum spread.

iii) Detector Construction:
In this we can create the geometry, environment, detectors and material properties. We can define new elements, compounds and mixtures. We don’t have a detector in our case, but we have still used this class to define the absorber material (LiH).

c) Simulation description

In this project simulation was done by sending muons with starting muons of 1000 MeV/c momentum through a fixed LiH box of given dimensions to study the energy loss by the muons. The Geant4 is written in C++ [6] and the code written for the purpose of simulation is provided in Annex 1.

Our setup environment is defined by a LiH box of 8.0 m sitting in an experimental hall full of air (see figure 6). 1000 negative muons (μ⁻) are sent of 1000 MeV/c momentum from x = -4.5 m, y = 0 and z = 0 where x is the direction of the muons. Then we get the information for each step that the muons underwent during the entire simulation. With this useful information, we calculate dE/dx and plot as a function of the total momentum p. To build a LiH box, we first defined the elements Li and H, their atomic masses and the density of LiH in the detector construction method. Then we set the origin and the boundaries of the box. A small piece of
code has been written in the output() method in order to print into a file the information we needed.

Figure 6: An example of simulation using Geant4, here we have 1 muon interacting with the LiH box of smaller dimensions.

In figure 6, the green colour is for the neutral particles like “gammas”, red are for the negatively charged particles, secondary particles like electrons and blue for the muons.
After getting the data from the GEANT4 simulation, the output is read by ROOT [3] version 5.20/00 for further analysis.

First a scatter plot for 1000 muons dE/dx as a function of their momentum is drawn (see figure 7(a) and figure 7(b)).
To study the distribution of muons energy loss over a range of momentum, distribution plots are provided for different momentum ranges (see figure 8(a) and figure 8(b)). The RMS value in the distribution plots is the Standard deviation but for some historical reasons, it is called as the RMS when produced using ROOT [8].
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

Figure 8(a): Distribution plot for muon energy loss in LiH with 100% $^7$Li and $50 \leq P < 100$ MeV/c

Figure 8(b): Distribution plot for muon energy loss in LiH with 100% $^7$Li and $950 \leq P < 1000$ MeV/c
Then in order to derive the mean \( < \frac{dE}{dx} > \) for further comparison, we tried the following methods.

A code in ROOT was written to form a loop which would sum all the values of \( \frac{dE}{dx} \) over a range of momentum and then calculate the average. The code is given in Annex 2.

The calculation method and logic of the code is:

\[ P = \text{momentum} \]
\[ P_i = \text{boundary bins (e.g. } P_i = 50, 100, 150 \ldots \ldots 1000 \text{ MeV/c)} \]

Loop over i check if

\[ \text{If } P_i < P < P_i + 50 \text{ MeV/c} \]

sum = sum + \( \frac{dE}{dx} \)

\[ N = N + 1 \]

At the end and after the loop:

\[ < \frac{dE}{dx} > = \frac{\text{sum}}{N} \]

For example:

Suppose there are 100 points of \( \frac{dE}{dx} \) in a momentum range 200–250, and then the program sums up all the 100 values of \( \frac{dE}{dx} \) in this range. Then \( \frac{dE}{dx} = \frac{\text{sum}}{100} \) for this case.

Lithium hydride is a salt-like crystalline compound with a face-centred cubic structure. It comes in nature as a mix of \( ^6\text{Li} \) (7.5%) and \( ^7\text{Li} \) (92.5%). LiH is chemically reactive and must be protected from atmospheric moisture. Its fabrication and cost depends on the isotope used and purity composition [7]. We have simulated \( \frac{dE}{dx} \) using LiH made either of 100% \( ^6\text{Li} \) or 100% \( ^7\text{Li} \) in order to compare the mean \( \frac{dE}{dx} \) and see if there is any difference.

<table>
<thead>
<tr>
<th>element</th>
<th>atomic mass (gmol(^{-1}))</th>
<th>percentage (%)</th>
<th>LiH density (g/cm(^3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(^6\text{Li})</td>
<td>6.015121</td>
<td>100</td>
<td>0.685</td>
</tr>
<tr>
<td>(^7\text{Li})</td>
<td>7.016003</td>
<td>100</td>
<td>0.782</td>
</tr>
<tr>
<td>Li</td>
<td>6.941</td>
<td>7.5% (^6\text{Li} + 92.5% (^7\text{Li})</td>
<td>0.82</td>
</tr>
</tbody>
</table>

To estimate and add the error bars over the comparison plots, a code was written in ROOT to calculate the Standard deviation of the average \( <\frac{dE}{dx}> \). The code used for this calculation is provided in Annex 2.
The calculation method and logic of the code is:
For each $P_i < P < P_i + 50$ MeV/c

$\text{Diff} = |dE/dx - \text{mean}|^2 + \text{Diff}$

$\text{S.D} = (\text{Diff} / N)^{1/2}$

$\text{S.D} = \text{Standard Deviation}$

Figure 9(a): comparison of GEANT4 simulation results with the PDG data
Figure 9(a) shows the comparison of mean energy loss between the PDG [2] data, LiH made of 100% $^6$Li and LiH made of 100% $^7$Li. LiH density used for the PDG data is 0.82 g/cm$^3$. Error bars have been added using the code discussed above. The difference in the $<\text{dE/dx}>$ seems to be mainly due to the material density. So a new simulation was done by keeping the same material density for $^6$Li and $^7$Li to check whether the isotope factor plays an important role on the $<\text{dE/dx}>$ or not.

**Energy loss as a function of muon momentum**

![Graph showing energy loss as a function of muon momentum]

Figure 9(b): comparison plot of simulation results with the PDG data using same LiH density

Figure 9(b) demonstrate the simulation results using LiH made of 100% $^6$Li and LiH made of 100% $^7$Li with the same LiH density as well as different density values and their comparison with the PDG data and also with each other. Energy loss $<\text{dE/dx}>$ was normalized by their density values.
**Result and Discussion**

From figure 9(a) and 9(b), it is quite evident that the energy loss by muons in LiH is different for different compositions of LiH. For all the momentum ranges from figure 9(a), the difference in $\langle dE/dx \rangle$ between LiH with 100% $^6$Li and LiH with 100% $^7$Li was calculated and the maximum difference found was ~16%. To our surprise, the $\langle dE/dx \rangle$ values of LiH with $^6$Li are closer to the PDG data values, even though the density of LiH used in PDG data is near the density of LiH with $^7$Li. So, it was expected that LiH with 100% of $^7$Li would give $\langle dE/dx \rangle$ values closer to the PDG data values.

**Conclusion**

Difference was found in the $\langle dE/dx \rangle$ values, when LiH with different compositions was used. Later a study was done by keeping the same material density for $^6$Li and $^7$Li and checking whether the isotope factor plays an important role on the $\langle dE/dx \rangle$ or not. It is shown in figure 9(b) and it can be seen from the figure that there is a difference between the $\langle dE/dx \rangle$ values even when same material density (0.82g/cm$^3$) was used. This difference between $^6$Li and $^7$Li is within the error bars, so running simulations with more number of muons would have been a better idea for the comparison.

**Acknowledgement**

It’s been a lifetime experience for me to work at CERN with people who are always so humble to share their experience and knowledge. I would like show my deepest gratitude to my supervisors Dr. M. Martini and Dr. G. Prior, for spending their precious time and sharing their experience with me and constantly motivating me. A special thanks to Dr. G. Prior for helping me so much and devoting a lot of time for getting me familiar and comfortable with my work. They have been a great source of inspiration for me, I have learned a lot from them. Finally I would like to say a very heartfelt thanks to Dr. R.K. Shivpuri and Dr. A. Sharma who made it possible for me to be a part of CERN, I have enjoyed each and every moment here at CERN, and it is in fact my first and best professional experience. I would love to be a part of CERN again in future and contribute as much as I can to this wonderful scientific community.

**Glossary:**

-
1) RF stands for Radio Frequency cavities and they are used to accelerate particles by switching the electric field, the name RF is because the switching rate of the electric fields is so high, that they operate at microwave frequencies. A standing wave is used whose frequency is set such that it gives particles an accelerating push as they pass through. For example, if a series of electron bunches are being accelerated then the sign of the wave will flip from positive to negative as the bunch passes through the cavity, returning to positive as the next bunch arrives.

2) Emittance is defined as the extent occupied by the particles of the beam in space and momentum phase space as it travels. A low emittance particle beam is a beam where the particles are confined to a small distance and have nearly the same momentum.

3) FODO is a cell consisting of FOcusing and DeFOcusing quadrupoles magnets separated by drift regions in between used for the focusing of a beam. These quadrupoles alternatively placed and perpendicular to each other has a net focusing effect. An accelerator is generally composed of a periodic repetition of FODO cells. FODO cells are used in the muon decay rings to focus the circulating muon ring.
Annex1

The code written in C++ required for this simulation is provided here. The part of the code in bold was modified or written by myself for the purpose of the simulation.

1) **new.cc:**
This is the main program where all the required classes are initialized and instantiated.

```cpp
// ********************************************************************
// * License and Disclaimer                                       *
// *                                                            *
// * The Geant4 software is copyright of the Copyright Holders of *
// * the Geant4 Collaboration. It is provided under the terms and *
// * conditions of the Geant4 Software License, included in the file *
// * LICENSE and available at http://cern.ch/geant4/license. These *
// * include a list of copyright holders.                          *
// *                                                            *
// * Neither the authors of this software system, nor their employing *
// * institutes, nor the agencies providing financial support for this *
// * work make any representation or warranty, express or implied, *
// * regarding this software system or assume any liability for its *
// * use. Please see the license in the file LICENSE and URL above *
// * for the full disclaimer and the limitation of liability.       *
// *                                                            *
// * This code implementation is the result of the scientific and *
// * technical work of the GEANT4 collaboration.                  *
// * By using, copying, modifying or distributing the software (or *
// * any work based on the software) you agree to acknowledge its *
// * use in resulting scientific publications, and indicate your *
// * acceptance of all terms of the Geant4 Software license.        *
// ********************************************************************

// $Id: ExN01PhysicsList.cc,v 1.6 2006/06/29 17:47:21 gunter Exp $
// GEANT4 tag $Name: geant4-09-01-patch-02 $
#include "ExN01DetectorConstruction.hh"
#include <iostream>
#include "G4VTrajectory.hh"
#include ".......hh"
#ifdef G4VIS_USE
#include "G4VisExecutive.hh"
#endif
#ifdef G4UI_USE
#include "G4VisExecutive.hh"
#endif

int main(int argc,char** argv){
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```c
FILE *test = fopen("lih6","w+");
fprintf(test,"output result for ROOT\n");
fprintf(test,"EventID:  Px  Py  Pz  dE  dX\n");

//Random engine
CLHEP::HepRandom::setTheEngine(new CLHEP::RanecuEngine);
G4VSteppingVerbose* verbosity = new output(test);
G4VSteppingVerbose::SetInstance(verbosity);

// Construct the default run manager
G4RunManager* runManager = new G4RunManager;

// set mandatory initialization classes
G4VUserDetectorConstruction* detector = new ExN01DetectorConstruction;
runManager->SetUserInitialization(detector);
G4VUserPhysicsList* physics = new ExN01PhysicsList;
runManager->SetUserInitialization(physics);

// set mandatory user action class
G4VUserPrimaryGeneratorAction* gen_action = new ExN01PrimaryGeneratorAction;
runManager->SetUserAction(gen_action);
G4UserSteppingAction* stepping_action = new ExN01SteppingAction;
runManager->SetUserAction(stepping_action);

// Initialize G4 kernel
runManager->Initialize();

// Get the pointer to the User Interface manager
G4UImanager* UI = G4UImanager::GetUIpointer();

if (argc!=1) {
    G4String command = "/control/execute ";
    G4String fileName = argv[1];
    UI->ApplyCommand(command+fileName);
} else {
    #ifdef G4VIS_USE
    G4VisManager* visManager = new G4VisExecutive;
    visManager->Initialize();
    #endif
    G4UIsession* session = 0;
    #ifdef G4UI_USE_TCSH
    session = new G4UIterminal(new G4UItcsh);
    G4cout << "G4UI_USE_TCSH defined" << G4endl;
    #endif
    UI->ApplyCommand("/control/execute vis.mac");
}
```
session->SessionStart();
delete session;

// Get the pointer to the UI manager and set verbosities
G4UImanager* UI = G4UImanager::GetUImanager();
UI->ApplyCommand("/run/verbose 1");
UI->ApplyCommand("/event/verbose 1");
UI->ApplyCommand("/tracking/verbose 10");

// instantiation and initialization of the visualization manager
#if G4VIS_USE
visManager->SetVerboseLevel (3);
visManager->Initialize();
#endif

// Start a run
G4int numberOfEvent = 1;
runManager->BeamOn(numberOfEvent);

#if G4VIS_USE
   delete visManager;
#endif
}
delete runManager;
delete verbosity;
fclose(test);
return 0;
2) **Physics list.cc**:
This class contains in it all the physics processes modified for the simulation.

```cpp
Geant4 Disclaimer { .....same as above .....}
#include " .... .hh “
ExN01PhysicsList::ExN01PhysicsList(): G4VUserPhysicsList()
{
    defaultCutValue = 1.0*cm;
    SetVerboseLevel(0);
}
ExN01PhysicsList::~ExN01PhysicsList()
{};
void ExN01PhysicsList::ConstructParticle()
{
    G4MuonPlus::MuonPlusDefinition();
    G4MuonMinus::MuonMinusDefinition();
    
    // e+/-
    G4Electron::ElectronDefinition();
    G4Positron::PositronDefinition();
    // nu_e
    G4NeutrinoE::NeutrinoEDefinition();
    G4AntiNeutrinoE::AntiNeutrinoEDefinition();
    // nu_mu
    G4NeutrinoMu::NeutrinoMuDefinition();
    G4AntiNeutrinoMu::AntiNeutrinoMuDefinition();
    
    // gamma
    G4Gamma::GammaDefinition();
}

void ExN01PhysicsList::ConstructProcess()
{
    // Define transportation process

    AddTransportation();
    ConstructEM();
    AddStepMax();
}
```

```cpp
#include "... processes .....hh"

void ExN01PhysicsList::ConstructEM()
{
    theParticleIterator->reset();
    while( (*theParticleIterator)() ){
        G4ParticleDefinition* particle = theParticleIterator->value();
```
G4ProcessManager* pmanager = particle->GetProcessManager();
G4String particleName = particle->GetParticleName();

if (particleName == "gamma") {
} else if (particleName == "e-") {
    // electron
    pmanager->AddProcess(new G4eMultipleScattering, -1, 1, 1);
    pmanager->AddProcess(new G4eIonisation, -1, 2, 2);
    pmanager->AddProcess(new G4eBremsstrahlung, -1, 3, 3);
} else if (particleName == "mu+" || particleName == "mu-" ) {
    // muon
    pmanager->AddProcess(new G4MuIonisation, -1, 2, 2);
}

#include "G4StepLimiter.hh"
#include "G4UserSpecialCuts.hh"

void ExN01PhysicsList::AddStepMax()
{
    // Step limitation seen as a process
    G4StepLimiter* stepLimiter = new G4StepLimiter();
    theParticleIterator->reset();

    while (!(*theParticleIterator)) {
        G4ParticleDefinition* particle = theParticleIterator->value();
        G4ProcessManager* pmanager = particle->GetProcessManager();

        if (particle->GetPDGCharge() != 0.0) {
            pmanager->AddDiscreteProcess(stepLimiter);
        }
    }
}

void ExN01PhysicsList::SetCuts()
{
    // the default cut value for all particle types
    SetCutsWithDefault();

    if (verboseLevel>0) DumpCutValuesTable();
}
3) Detector construction.cc:
The LiH box, its geometry, material, elements are defined in this class.

Geant4 Disclaimer { ....same as above .....}
#include “ .... .hh “

ExN01DetectorConstruction::ExN01DetectorConstruction() :
    experimentalHall_log(0), LiH_log(0), experimentalHall_phys(0),
    LiH_phys(0)
{}

ExN01DetectorConstruction::~ExN01DetectorConstruction()
{
}

G4VPhysicalVolume* ExN01DetectorConstruction::Construct()
{
// ----------------------------------------------- materials
// LiH
    G4double ali; // Li atomic mass
    G4double ah; // H atomic mass
    G4double zli; // Li atomic number
    G4double zh; // H atomic number
    G4double density;

    G4int ncomponents;
    G4int natoms;

    // ali = 7.016003*g/mole;     // when ⁷Li is used
    ali = 6.015121*g/mole;       // when ⁶Li is used
    G4Element* elLi = new G4Element("Lithium","Li", zli= 3.,ali);

    ah = 1.01*g/mole;
    G4Element* elH = new G4Element("Hydrogen","H", zh= 1.,ah);

    //Li7
density= 0.782*g/cm3;     // when ⁷Li is used
    G4Material* LiH = new G4Material("Lithium Hydride",density,ncomponents=2);
    //Li6
density= 0.685*g/cm3;     // when ⁶Li is used
    G4Material* LiH = new G4Material("Lithium Hydride",density,ncomponents=2);

    LiH->AddElement(elLi, natoms=1);
    LiH->AddElement(elH, natoms=1);

    // Air
    G4double aN; // N atomic mass
    G4double aO; // O atomic mass
G4double zN; // N atomic number
G4double zo; // O atomic number
G4double fractionmass;

aN = 14.01*g/mol;
G4Element* elN = new G4Element("Nitrogen", "N", zN= 7., aN);

aO = 16.00*g/mole;
G4Element* elO = new G4Element("Oxygen", "O", zo= 8., aO);

density = 1.290*mg/cm3;
G4Material* Air = new G4Material("Air", density, ncomponents=2);
Air->AddElement(elN, fractionmass=70*perCent);
Air->AddElement(elO, fractionmass=30*perCent);

// -------------------------------
// experimental hall (world volume)
// -------------------------------
G4double expHall_x = 5.0*m;
G4double expHall_y = 5.0*m;
G4double expHall_z = 5.0*m;
G4Box* experimentalHall_box = new G4Box("expHall_box", expHall_x, expHall_y, expHall_z);
experimentalHall_log = new G4LogicalVolume(experimentalHall_box, Air,"expHall_log",0,0,0);
experimentalHall_phys = new G4PVPlacement(0, G4ThreeVector(), experimentalHall_log,"expHall",0,false,0);

// -------------------------------
// the LiH Box
G4double box_x = 4*m;
G4double box_y = 4*m;
G4double box_z = 4*m;
G4Box* LiH_box = new G4Box("LiH_box", box_x, box_y, box_z);
LiH_log = new G4LogicalVolume(LiH_box, LiH, "LiH_log",0,0,0);
G4double boxPos_x = 0.0*m;
G4double boxPos_y = 0.0*m;
G4double boxPos_z = 0.0*m
G4ThreeVector(boxPos_x, boxPos_y, boxPos_z);
LiH_log, "LiHbox", experimentalHall_log, false, 0);

return experimentalHall_phys;
}
4) output.cc :
A small code was written in this method for printing out the data from Geant4 simulation into a file, which was then read using ROOT for analysis.

Geant4 Disclaimer { ....same as above .....}
#include " ... .hh "
output::output(FILE* bla){
    fbla = bla;
}
output::~output()
{
}

void output::StepInfo()
{
    G4int prec = G4cout.precision(2);

    if( verboseLevel >= 1 ) {
        if( verboseLevel >= 4 ) VerboseTrack();
        if( verboseLevel >= 3 ) {
            G4cout << std::setw( 9) << "KineE"" "
            << std::setw( 9) << "dEStep"" "
            << std::setw(15) << "Momentum px py pz" << G4endl;

            G4cout << std::setw(6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
                << std::setw(6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"Energy")
                << std::setw(6) << G4BestUnit(fStep->GetStepLength(),"Length")<<G4endl;
            G4cout<< std::setw(10) << G4BestUnit(fTrack->GetMomentum(),"Energy")<< G4endl;

            if (fTrack->GetParentID() == 0) {
                if (fTrack->GetMomentum().x() >=50 & & fTrack->GetMomentum().x() <=1000){
                    fprintf(fbla,"%d %f %f %f %f\n",fTrack->GetTrackID(),fTrack->GetMomentum().x()/MeV,fTrack->GetMomentum().y()/MeV,fTrack->GetMomentum().z()/MeV,fStep->GetTotalEnergyDeposit()/MeV,fStep->GetStepLength()/m);
            }

            }

    G4cout.precision(prec);
}
}

void output::TrackingStarted()
{
    CopyState();
    G4int prec = G4cout.precision(2);
5) Primary generator .cc

Particle direction and energy are defined in this class.

ExN01PrimaryGeneratorAction::ExN01PrimaryGeneratorAction() {
    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4String particleName;
    particleGun->SetParticleDefinition(particleTable->FindParticle(particleName="mu-"));
    particleGun->SetParticlePosition(G4ThreeVector(-4.5*m, 0.0*m, 0.0));
}

ExN01PrimaryGeneratorAction::~ExN01PrimaryGeneratorAction() {
    delete particleGun;
}

void ExN01PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent) {
    G4ThreeVector v(1.0,0.0,0.0);
    particleGun->SetParticleMomentumDirection(v);
    particleGun->GeneratePrimaryVertex(anEvent);
}
6) new.in :

In this class, number of events and particles energy was defined
/tracking/verbose 2

/gun/energy 900 MeV
/gun/particle mu-
/run/beamOn 999

exit

Annex2: This contains all the code created and well defined macro files which were used for reading and analyzing the data provided by the Geant4 simulation.

1) myfile.C :

This contains the code for reading the data, calculating the mean dE/dX and then plotting the vectors on a TGraph and also creating a scatter plot for muons going through LiH with either 100% of $^6$Li or 100% of $^7$Li on the canvas.

```c
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <cstdio>
#include "TF2.h"
#include "TMath.h"
using namespace std;

void myfile(char outplots[] = "geant.ps"){

    gROOT->SetStyle("Plain");
    gROOT->Reset();

    // Define canvas
    TCanvas c0("c1","c1",1000,800);

    //for histogram
    TCanvas *A3 = new TCanvas("A3","Plotting Canvas",150,100,990,660);
    A3->SetFillColor(10);
    A3->Print(Form("%s[",outplots));
    A3->SetGrid();
    A3->cd();
    c0.Print(Form("%s[",outplots));
    c0.SetGrid();
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
std::string filename = "new/lih6";

const Int_t maxline1 = 37121;

Int_t line1;
char header1[1000], header2[1000];

// variables defined
Double_t P[maxline1], Px[maxline1], Py[maxline1], Pz[maxline1], dedx[maxline1];
Double_t EE2[maxline1], x[maxline1], y[maxline1];
Double_t f[maxline1], h[maxline1], P2[maxline1], Evt[maxline1];
Double_t dE[maxline1], dX[maxline1], dex[maxline1], R[maxline1];

// defining histogram
TH2F *hist = new TH2F("hist","energy loss as a function of muon momentum",2000,0.,1000.,2000,0.,11.);

// Tmultigraph
TMultiGraph *pdg = new TMultiGraph("pdg","Muon Radiation Loss Through LiH");
TGraphErrors *GYm1;

// Initialize variables
line1 = 0;

// Opening file which contains data
ifstream fin1(filename.c_str());
if(!fin1){
    cout << "Cannot open filename1 " << endl;
    return;
}
else cout << "file " << filename << " opened" << endl;

//TH1F* hist = new TH1F("hist","hist",20,0,1000);
fin1.getline(header1,1000);
fin1.getline(header2,1000);
while(fin1 && line1 < maxline1){
    //reading data
    fin1 >> Evt[line1] >> Px[line1] >> Py[line1] >> Pz[line1]
    >> dE[line1] >> dX[line1];

    if (dX[line1] != 0)
    {
        dex[line1] = dE[line1]/(dX[line1]*100*0.685);
    }
    f[line1] = Px[line1]*Px[line1] + Py[line1]*Py[line1] + Pz[line1]*Pz[line1];
    P[line1] = sqrt(f[line1]);
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
line1++;
}
  // for mean
const Int_t N=19;
Double_t a[N];
Double_t sum[N];
Double_t avg[N];
Double_t EE1[N];
Int_t count[N];

for(int i=0;i<N;i++)
{
  a[i] = i*1000/N + 50;
  count[i]=0;
  sum[i]=0;
  avg[i]=0;
  cout << "a[i] " << a[i] << endl;
}

for(int k=0;k<maxline1;k++)
{
  for (i=0; i<N; i++)
  {
    if (a[i]<=P[k] && P[k]<(a[i] + 50))
    {
      sum[i]+= dex[k];
      count[i]++;
    }
  }
}

for (i=0; i<N; i++)
if(count[i]!=0)
{
  avg[i] = sum[i]/count[i];
}

  // to get the average on canvas
TCanvas *mycanv = new TCanvas("mycanv","Muon Energy Loss scatter plot ",600,600);
TGraphErrors* plot = new TGraphErrors(N,a,avg);
plot->SetTitle("Mean Muon Energy Loss In LiH with Li(7)"割);
plot->GetXaxis()->SetTitle("P (MeV/c)"割);
plot->GetYaxis()->SetTitle("dE/dX (MeV.cm^2/g)"割);
plot->GetYaxis()->SetRangeUser(0,10);
plot->GetXaxis()->SetRangeUser(0,1000);
plot->Draw("APL");
plot->SetMarkerColor(4);
plot->SetLineColor(1);
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
plot->SetMarkerStyle(20);
mycanv->SaveAs("avg.png");

for(int i=0;i<N;i++)
{
    printf("Momentum: %e, Average dEdx: %e\n",a[i],avg[i]);
}
for(int i=0;i<maxline1;i++)
{
    hist->Fill(P[i],dex[i]);
}
hist->Draw();
hist->SetXTitle("P (MeV/c)");
hist->SetYTitle("dE/dX (MeV.cm^2/g)");
leg = new TLegend(0.5,0.7,0.89,0.5);
leg->SetTextSize(0.02);
leg->SetFillColor(0);
leg->AddEntry(hist,"LiH with 100% ^{6}Li","p");
leg->Draw();
fin1.close();
fin1.clear();
c0.cd();

TGraphErrors *GYm1 = new TGraphErrors(maxline1,P,dex);
c0.Clear();
GYm1->SetMarkerColor(2);
GYm1->SetLineColor(4);
GYm1->SetLineWidth(2);
GYm1->SetMarkerStyle(2);
pdg->Add(GYm1);
pdg->Draw("AP");
pdg->GetXaxis()->SetTitle("P{Mom.}(MeV/C)"");
pdg->GetYaxis()->SetTitle("Diff(dE/dX)");
pdg->GetYaxis()->SetTitleSize(0.025);
pdg->GetYaxis()->SetRangeUser(0,15);
pdg->GetXaxis()->SetRangeUser(0,1020);
c0.Print("%s\]",outplots);
c0.Print(Form("%s",outplots));
```
2) final.C
This macro is responsible for the comparison plots between PDG data, LiH with 100% of $^7$Li and LiH with 100% of $^6$Li with different LiH densities.

```cpp
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <cstdio>
#include <TMath.h>
using namespace std;

void final(char outplots[] = "newcompare.ps"){

    // FILE *test = fopen("avgvalues.txt","w+");
    gROOT->SetStyle("Plain");
    gROOT->Reset();
    gStyle->SetOptFit();
    gStyle->SetOptStat(111111);

    // Define canvas
    TCanvas c0("c1","c1",1000,800);
    c0.Print(Form("%s[",outplots));
    c0.SetGrid();
    //c0.SetLogy();

    std::string filename1 = "r.txt";
    std::string filename2 = "new/LiH7";
    std::string filename3 = "new/LiH6";

    Int_t maxline1 = 31;
    Int_t maxline2 = 42536;
    Int_t maxline3 = 42233;

    Int_t line1,line2,line3;

    char header1[1000],header2[1000],header3[1000],header4[1000],header5[1000];

    //variables pdg
    Double_t T[31],Pd[31],I[31],EE1[31];
    Double_t b[31],p[31],ph[31],R[31],u[31];
    Double_t pdex[31],r[31],dlt[31],dr[31];

    //variables lih(7)
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
Double_t P[42536], Px[42536], Py[42536], Pz[42536];
Double_t EE2[19], Evt[42536];
Double_t f[42536], h[42536], P2[42536];
Double_t dE[42536], dX[42536], dex[42536];
Double_t df[42536], sdf[42536], l[42536], ls[42536];

// variables lih(6)
Double_t P6[42233], Px6[42233], Py6[42233], Pz6[42233];
Double_t EE3[42233], Evt6[42233];
Double_t f6[42233], h6[42233], P26[42233];
Double_t DE[42233], DX[42233], dex6[42233];
Double_t df2[42233], sdf2[42233], l2[42233], ls2[42233];

TH1F *his = new TH1F("his","to see bin content",20,0.,1000.);
TH1F *his2 = new TH1F("his2","to see bin content",20,0.,1000.);
TH1F *his2 = new TH1F("his2","to see bin content",50,0.,10.);

TMultiGraph *super = new TMultiGraph("super","Energy loss as a function of muon momentum");

TGraphErrors *GYmp;
TGraphErrors *GYm7;
TGraphErrors *GYm6;

// Initialize variables
line1 = 0;
line2 = 0;
line3 = 0;

// Open file
ifstream fin1(filename1.c_str());
if(fin1){
    cout << "Cannot open filename1 " << endl;
    return;
}
else cout << "file " << filename1 << " opened" << endl;

fin1.getline(header1,1000);

while(fin1 && line1 < maxline1){
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
u[line1] = pdex[line1]*0.82;
line1++;
}

fin1.close();
fin1.clear();

// Open input file2
ifstream fin2(filename2.c_str());
if(!fin2){
    cout << "Cannot open filename2 " << endl;
    return;
}
else cout << "file " << filename2 << " opened" << endl;

fin2.getline(header2,1000);
fin2.getline(header3,1000);
while(fin2 && line2 < maxline2){
    f[line2] = Px[line2]*Px[line2] + Py[line2]*Py[line2] + Pz[line2]*Pz[line2];
P[line2] = sqrt(f[line2]);
    if (dX[line2]!=0 && P[line2] >= 50 ){
        dex[line2] = dE[line2]/(dX[line2]*100*0.782);
    }
    line2++;
}
```

// for(int i=0;i<42536;i++){
  // if(500 <= P[i] && P[i] <550){
  //   // for(int i=0;i<maxline1;i++){
  //   //  //cont = his->GetBinContent(4);
  //   //  //for(int i=0; i<maxline1; i++){
  //   //  //  
  //   //  //}
  //  //}
  //   //  //}
  //  //}
  //}
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
// cont = his->GetBinContent(4);
// if(bin = 1){
// if(50 <= P[i] <100){
// his->Fill(dex[i]);
// }

// }
// }
// his->Draw();

for(int i=0; i<42536; i++){
  his->Fill(P[i]);
}
// his->Draw();

const Int_t N=19;
Double_t a[N];
Double_t d[N];
Double_t sum[N];
Double_t avg[N];
Int_t count[N];

for(int i=0;i<N;i++)
{
  a[i] = i*1000/20 + 50;
  d[i] = a[i] + 25;
  count[i]=0;
  sum[i]=0;
  avg[i]=0;
}

for(int k=0;k<maxline2;k++)
{
  for (int i=0; i<N; i++)
  {
    if (a[i]<=P[k] && P[k]<(a[i] + 50))
    {
      sum[i]+= dex[k];
      count[i]++;
    }
  }
}

for (int i=0; i<N; i++)
if(count[i]!=0)// && avg[i]!=0)
{
  avg[i] = sum[i]/count[i];
}
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```c
const Int_t n=19;
Int_t bin;
Double_t sd[n];
Double_t std[n];
Double_t sem[n];

for(int i=0;i<n;i++)
{
    a[i] = i*1000/20 + 50;
}

for(int k = 0; k < maxline2; k++)
{
    for (int i=0; i<n; i++)
    {
        if (a[i]<=P[k] && P[k]<(a[i] + 50)){
            df[i] = df[i] + (dex[k] - avg[i])*(dex[k] - avg[i]);
        }
    }
}

for (int i=0; i<n; i++)
{
    // printf("sqrdf: \%e\n",sdf[k]);
    // sum[i]+= sdf[k];
    sd[i] = sqrt(df[i]/count[i]);
    //printf("sum: \%e\n",sum[i]);
    printf("sd: \%e\n",sd[i]);
}
fin2.close();
fin2.clear();

// Open input file3
ifstream fin3(filename3.c_str());
if(!fin3){
    cout << "Cannot open filename3 " << endl;
    return;
}
else cout << "file " << filename3 << " opened" << endl;
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```cpp
fin3.getline(header4,1000);
fin3.getline(header5,1000);

while(fin3 && line3 < maxline3)
{
    fin3 >> Evt6[line3] >> Px6[line3] >> Py6[line3] >> Pz6[line3]
    >> DE[line3] >> DX[line3];

    f6[line3] = Px6[line3]*Px6[line3] + Py6[line3]*Py6[line3] + Pz6[line3]*Pz6[line3];

    P6[line3] = sqrt(f6[line3]);

    if (DX[line3]!=0 && P6[line3] >= 50)
    {
        dex6[line3] = DE[line3]/(DX[line3]*100*0.685);
    //dex6[line3] = DE[line3]/(DX[line3]*100);
    
    }
    line3++;
}

// for(int i=0;i<maxline3;i++){
//     // if(500 <= P6[i] && P6[i] < 550){
//         // for(int i=0;i<maxline1;i++){
//             // cont = his->GetBinContent(4);
//             // for(int i=0; i<maxline1; i++){
//                 // cont = his->GetBinContent(4);
//                 // if(bin = 1){
//                     // if(50 <= P[i] <100){
//                         // his2->Fill(dex6[i]);
//                         // // }
//                     // }
//                     // // }
//                     // // his2->Draw();
//                 // }
//             // }
//         // }
//     // }
//     // his2->Fill(P6[i]);
// }
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```c
const Int_t m=19;
Double_t b[m];
Double_t e[m];
Double_t sums[m];
Double_t avgs[m];
Int_t counts[m];

for(int j=0;j<m;j++)
{
    b[j] = j*1000/20 + 50;
    e[j] = b[j] + 25;
    counts[j]=0;
    sums[j]=0;
    avgs[j]=0;
}

for(int q=0;q<maxline3;q++)
{
    for (j=0; j<m; j++)
    {
        if (b[j]<=P6[q] && P6[q]<(b[j] + 50))
        {
            sums[j]+= dex6[q];
            counts[j]++;  
        }
    }
}

for (j=0; j<m; j++)
if(counts[j]!=0)
{  
    avgs[j] = sums[j]/counts[j];
}

```

```c
//   for(int j=0;j<N;j++)  
//    {  
//        printf("Momentum: %e, Average dEdx: %e\n",e[j],avgs[j]);  
//    }
```

```
const Int_t g=19;
Double_t sd2[g];
Double_t std2[g];
Double_t sem2[g];

for(int j=0;j<g;j++)
{  
    b[j] = j*1000/20 + 50;
}
```
for(int k = 0; k < maxline3; k++){
    for (int j=0; j<g; j++){
        if (b[j]<=P6[k] && P6[k]<(b[j] + 50)){
            df2[j] = df2[j] + (dex6[k] - avgs[j])*(dex6[k] - avgs[j]);
        }
    }
    for (int j=0; j<g; j++){
        sd2[j] = sqrt(df2[j]/counts[j]);
        printf("sd2: %e\n",sd2[j]);
    }
}

fin3.close();
fin3.clear();

TGraphErrors *GYmp = new TGraphErrors(maxline1,Pd,pdex);
TGraphErrors *GYm7 = new TGraphErrors(N,d,avg,0,sd);
TGraphErrors *GYm6 = new TGraphErrors(m,e,avgs,0,sd2);

GYmp->SetMarkerColor(1);
GYmp->SetLineColor(1);
GYmp->SetLineWidth(2);
GYmp->SetMarkerStyle(21);
super->Add(GYmp);

GYm7->SetMarkerColor(4);
GYm7->SetLineColor(4);
GYm7->SetLineWidth(2);
GYm7->SetMarkerStyle(2);
super->Add(GYm7);

GYm6->SetMarkerColor(2);
GYm6->SetLineColor(2);
GYm6->SetLineWidth(2);
GYm6->SetMarkerStyle(5);
super->Add(GYm6);

super->Draw("ap");
super->GetXaxis()->SetTitle("P (MeV/c)"pname="a.png" );
super->GetYaxis()->SetTitle("dE/dx (MeV.cm^2/g)"pname="b.png" );
//super->GetYaxis()->SetTitle("dE/dx (MeV/cm)"pname="c.png" );
super->GetYaxis()->SetLabelSize(0.025);
super->GetYaxis()->SetTitleSize(0.025);
super->GetYaxis()->SetRangeUser(0,8);
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```
// for legend
leg = new TLegend(0.65,0.7,0.89,0.89);
leg->AddEntry(GYmp,"PDG data","p");
leg->AddEntry(GYm7,"LiH with 100% ^7Li","p");
leg->AddEntry(GYm6,"LiH with 100% ^6Li","p");
leg->Draw();

c0.SaveAs("finres.png");
c0.Print(outplots);

c0.Print(Form("%s",outplots));
```
3) wecomsame.C
This macro is responsible for the comparison plots between PDG data, LiH with 100% of $^7$Li and LiH with 100% of $^6$Li with LiH having same density (0.82 g/cm$^3$) for both the cases.

```cpp
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <cstdio>
#include <TMath.h>
using namespace std;

void wecomsame(char *outplots) {
    gROOT->SetStyle("Plain");
gROOT->Reset();

gStyle->SetOptFit();
gStyle->SetOptStat(111111);

// Define canvas
TCanvas c0("c1","c1",1000,800);
c0.Print(Form("%s[",outplots));
c0.SetGrid();

std::string filename1 = "r.txt";
std::string filename2 = "new/Ls7";
std::string filename3 = "new/Ls6";

Int_t maxline1 = 31;
Int_t maxline2 = 41694;
Int_t maxline3 = 39210;

Int_t line1,line2,line3;
char header1[1000],header2[1000],header3[1000],header4[1000],header5[1000];

//variables pdg
Double_t T[31],Pd[31],I[31],EE1[31];
Double_t b[31],p[31],ph[31],R[31],u[31];
Double_t pdex[31],r[31],dlt[31],dr[31];

//variables lih(7)
Double_t P[41694],Px[41694],Py[41694],Pz[41694];
```
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

Double_t EE2[19], Evt[41694];
Double_t f[41694], h[41694], P2[41694];
Double_t dE[41694], dX[41694], dex[41694];
Double_t df[41694], sdf[41694], l[41694], ls[41694];

//variables lih(6)
Double_t P6[39210], Px6[39210], Py6[39210], Pz6[39210];
Double_t EE3[39210], Evt6[39210];
Double_t f6[39210], h6[39210], P26[39210];
Double_t DE[39210], DX[39210], dex6[39210];
Double_t df2[39210], sdf2[39210], l2[39210], ls2[39210];

TH1F *his = new TH1F("his","to see bin content",20,0.,1000.);
TH1F *his2 = new TH1F("his2","to see bin content",20,0.,1000.);
TMultiGraph *super = new TMultiGraph("super","Energy loss as a function of muon momentum");

TGraphErrors *GYmp;
TGraphErrors *GYm7;
TGraphErrors *GYm6;

// Initialize variables
line1 = 0;
line2 = 0;
line3 = 0;

// Open file r
ifstream fin1(filename1.c_str());
if(!fin1){
    cout << "Cannot open filename1 " << endl;
    return;
}
else cout << "file " << filename1 << " opened " << endl;

fin1.getline(header1,1000);

while(fin1 && line1 < maxline1){

    u[line1] = pdex[line1]*0.82;

    line1++;
}
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```c
fin1.close();
fin1.clear();

// Open input file2

ifstream fin2(filename2.c_str());
if(!fin2){
    cout << "Cannot open filename2 " << endl;
    return;
}
else cout << "file " << filename2 << " opened" << endl;

fin2.getline(header2,1000);
fin2.getline(header3,1000);

while(fin2 && line2 < maxline2){
    fin2 >> Evt[line2] >> Px[line2] >> Py[line2] >> Pz[line2]
        >> dE[line2] >> dX[line2];
    f[line2] = Px[line2]*Px[line2] + Py[line2]*Py[line2] + Pz[line2]*Pz[line2];
P[line2] = sqrt(f[line2]);

    if (dX[line2]!=0 && P[line2] >= 500) {
        //dex[line2] = dE[line2]/(dX[line2]*100*0.82);
        dex[line2] = dE[line2]/(dX[line2]*100);
    }
    line2++;
}

for(int i=0; i<41694; i++){
    his->Fill(P[i]);
}

const Int_t N=19;
Double_t a[N];
Double_t d[N];
Double_t sum[N];
Double_t avg[N];
Int_t count[N];

for(int i=0;i<N;i++) {
```
a[i] = i*1000/20 + 50;
d[i] = a[i] + 25;
count[i]=0;
sum[i]=0;
avg[i]=0;
}

for(int k=0;k<maxline2;k++) {
    for (i=0; i<N; i++) {
        if ( a[i] <= P[k] && P[k] < (a[i] + 50) ) {
            sum[i]+= dex[k];
count[i]++;
        }
    }
}

for (i=0; i<N; i++) {
    if(count[i]!=0) {
        avg[i] = sum[i]/count[i];
    }
}

// for error bars
const Int_t n=19;
Int_t bin;
Double_t sd[n];
Double_t std[n];
Double_t sem[n];

for (i=0; i < n; i++) {
    cont = his->GetBinContent(i+1);
    sum[i]=0;
    bin=i+1;
    for(int k = 0; k < cont; k++) {
        df[k] = dex[k] - avg[i];
sdf[k] = df[k]*df[k];
        sum[i] += sdf[k];
    }
    printf("bin: %e\n",bin);
    if (cont != 0){
        sd[i] = sum[i]/cont;
    }
    printf("cont: %e\n",cont);
    std[i] = sqrt(sd[i]);
    printf("std: %e\n",std[i]);
}

//calculating standard error
l[k] = sqrt(cont);
if (l[k] != 0){
sem[i] = std[i]/l[k];
}
}

fin2.close();
fin2.clear();
// Open input file3
ifstream fin3(filename3.c_str());
if(!fin3){
    cout << "Cannot open filename3 " << endl;
    return;
} else cout << "file " << filename3 << " opened" << endl;

fin3.getline(header4,1000);
fin3.getline(header5,1000);

while(fin3 && line3 < maxline3){
    fin3 >> Evt6[line3] >> Px6[line3] >> Py6[line3] >> Pz6[line3]
                 >> DE[line3] >> DX[line3];
    f6[line3] = Px6[line3]*Px6[line3] + Py6[line3]*Py6[line3] + Pz6[line3]*Pz6[line3];
P6[line3] = sqrt(f6[line3]);
    if (DX[line3]!0 && P6[line3] >= 50) {
        dex6[line3] = DE[line3]/(DX[line3]*100*0.82);
    }
    line3++;
}

for(int i=0; i<39210; i++){
    his2->Fill(P6[i]);
}

const Int_t m=19;
Double_t b[m];
Double_t e[m];
Double_t sums[m];
Double_t avgs[m];
Int_t counts[m];

for(int j=0;j<m;j++) {
    b[j] = j*1000/20 + 50;
    e[j] = b[j] + 25;
    counts[j]=0;
    sums[j]=0;
    avgs[j]=0;
for(int q=0; q<maxline3; q++) {
    for (j=0; j<m; j++) {
        if ( b[j] <= P6[q] && P6[q] < (b[j] + 50) ) {
            sums[j]+= dex6[q];
            counts[j]++;
        }
    }
}

for (j=0; j<m; j++)
    if(counts[j]!=0)
    {
        avgs[j] = sums[j]/counts[j];
    }

const Int_t g=19;
Double_t sd2[g];
Double_t std2[g];
Double_t sem2[g];

for (j=0; j < g; j++)
{
    cont2 = his2->GetBinContent(j+1);
    sum[j]=0;
    bin=j+1;
    for(int w = 0; w < cont2; w++)
    {
        df2[w] = dex6[w] - avgs[j];
        sdf2[w] = df2[w]*df2[w];
        sum[j]+= sdf2[w];
    }
    printf("bin: %e\n",bin);

    if (cont2 != 0){
        sd2[j] = sum[j]/cont2;
    }
    printf("cont: %e\n",cont2);
    std2[j] = sqrt(sd2[j]);
    printf("std: %e\n",std2[j]);

    //calculating standard error
    l2[w] = sqrt(cont2);
    if (l2[w] != 0){
        sem2[j] = std2[j]/l2[w];
        printf("sem2: %e\n",sem2[j]);
    }
}

fin3.close();
Simulation of muons energy loss in lithium hydride for the neutrino factory design study

```c
fin3.clear();

TGraphErrors *GYmp = new TGraphErrors(maxline1,Pd,u);
TGraphErrors *GYm7 = new TGraphErrors(N,d,avg,0,sem);
TGraphErrors *GYm6 = new TGraphErrors(m,e,avgs,0,sem2);

GYmp->SetMarkerColor(1);
GYmp->SetLineColor(1);
GYmp->SetLineWidth(2);
GYmp->SetMarkerStyle(21);
super->Add(GYmp);

GYm7->SetMarkerColor(4);
GYm7->SetLineColor(4);
GYm7->SetLineWidth(2);
GYm7->SetMarkerStyle(2);
super->Add(GYm7);

GYm6->SetMarkerColor(2);
GYm6->SetLineColor(2);
GYm6->SetLineWidth(2);
GYm6->SetMarkerStyle(5);
super->Add(GYm6);

super->Draw("ap");
super->GetXaxis()->SetTitle("P (MeV/C)" );
super->GetYaxis()->SetTitleOffset(1.8);
//super->GetYaxis()->SetTitle("dE/dx (MeV.cm^{2}/g)" );
super->GetYaxis()->SetTitle("dE/dx (MeV/cm) " );
super->GetYaxis()->SetLabelSize(0.025);
super->GetYaxis()->SetTitleSize(0.025);
super->GetYaxis()->SetRangeUser(0,8);
// for legend
leg = new TLegend(0.65,0.7,0.89,0.89);
leg->AddEntry(GYmp,"PDG data","p");
leg->AddEntry(GYm7,"LiH with 100% ^{7}Li","p");
leg->AddEntry(GYm6,"LiH with 100% ^{6}Li","p");
leg->Draw();

c0.SaveAs("wecomesamenon.png");
c0.Print(outplots);
c0.Print(Form("%s",outplots));
```
REFERENCES


