44-88 MHz MUON FRONT-END FOR THE NEUTRINO FACTORY DESIGN STUDY

Alexandri Anastasia Evgenia

University of Patras, Rio, Greece

Summer Student Project

3.7.2011 – 26.8.2011

INDEX

1) Introduction

This project was performed between the 3rd of July and the 26th of August 2011, during the Summer Student program organised at CERN [(1)]. It is an alternative study to the International Design Study for the Neutrino Factory (IDS-NF) [1] and is focusing on a 44 - 88 MHz muon front-end. The aim is to re-evaluate a muon front – end scenario [2] of the Neutrino Factory which used 44-88 MHz radio-frequency (RF) cavities, capturing one particle sign at a time, in a single bunch-to-bucket mode. In this report, we describe the basics of neutrino physics, the current Neutrino Factory baseline and the results of our simulations for the 44-88 MHz proposed alternative front – end.

2) Neutrino physics

According to our knowledge, the observable ordinary matter constitutes only 4.6% of the mass-energy composition of the Universe. The rest of the Universe consists of 23% Dark Matter and 72% Dark Energy, which are exotic forms of matter and energy about which little is known. The known matter is described by the so called Standard Model of Physics (SM), which is the most prevailed model until now. According to the SM, two basic sets of particles, the quarks and the leptons (see Figure 1) and their antiparticles are sufficient to describe our Universe. These particles interact with each other through four fundamental forces. These forces are the electromagnetism, the strong interaction, the weak interaction and the gravitation. They are transmitted by the exchange of particles (force carriers) called gauge bosons. Three quarks can combine to form particles called baryons (e.g. neutrons and protons), while a pair of a quark and an antiquark will form particles called mesons (e.g. the pion, π). The quarks interact through all four fundamental interactions, while the leptons do not participate in strong interactions. Finally, in the SM the particles gain mass through the Higgs mechanism [(2)].

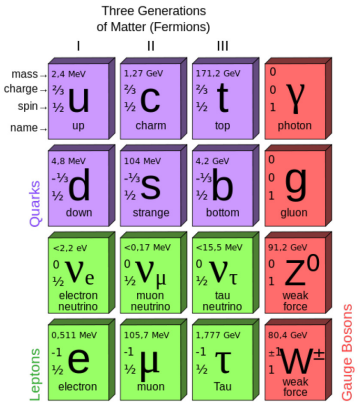

Figure 1: Elementary particles and the forces carriers, as described in the Standard Model [3].

There are three generations of leptons. Neutrinos are leptons; they come in three flavours, each of them associated to a different lepton. The first generation is

the electronic leptons (e^-, ν_e), the second is the muonic leptons (μ^-, ν_μ) and the third is the tauonic leptons (τ^-, ν_τ). In the SM, the lepton number L [3] and the leptonic family numbers L_e, L_μ, and L_τ are conserved in the interactions as in the examples given in Table 1. The three known flavours of neutrinos interact only through the weak forces, with the exchange of a $W^\pm$ (charged-current interaction) or a Z^0 (neutral-current interaction) boson; they have no electric charge. According to the SM neutrinos are massless (see Table 2 for leptons properties).

Leptonic family numbers	**Neutrino Interactions**							**Lepton number**
	n	→	p	+	e^-	+	$\bar{\nu}_e$	
L_e	0		0		+1		-1	
L								0
	μ^+	→	e^+	+	ν_e	+	$\overline{\nu}_\mu$	
L_e	0		-1		+1		0	
L_μ	-1		0		0		-1	
L								-1
	τ^-	→	e^-	+	ν_τ	+	$\bar{\nu}_e$	
L_e	0		+1		0		-1	
L_τ	+1		0		+1		0	
L								+1

Table 1: Some interactions involving neutrinos. The first equation is the beta-decay β^- interaction, the second equation is the muon decay main channel and the third equation is one of the tau decay possible channels. The leptonic family and lepton numbers are conserved in all three interactions.

LEPTONS IN THE STANDARD MODEL						
Name	**Symbol**	**CHARGE Q(e)**	**L_e**	**L_μ**	**L_τ**	**Mass (MeV/c^2)**
Electron	e^-	-1	+1	0	0	0.5
Positron (Antielectron)	e^+	+1	-1	0	0	0.5
Electron neutrino	ν_e	0	+1	0	0	0
Electron anti-neutrino	$\bar{\nu}_e$	0	-1	0	0	0
Muon minus (muon)	μ^-	-1	0	+1	0	105.7
Muon plus (anti-muon)	μ^+	+1	0	-1	0	105.7
Muon neutrino	ν_μ	0	0	+1	0	0
Muon anti-neutrino	$\bar{\nu}_\mu$	0	0	-1	0	0
Tau minus (tau)	τ^-	-1	0	0	+1	1776.8
Tau plus (anti-tau)	τ^+	+1	0	0	-1	1776.8
Tau neutrino	ν_τ	0	0	0	+1	0
Tau anti-neutrino	$\bar{\nu}_\tau$	0	0	0	-1	0

Table 2: Basic properties of leptons in the SM. Q is the electric charge of the particles and L_e, L_μ, and L_τ are the three leptonic family numbers.

Several experiments have produced results indicating that neutrinos oscillate. Neutrino oscillation is a phenomenon in which neutrinos change from one flavour to another when they propagate for a long distance. In order for this phenomenon to occur neutrinos are required to have mass. In addition, the leptonic family number is not conserved if neutrinos oscillate. Neutrino oscillations have been confirmed experimentally leading to new physics. As a result we know that the SM is incomplete and thus it must be extended to include the neutrino mass, mixing among the three neutrino flavours and leptonic family numbers non-conservation.

The observed neutrino flavour eigenstate is in reality a superposition of three neutrino mass eigenstates. The neutrino mixing is described by the relation below, where U is the PMNS (Pontecorvo-Maki-Nakagawa-Sakata) leptonic mixing matrix (in analogy to quark mixing [4]).

$$|\nu_\alpha\rangle = \sum_\iota U_{\alpha i}^* \, |\nu_i\rangle$$

The PMNS matrix is unitary. The probability of a neutrino to oscillate from a flavour α into another flavor β is:

$$P_{\nu\alpha\to\nu\beta} = \delta_{\alpha\beta} - 4\sum_{i>j} Re\left(U_{\alpha i}^* U_{\beta i} U_{\alpha j} U_{\beta j}^*\right) sin^2\left(\frac{\Delta m_{ij}^2 \, \mathrm{L}}{4E}\right) + 2\sum_{i>j} Im(U_{\alpha i}^* U_{\beta i} U_{\alpha j} U_{\beta j}^*) \, sin\left(\frac{\Delta m_{ij}^2 \, \mathrm{L}}{2E}\right)$$

with Δm_{ij}^2 being the difference of the squared masses between two neutrino mass eigenstates, E the energy of the neutrino and L the distance it propagates.

The general form of the PMNS matrix can be written as below:

$$U = \begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix}$$

The PMNS matrix can be parameterized as below:

$$U = \begin{bmatrix} c_{12}c_{13} & s_{12}c_{13} & s_{13}\, e^{-i\delta} \\ -s_{12}c_{23} - c_{12}s_{23}s_{13}e^{i\delta} & c_{12}c_{23} - s_{12}s_{23}s_{13}e^{i\delta} & s_{23}c_{13} \\ s_{12}s_{23} - c_{12}c_{23}s_{13}e^{i\delta} & -c_{12}s_{23} - s_{12}c_{23}s_{13}e^{i\delta} & c_{23}c_{13} \end{bmatrix} \times diag(e^{i\frac{\alpha_1}{2}}, e^{i\frac{\alpha_2}{2}}, 1)$$

With $s_{ij} = \sin\theta_{ij}$ and $c_{ij} = \cos\theta_{ij}$, θ_{ij} are the mixing angles and δ is the CP violation phase [5]. The phase factors, α_1 and α_2, appear only if neutrinos are Majorana particles, i.e. the neutrino and the antineutrino is the same particle.

It is still undefined whether neutrinos are Dirac (particles and antiparticles differ) or Majorana particles and many questions arose about their nature and their

properties. As it was mentioned, there are at least three neutrino mass eigenstates and three neutrinos flavour eigenstates, but a fourth state not sensitive to the weak interaction, can exist (sterile neutrinos). The ordering (called hierarchy) of the neutrino mass eigenstates can be normal or inverted (see Figure 2). Some of the parameters such as the mixing angles and the squared mass differences have been measured, but others are still unknown. The Majorana nature of neutrinos can only be confirmed with neutrinoless double beta decay experiments [6]. Information on the mechanism with which the neutrino mass is generated, whether neutrinos are Majorana or Dirac particles or the absolute value of the neutrino mass cannot be obtained at a Neutrino Factory.

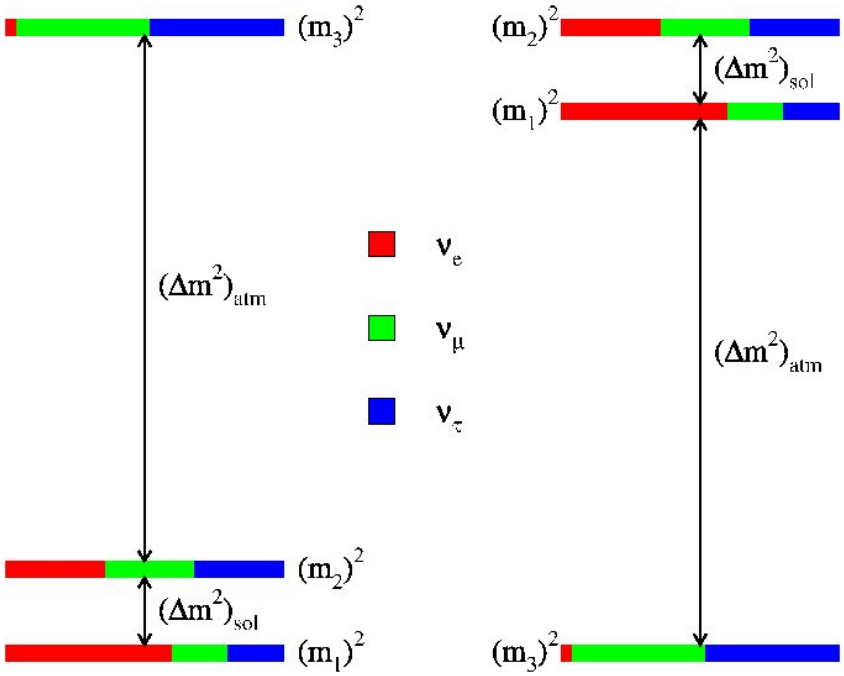

Figure 2: Neutrino normal (left) and inverted (right) mass hierarchy. The colors for each mass eigenstate show the relative flavour abundance [4].

The absolute neutrino mass is not known. In the oscillation experiments only the squared differences of the mass eigenstates are available ($\Delta m_{ij}^2 = \Delta m_i^2 - \Delta m_j^2$). From various experiments [7] we have found the values of some of these parameters. These experiments include the observation of solar neutrinos produced in the sun's core, atmospheric neutrinos produced by cosmic-ray interactions with the earth's atmosphere, neutrinos produced at nuclear reactors and neutrinos produced in an accelerator. At present, what it is known, combining the results from the different experiments is [5]:

- $\sin^2(2\theta_{12}) = 0.857\ ^{+0.023}_{-0.025}$
- $\Delta m_{21}^2 = (7.50\ ^{+0.19}_{-0.20}) \times 10^{-5}$ eV^2
- $\sin^2(2\theta_{23}) > 0.95$
- $|\Delta m_{32}^2| = (2.32\ ^{+0.12}_{-0.08}) \times 10^{-3}$ eV^2 (the sign of Δm_{32}^2 is not known)

- $\sin^2(2\theta_{13}) = 0.098 \pm 0.013$

The unknown neutrino parameters are:

- the sign of Δm^2_{32}
- the absolute neutrino mass
- δ
- α_1, α_2

Thus obtaining a better understanding of neutrino properties is essential for physics. Measuring the CP violation phase δ would help understand better the baryon asymmetry in the universe [8]. Some theories have also postulated that very massive neutrinos could be a candidate for dark matter. New machines need to be built in order to make more precise observations of neutrino properties. One of these machines is the Neutrino Factory. In a Neutrino Factory it is possible to generate intense, high-energy neutrino and antineutrino beams, which are required to make precision measurements of the neutrino parameters, probe the CP violation phase and the neutrino mass hierarchy. These beams will also be useful should exist a fourth generation of sterile-neutrino or non-standard interactions [9].

3) The Neutrino Factory

A Neutrino Factory (NF) is a machine able to produce 10^{21} neutrinos per year. The construction of such a machine is currently under study. Its main objective is to do precision measurements of the neutrino oscillation parameters and probe physics beyond the Standard Model, such as CP-invariance violation. In a conventional beam the proton beam hits a target to produce pions. These pions are focused and then decaying in either a muon neutrino (ν_μ) beam or a muon anti-neutrino ($\bar{\nu}_\mu$) beam ($\pi^+ \rightarrow \mu^+ + \nu_\mu$). The beam also contains a small electron neutrino (ν_e) or electron anti-neutrino ($\bar{\nu}_e$) contamination, due to the decay of muons. In a conventional beam it is also possible to have only one sign at a time with the use of a horn. The horn captures only positive or only negative pions. NF is different from a conventional beam. The basic concept of a Neutrino Factory is to take a high-intensity, high-energy neutrino beam with a very well-known energy spectrum. In a Neutrino Factory neutrinos come from the decay of muons ($\mu^+ \rightarrow e^+ + \bar{\nu}_\mu + \nu_e$) and they are 50% $\nu_\mu\,(\bar{\nu}_\mu)$ – 50% $\bar{\nu}_e\,(\nu_e)$. Two other machines aimed at providing precision measurements of the neutrino parameters are also under study: the Superbeam and the Betabeam. The Superbeam is the upgrade of a conventional beam to much higher intensity, where neutrinos ν_μ or $\bar{\nu}_\mu$ come from the decay of pions. In the Betabeam pure ν_e and $\bar{\nu}_e$ beams are produced through beta-decay of radioactive ions such as ^{6}He and ^{18}Ne.

The NF baseline (see Figure 3) parameters taken into consideration are described in Table 3.

Parameter	Value
Muon total energy	25 GeV
Production straight muon decays in 10^7 s (in target direction)	10^{21}
Maximum RMS angular divergence of muons in production straight	0.1/γ
Distance to intermediate baseline detector	3000-5000 km
Distance to long base line detector	7000-8000 km

Table 3: Parameters characterising the muon beam produced by the accelerator facility. Muon decays are a total for all signs and detector baselines.

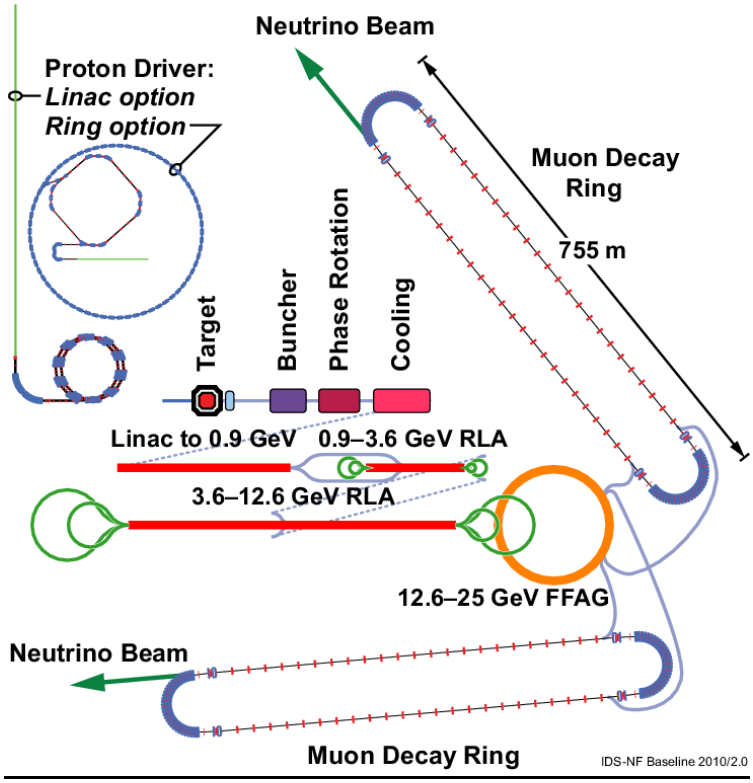

Figure 3: Schematic drawing of the baseline for the Neutrino Factory accelerator complex. [6]

3.1) Proton Driver

The proton driver at the Neutrino Factory is required to deliver a proton beam of 4MW at a repetition rate of 50 Hz to the pion production target. The proton beam energy is in the multi-GeV range in order to maximize the pion yield. It is important for the efficiency of the Neutrino Factory that the proton beam consists of 3 very short bunches separated by 120 μs where each bunch will become a different muon bunch train later. In order to create these very short bunches a bunch compression system is required, which will be able to manage the very strong space-charge forces. For this purpose, three possible designs of proton drivers have been considered in three of the world's proton accelerator laboratories: CERN, FNAL [10] and RAL [11].

A typical proton driver scheme consists of: an H^- ion source, a radio-frequency quadrupole (RFQ), a chopper and a linear accelerator. In the CERN scenario a HP-SPL (high power – superconducting proton linac) delivers a beam of 5 GeV, consisting

of 10^{14} protons with a repetition rate of 50 Hz. The beam needs to be worked out before hitting the target. It must be formed into bunches with the proper number of bunch per train and bunch separation. For that purpose an accumulator ring and a compressor ring are used. The accumulator forms the beam in 120 ns long bunches without using a RF system and the compressor is used for fast phase rotation. In the FNAL scenario a similar layout is used and in the RAL scenario a RCS (rapid cycling synchrotron) and/or a FFAG (fixed-field alternating-gradient [12]).

3.2) Target

The produced proton beam of 4 MW will collide with a liquid mercury (Hg) jet target, operating in a solenoid, pion capture channel (see Figure 4). The target has also to be into a solenoid magnetic field in order to have both signs of pions captured. The solenoid field of 20 T, tapers down to 1.5 T over a distance of 15 m. For the liquid mercury jet (Z = 80, atomic weight = 200.6, density ρ = 13.5 g/cm^3) the parameters chosen are diameter d = 8 mm, flowing speed u = 20 m/s, volume-flow rate 1.0 L/s and a mechanical power of 2.7 kW. The outcome of the collision of the beam on the target will be the production of pions and kaons.

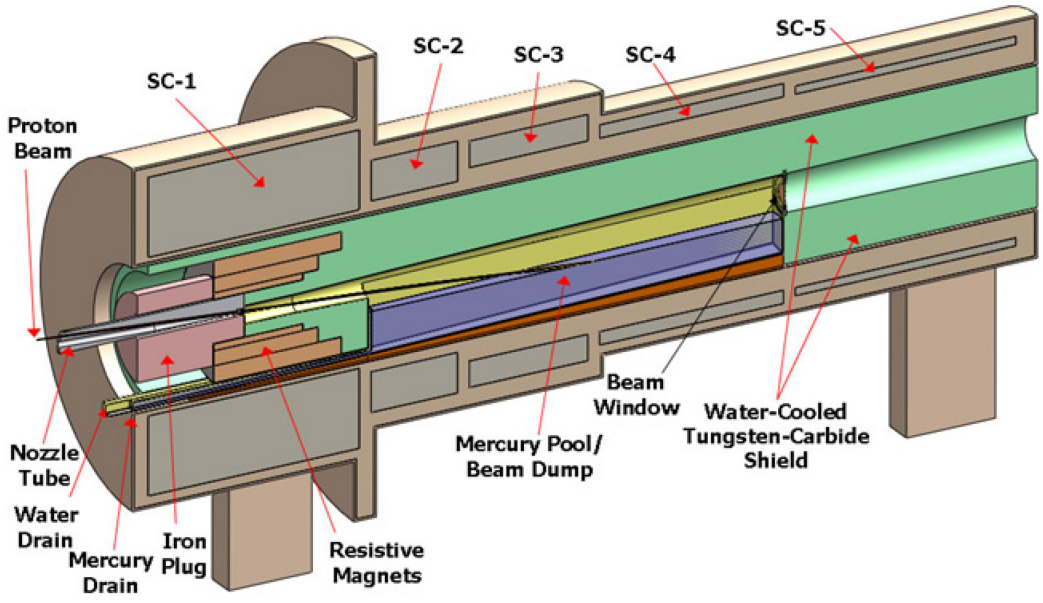

Figure 4: Baseline target system concept. SC-n are the superconducting magnets.

3.3) Muon Front-End

The muon front – end consists of a pion decay channel and longitudinal drift, followed by an adiabatic buncher, a phase-rotation system, and an ionization-cooling channel.

3.3.1) Decay and longitudinal drift

The produced muons beam from pions decay needs to be reconfigured before it enters the accelerator complex. In Figure 5 a plot of the distribution of the muons momentum vs time is presented for different regions of the Neutrino Factory.

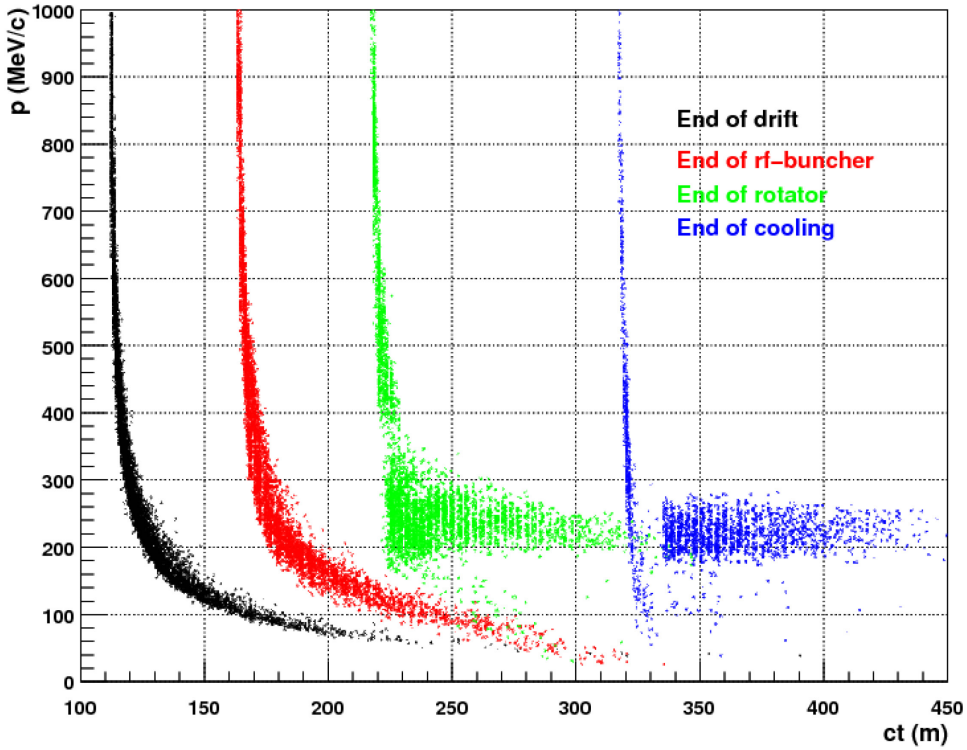

Figure 5: This plot shows the way the muon phase space is transformed by the bunching rotator and cooling systems of the front-end.

After the target the beam pipe radius increases from 0.075 m to 0.3 m, so that the secondary-pion beam with a large energy spread can be captured. The initial short proton bunch produces a short pion bunch that develops a time - energy correlation in the decay channel, as it drifts longitudinally from the target. The longitudinal drift length is L_D = 57.7 m.

3.3.2) Buncher

In the buncher the muons coming from the decay drift are turned into a train of bunches, by a sequence of RF cavities [13].

Positive muons are separated from negative muons in different bunches. The bunch train is less than 80 m long. The buncher is equipped with 0.4 – 0.5 m long RF cavities, placed within 0.75 m long cells. The frequency of the RF cavities is decreased from 320 MHz (λ = 0.94 m) to 230 MHz (λ = 1.3 m) and they have an

increasing gradient from 4 MV/m at the start to 9 MV/m at the end of the buncher, which is 33 m long.

3.3.3) **Rotator**

The individual muon bunches, entering the rotator, have their respective energy changed and so at the end all bunches acquire the same energy. For this to be achieved the phase rotation section decelerates the early in time high energy bunches and accelerates the late in time low energy bunches.

The rotator is 42 m long, equipped with 0.5 m long RF cavities in 0.75 m long cells, with decreasing frequency from 230.2 MHz to 202.3 MHz and a gradient of 12 MV/m. In total 56 RF cavities will be used. In the rotator the bunches are formed around a central bunch. The bunches before the central one are decelerated, while those after the central one are accelerated, in order to have bunches with equal energy for both muon species.

It is important to notice that the muon collection, bunching and rotation system is able to form the bunch for both sign with similar performance.

3.3.4) **Ionization-cooling channel**

The ionization-cooling channel reduces the transverse emittance of the bunches, so that the number of muons that can be transmitted into the acceptance of the acceleration system is increased. In the absorbers the total momentum of the bunches is reduced and in the RF cavities the longitudinal momentum is increased.

It is equipped with identical 1.5 m long cells, where in each cell there are two 0.5 m long RF cavities, with 1.1 cm thick LiH discs (energy loss material) at the end of each cavity and 0.25 m spacing between the cavities. In total, it is 75 m long, consisting of 50 cells. The cells are equipped with two solenoid coils with opposite polarity. To determine whether a particle will be accepted by the downstream accelerator systems, some acceptance criteria have been defined. A particle is within the acceptance of the machine if the transverse amplitude squared is less than 0.03 m, the longitudinal amplitude squared less than 0.15 m and the momentum is between 100 MeV/c and 300 MeV/c. Particles with larger amplitudes are removed by the acceptance criteria we set.

3.4) The muon accelerator complex

After the muon beam exits the buncher, rotator and cooling sections it is ready to be injected in the rest of the accelerator complex. The acceleration process is happening in three stages. At first, the muon beam enters in a linac (linear accelerator [14]) where it is accelerated to the energy of 0.9 GeV. The muon beam runs through the linac only one time. Then it continues in two recirculating linear accelerators (RLAs) [15] where its energy increases to 3.6 GeV and 12.6 GeV respectively. In the RLAs the beam makes multiple passes through the RF cavities (4.5 passes).

The muons are accelerated to a final energy of 25 GeV by a FFAG and then they are injected in two storage rings, where they decay into neutrinos. In order to maximize the number of muons decaying in the direction of the detectors the lengths of the straight sections, form a big fraction of the decay ring circumference. The two detectors are located, one 7000 – 8000 km away from the source and the other 2500 – 5000 km away from it, so that the neutrinos will have the time to oscillate from one flavour to another. It is important to have both detector baselines, because at long distance CP violation effects cancel out with matter effects, so that the far detector is used for the measurement of θ_{13} angle. With the combination of both short and long distance baselines other parameters can be probed.

4) The 44 – 88 MHz lattice

The front – end design described above appears to have specific problems. In particular, the presence of a magnetic field can cause a breakdown in the RF cavities and that forces the system to operate at a lower gradient field. In order to achieve higher efficiency in the muon front – end of the Neutrino Factory, several alternative lattices have been proposed. One of them is the 44 – 88 MHz lattice, described below.

The 44 – 88 MHz lattice in contrary with the previous front – end design, is capturing the muons in a single bunch-to-bucket mode. In the beam simulation of this lattice we use an 8 GeV proton beam, which collides with a Hg-jet target as in the baseline configuration. The Hg target is immersed in a 20 T solenoid field which is reduced to 1.75 T over 12.2 m distance. The pions and kaons produced after the collision are entering a 30 m long decay channel with 1.8 T solenoid field. Then, they continue in a 30 m long phase rotation channel with 1.8 T solenoids and thirty 44 MHz cavities, each 1 m long, set at a -90^{o} phase and 2 MV/m gradient. After the rotation channel there is a cooling section with 44 MHz cavities and hydrogen absorbers, followed by an accelerating section with 44 MHz cavities and finally an additional cooling section with 88 MHz cavities.

5) Simulation codes

For the purpose of this project we used the ICOOL [7] simulation code. ICOOL is a 3-dimensional tracking program that simulates particle-by-particle propagation through materials and electromagnetic fields. The particles are tracked through a sequence of regions that have a fixed length in z. A region is cylindrical in shape and may be subdivided radially. Every region has a specified material and field type associated with it. Regions can be grouped in cells and a separated cell field can be superimposed over the region fields when tracking in code. The program generates several output files over which the one called for009.dat contains all the particles variables such as position, momentum and time. Analysis of the results is performed using ECALC9F [8]. ECALC9F is an ICOOL utility that reads the file for009.dat and performs a standardized emittance and lattice function analysis.

6) Project

The aim of this project is to re-evaluate the muon front – end scenario of the Neutrino Factory which used 44-88 MHz RF cavities. In this project the ionization – cooling channel was not included, due to lack of time. For this purpose we followed the procedure described below.

The baseline of the procedure of the project was: Firstly, becoming familiar with coding in C++ and using ROOT [9] by working out samples of similar code. Secondly, starting step by step writing a code that can use the output files from ICOOL (for009.dat) and the routine ECALC9F (ecalc9f.dat) and elaborate them in a way that we could draw some conclusions for the muons coming out of the decay drift and rotation system. By compiling the code, histograms and graphs were produced using ROOT as graphical interface. The most important graphs and results are presented below. The procedure was repeated and different results were obtained by applying different conditions in ICOOL, such as using different lengths of drifts for the pions coming out of the Hg target (10 m, 20 m, 30 m) and using different acceptance cuts. In Table 4 are given the acceptance and momentum criteria we used to calculate the number of muons n, n0, n1 and n2. Finally, a combined code was written which included all the above. The code is given in Annex.

Cuts	n	n0	n1	n2
$100 < p_z < 300$ MeV/c	N	Y	Y	Y
$A_L = 150$ mm	N	N	Y	Y
$A_T = 15$ mm	N	N	Y	N
$A_T = 30$ mm	N	N	N	Y

Table 4: The momentum and acceptance cuts, where Y and N stand for cut used and cut not used respectively. A_L is the longitudinal acceptance, A_T is the transverse acceptance and p_z is the longitudinal momentum.

7) Simulation Results

Figure 6 shows n and n0 for different drifts. Figure 7 gives the number of muons n1 and n2 for the three drift lengths studied.

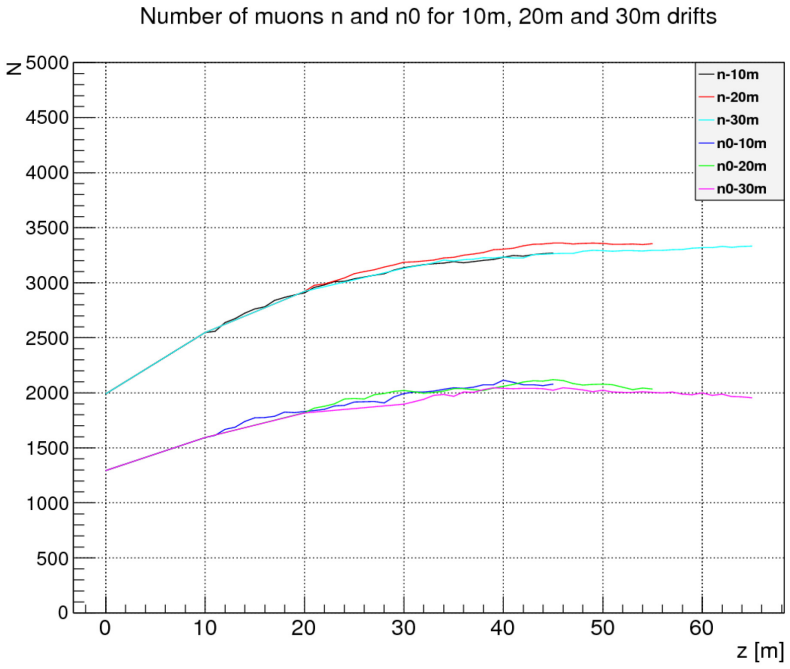

Figure 6: Number of muons n and n0 as function of z for three drift lengths.

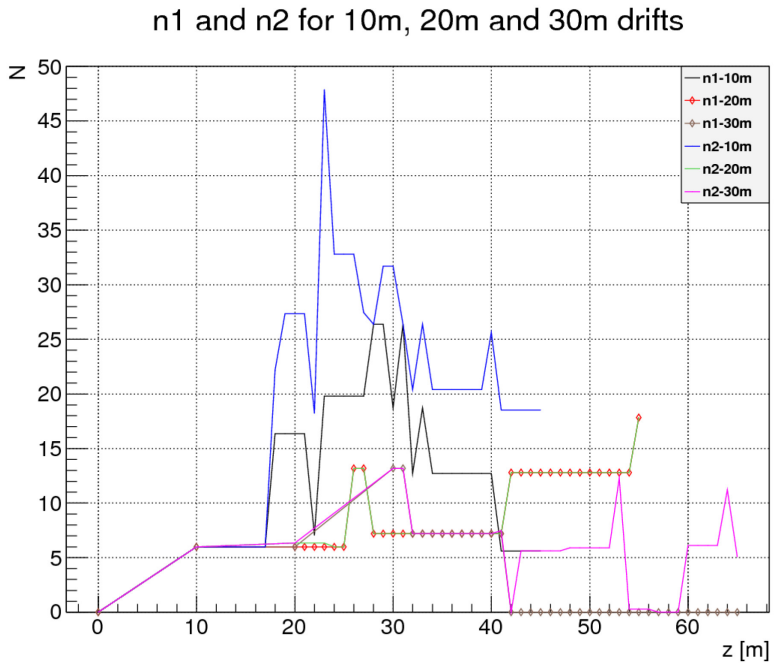

Figure 7: Number of muons n1 and n2 as function of z for three drift lengths.

We observe that the number of muons n1 and n2 is not increasing when the drift before the rotator is changing from 10 m to 20 m, but it is doubled for a 30 m drift. The number of muons n1 and n2, after entering the rotator, is multiplied by 3 for a 20 m drift; it is bigger for a 10 m drift, but in that case it is falling after 20 m in the rotator; finally it is decreasing for a 30 m drift. We conclude that the 20 m drift seems optimum, but additional study may improve this number. As a result the rotator's RF phase needs to be adjusted and future study is needed.

Figures 8 & 9 show the momentum-time distribution for the muons in a 10 m drift for the first and the last region respectively. The first region is in the beginning of the drift and the last region is at the end of the rotator.

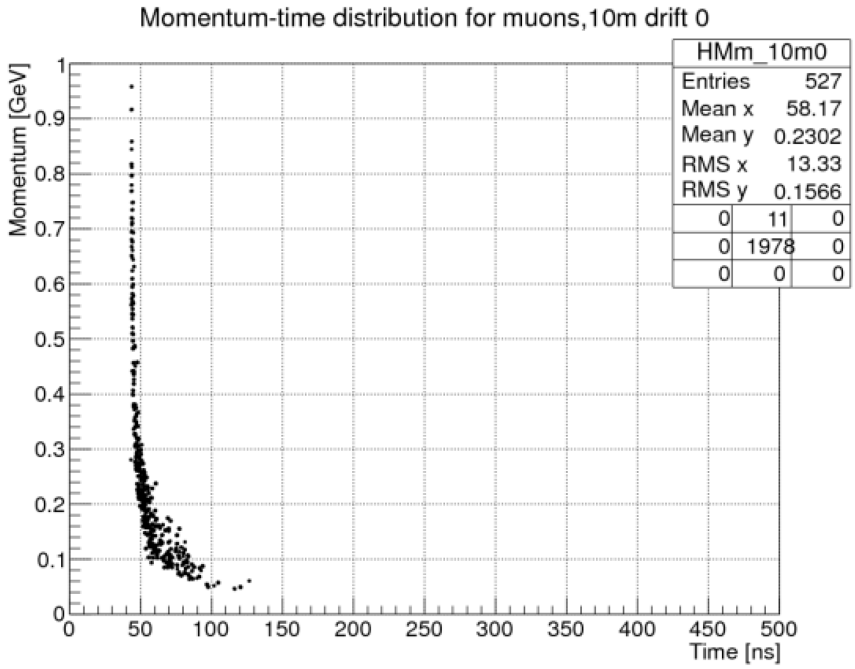

Figure 8: Momentum-time distribution of muons in the first region and 10 m drift.

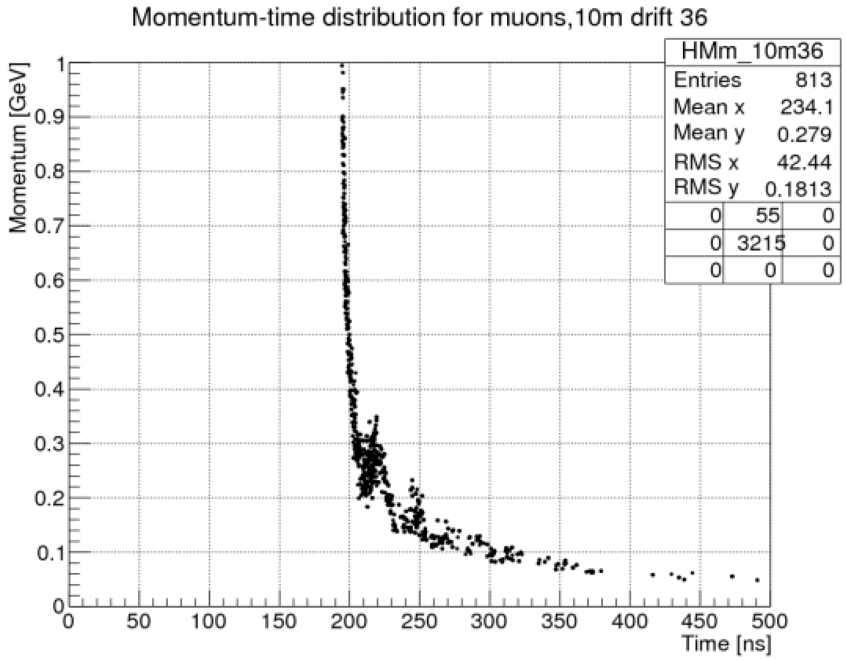

Figure 9: Momentum-time distribution of muons in last region and 10 m drift.

We also performed the simulation with 0.5 m spacing between the RF cavities in the rotator (Figures 10-11). In the results, n2 represents 1/50 of the initial muons that are falling in acceptance cuts, where only 1/100 muons were accepted for the lattice with no space between the cavities. The momentum spread has been reduced from 11% to 1% and the total number of cavities has also been reduced from 30 to 14 cavities, a factor which would help lowering the cost of the front-end.

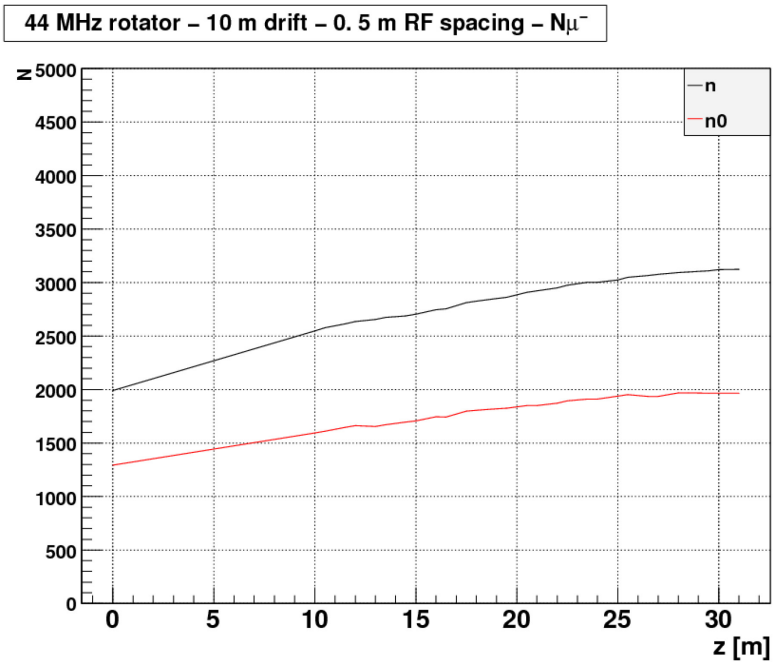

Figure 10: Number of muons n an n0 as a function of z for a configuration with 0.5 m spacing between cavities.

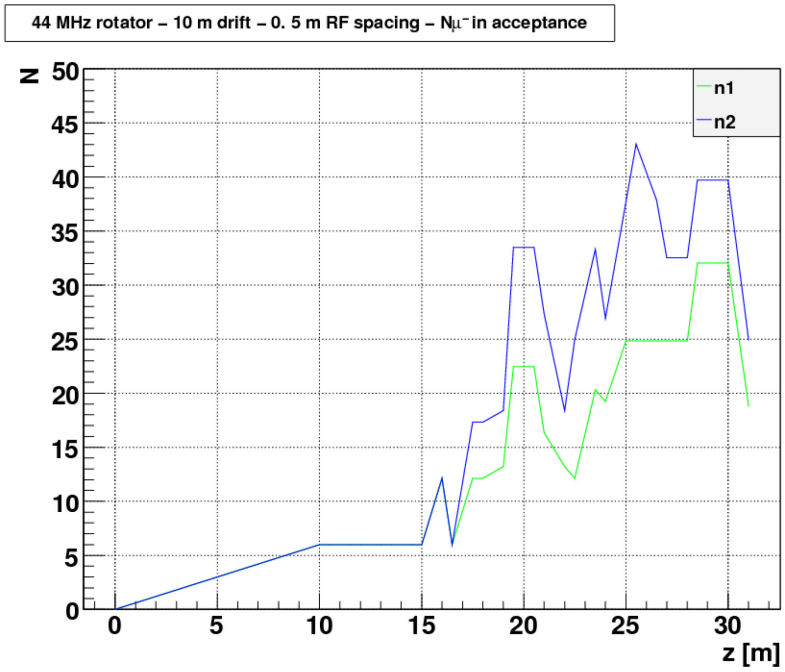

Figure 11: Number of muons n1 and n2 as function of z for a configuration with 0.5 m spacing between cavities.

Figure 12 and 13 present the muon momentum and time spreads for a configuration of a 10 m drift and a 30 m rotator. Figure 12 has the momentum spread in the beginning of the drift, which is 11% and Figure 13 has the momentum spread at the end of the rotator, which is reduced to 2-3% compared to the initial

one. Finally, Figure 14 shows the magnetic field on axis for a 10 m drift and a 30 m long rotator.

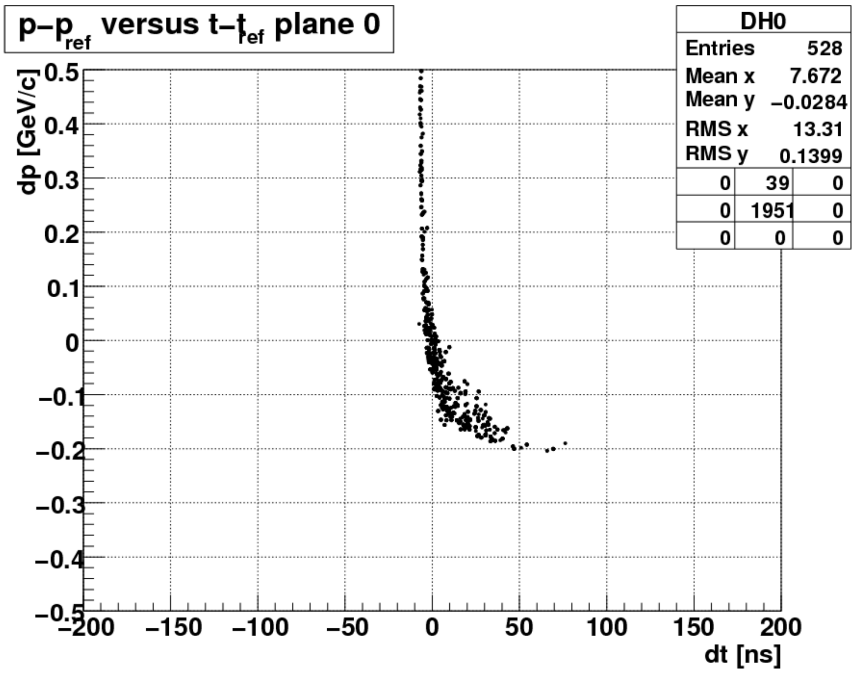

Figure 12: The muon momentum and time spread for a 10 m drift and 30 m rotator before the drift.

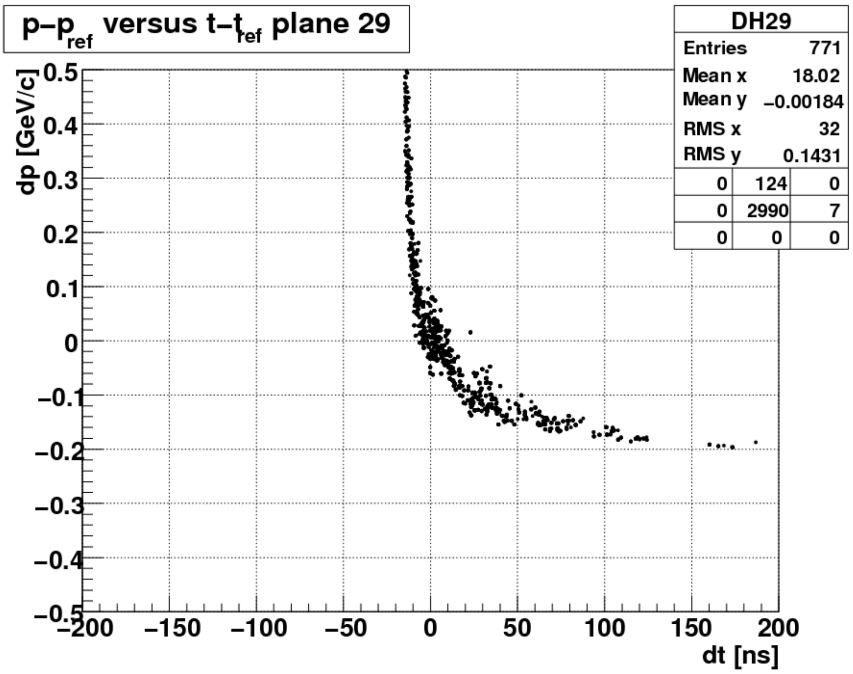

Figure 13: The muon momentum spread for a 10 m drift and 30 m rotator at the end of the rotator.

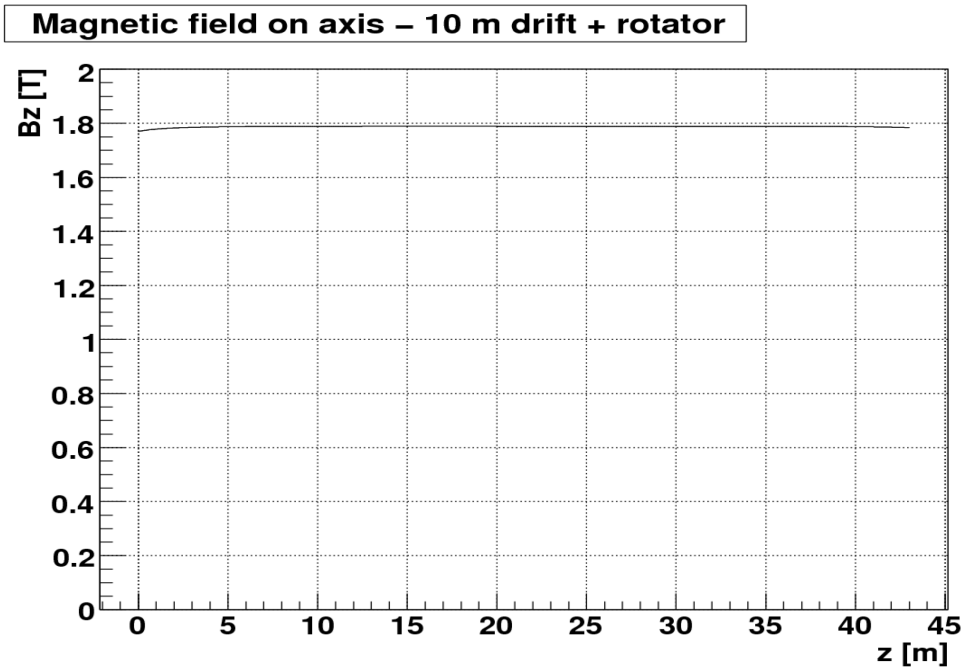

Figure 14: The magnetic field on axis for a 10 m drift and a 30 m long rotator.

8) Conclusion

The 44 - 88 MHz lattice configuration has been studied for the rotator section using the IDS-NF baseline beam and ICOOL. Different drift lengths (without change of the lattice configuration in comparison with previous studies) have been studied showing a very low performance in comparison with past studies [10]. By increasing the space between the RF cavities, a gain in the number of muons captured in the performance was shown. Future optimization studies with an increased RF bucket height, different RF cavities spacing, re-adjustment of the phase and reference particle time, will be done.

9) Acknowledgments

The project was supervised primarily by Gersende Prior and secondly by Simone Gilardoni from CERN, Geneva, Switzerland. For the purpose of the project, new computing skills and knowledge in physics had to be obtained. More specifically during the project, particle and accelerator physics, programming in Linux terminal, writing code in C++, making graphs by using ROOT, using softwares such as Ghostview, PostScript/PDF Viewer and Xfig and the ICOOL simulation program are some of the new things that needed to be learned. I would like to express my deep appreciation to my supervisor Gersende Prior for being patiently guiding me every time a problem came up, for devoting a lot of her time to teach me all the above and for pleasantly passing to me the knowledge of physics and computing that I never had before during the past years of my studies. Working at CERN is by far the best experience I ever had, concerning my studies. All the amazing new things I saw and

learned and especially all the open-minded and interesting persons I met will neverbe forgotten. CERN showed me that people from completely different cultures around the whole world, can work together having wonderful cooperation under a same objective, unravelling the mysteries of the universe.

10) Glossary and Useful links

(1)	CERN	European Organization for Nuclear Research http://public.web.cern.ch/public/
(2)	Higgs mechanism	According to the Higgs mechanism particles gain mass by interacting with the Higgs field, in particular with the exchange of a Higgs boson. The search for the Higgs boson is currently performed at the LHC, at CERN and latest results indicate that we are very close to finding it. http://press.web.cern.ch/press/PressReleases/Releases2012/PR17.12E.html
(3)	Lepton number	The lepton number (L) is an additive quantum number, which we attribute to every lepton. All leptons have L = +1 and all antileptons have L = -1. Particles that are not leptons have zero lepton number. In a particle interaction the sum of the lepton number is conserved. For each lepton family a separate lepton number is defined (electronic number L_e, muonic number L_μ and tauonic number L_τ). These three lepton numbers are also conserved.
(4)	Quark mixing	As in lepton mixing there is quark mixing (CKM matrix) that specifies the mismatch of quantum states of quarks when they propagate freely and when they take part in the weak interaction.
(5)	Charge parity symmetry	A parity transformation is the flip in the sign of the spatial coordinates. In a three dimensional space it is the flip of the sign of all three coordinates. Parity is violated in weak interactions. $\mathrm{P}: \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} -x \\ -y \\ -z \end{pmatrix}$ C-symmetry implies that the physical laws under a charge-conjugation transformation remain unchanged. For example, if a charge q was to be replaced with a charge –q and the direction of the electric and magnetic fields were reversed, laws of electromagnetism would remain invariant. Weak interactions involving neutrinos violate C-symmetry. The CP symmetry or charge parity symmetry is the combination of C symmetry and P symmetry. It states

that if a particle is subjected to the C symmetry and the P symmetry subsequently the law of physics remain unchanged. For instance direct CP symmetry violation is described in the CMK matrix of quark mixing and the PMNS matrix of neutrino mixing by the mean of a complex phase.

(6) Neutrinoless double beta decay

Beta decay is a type of radioactive decay in which a beta particle (an electron or a positron) is emitted from an atom.

- Beta minus decay: $n^0 \rightarrow p^{+1} + e^{-1} + \bar{\nu}_e$
- Beta plus decay: $p^{+1} \rightarrow n^0 + e^{+1} + \nu_e$

In double beta decay two neutrons in the nucleus convert in two protons, releasing two electrons and two electron antineutrinos. The process into which two protons convert in two neutrons, with emission of two electron neutrinos and absorption of two orbital electrons is also theoretically possible however has been only rarely observed. If the neutrino is a Majorana particle and at least one type of neutrino has non-zero mass then it is possible for neutrinoless double-beta decay to occur (the two neutrinos annihilate each other, or equivalently, one nucleon absorbs the neutrino emitted by another nucleon of the nucleus).
http://en.wikipedia.org/wiki/Double_beta_decay

(7) Neutrino experiments

List of neutrino experiments.
http://en.wikipedia.org/wiki/List_of_neutrino_experiments

(8) Matter-antimatter asymmetry

It has been observed that matter is dominant over antimatter in the Universe. This asymmetry of matter and antimatter happened during the first moments of the Big Bang and is mainly attributed to the violation of the CP symmetry. Due to CP violation, the laws of physics affected differently matter over antimatter in the early age of the universe and this caused the observed matter-antimatter asymmetry. Thus, the observation of CP violation in neutrino sector is a very important discovery.
http://en.wikipedia.org/wiki/Matter-antimatter_asymmetry#Origin_and_asymmetry

(9) Non-standard interactions

These are neutrino interactions predicted by physics beyond the Standard Model that give a different explanation about neutrino oscillations. A Neutrino Factory will be able to search for this kind of interactions.
http://theophys.kth.se/tepp/research_non-standard_neutrino_interactions.html

(10) FNAL — Fermilab, Fermi National Accelerator Laboratory.
http://www.fnal.gov/

(11) RAL — Rutherford Appleton Laboratory
http://www.stfc.ac.uk/About+STFC/51.aspx

(12) FFAG — In a Fixed-Field Alternative Gradient accelerator particles are accelerated in a circular orbit. It is equipped with strong focusing alternating-gradient quadrupole fields for the beam confinement and with a dipole bending magnet to configure the particles in a close circular orbit. There are two types of FFAG discerned by the type of beam focusing in the transverse direction of motion. A scaling FFAG is made with non-chromatic beam optics and uses large non-linear fields for the beam focusing. In a non-scaling FFAG the fields are linear.
http://en.wikipedia.org/wiki/FFAG

(13) RF cavity — A radio frequency (RF) cavity (see Figure 19) is a special type of resonator, consisting of a closed (or largely closed) metal structure that confines electromagnetic fields in a frequency region of the spectrum. The size of the RF depends on the wave length of the field. In accelerators they are used to accelerate charged particles with a longitudinal or transverse electric field.

Figure 15: An RF cavity. Photograph taken at the exhibition in SM18 building at CERN.

(14) Linac — In a Linear Particle accelerator charged particles, produced by a source, are accelerated in a straight beam line by oscillating electric potentials (superconducting RF cavities, iron-shielded solenoids), and then, they are directed in a target.

http://en.wikipedia.org/wiki/Linac

(15) RLAs — A Recirculating Linear Accelerator (RLA) is equipped with superconducting RF cavities for the beam acceleration and quadrupoles for transverse focusing of the beam. In an RLA the beam is recirculating several times in the beam pipe and each time it gets an energy boost. For the re-injection of the beam in the linear part of the RLA, energy-dependant droplet arcs are used.

11) Reference

[1] S. Choubey *et al.* (The IDS-NF collaboration), "International Design Study for the Neutrino Factory: Interim Design Report", IDS-NF-020 (2011)
https://www.ids-nf.org/

[2] P. Gruber. "Ionization Cooling for a Neutrino Factory", Ph.D. dissertation (2001)

[3] Standard Model of Elementary Particles
http://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg

[4] Neutrino mass spectrum
http://www.tifr.res.in/TIFR_Science_News/faq.php?telid=8

[5] J. Beringer *et al.* (Particle Data Group), PRD 86 010001 (2012)

[6] Neutrino Factory baseline
https://www.ids-nf.org/wiki/FrontPage/Accelerator/Documentation?action=AttachFile&do=view&target=IDS-NuFact-110109-150.png

[7]R.C. Fernow et al., "ICOOL reference Manual, version 3.20" Brookhaven National Laboratory (2009)

[8] R. C. Fernow, Physics analysis performed by ECALC9F MUCOOL-NOTE-0280

[9] R. Brun, F. Rademakers, "ROOT - An Object Oriented Data Analysis Framework", Proceedings AIHENP'96 Workshop, NIM A 389 (1997) p. 1-86
http://root.cern.ch/

[10] A. Lombardi, "A 40-80 MHz system for phase rotation and cooling", CERN-NUFACT note 034 (2000)

12) Annex

The Root code developed for the extraction of the results is presented below.

```
//Code for the 44 - 88 MHz lattice, in 10m, 20m and 30m drift

#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <strstream.h>
#include <cstdio>
#include <TMath.h>

using namespace std;

void code_10m_20m_30m_drift(char outplots[] = "code_10m_20m_30m_drift.ps") {

 std::string filein1 = "for009_10m.dat";
 std::string filein3 = "for009_20m.dat";
 std::string filein5 = "for009_30m.dat";
 std::string filein2 = "ecalc9f_10m.dat";
 std::string filein4 = "ecalc9f_20m.dat";
 std::string filein6 = "ecalc9f_30m.dat";

 gStyle->SetOptStat(111111);
 TCanvas c0("c1", "c1",1000,800);
 c0.Print(Form("%s[",outplots));
 c0.SetGrid();

 //Creating a multiple historgam for the momentum=f(time) distribution of muons
 TH2F *HMm_10m[37],*HPm_10m[37];
 TH1F *HMY_10m[37],*HPY_10m[37];
 TH2F *HMm_20m[38],*HPm_20m[38];
 TH1F *HMY_20m[38],*HPY_20m[38];
 TH2F *HMm_30m[39],*HPm_30m[39];
 TH1F *HMY_30m[39],*HPY_30m[39];

 char hist1_10m[50],title1_10m[60];
 char hist2_10m[50],title2_10m[60];
 char hist3_10m[50],title3_10m[60];
 char hist4_10m[50],title4_10m[60];
 char hist1_20m[50],title1_20m[60];
 char hist2_20m[50],title2_20m[60];
 char hist3_20m[50],title3_20m[60];
 char hist4_20m[50],title4_20m[60];
 char hist1_30m[50],title1_30m[60];
 char hist2_30m[50],title2_30m[60];
 char hist3_30m[50],title3_30m[60];
 char hist4_30m[50],title4_30m[60];
 Int_t maxplane_10m;
 maxplane_10m = 37;
 Int_t maxplane_20m;
 maxplane_20m = 38;
 Int_t maxplane_30m;
 maxplane_30m = 39;

 //10m drift
```

```
for(Int_t i = 0; i < maxplane_10m; i++){
 sprintf(hist1_10m,"HMm_10m%d",i);
 sprintf(title1_10m,"Momentum-time distribution for muons,10m drift %d",i);
 HMm_10m[i] = new TH2F(hist1_10m, title1_10m, 1000,0.,500.,1000,0.,1.);
}
for(Int_t i = 0; i < maxplane_10m; i++){
 sprintf(hist2_10m,"HPm_10m%d",i);
 sprintf(title2_10m,"Momentum-time distribution for pions,10m drift %d",i);
 HPm_10m[i] = new TH2F(hist2_10m, title2_10m, 1000,0.,250.,1000,0.,1.);
}
for(Int_t i = 0; i < maxplane_10m; i++){
 sprintf(hist3_10m,"HMY_10m%d",i);
 sprintf(title3_10m,"Yield of muons,10m drift %d",i);
 HMY_10m[i] = new TH1F(hist3_10m, title3_10m, 50,0.,5.);
}
for(Int_t i = 0; i < maxplane_10m; i++){
 sprintf(hist4_10m,"HPY_10m%d",i);
 sprintf(title4_10m,"Yield of pions,10m drift %d",i);
 HPY_10m[i] = new TH1F(hist4_10m, title4_10m, 50,0.,5.);
}

//20m drift
for(Int_t i = 0; i < maxplane_20m; i++){
 sprintf(hist1_20m,"HMm_20m%d",i);
 sprintf(title1_20m,"Momentum-time distribution for muons,20m drift %d",i);
 HMm_20m[i] = new TH2F(hist1_20m, title1_20m, 1000,0.,600.,1000,0.,1.);
}
for(Int_t i = 0; i < maxplane_20m; i++){
 sprintf(hist2_20m,"HPm_20m%d",i);
 sprintf(title2_20m,"Momentum-time distribution for pions,20m drift %d",i);
 HPm_20m[i] = new TH2F(hist2_20m, title2_20m, 1000,0.,300.,1000,0.,1.);
}
for(Int_t i = 0; i < maxplane_20m; i++){
 sprintf(hist3_20m,"HMY_20m%d",i);
 sprintf(title3_20m,"Yield of muons,20m drift %d",i);
 HMY_20m[i] = new TH1F(hist3_20m, title3_20m, 50,0.,5.);
}
for(Int_t i = 0; i < maxplane_20m; i++){
 sprintf(hist4_20m,"HPY_20m%d",i);
 sprintf(title4_20m,"Yield of pions,20m drift %d",i);
 HPY_20m[i] = new TH1F(hist4_20m, title4_20m, 50,0.,5.);
}

//30m drift
for(Int_t i = 0; i < maxplane_30m; i++){
 sprintf(hist1_30m,"HMm_30m%d",i);
 sprintf(title1_30m,"Momentum-time distribution for muons,30m drift %d",i);
 HMm_30m[i] = new TH2F(hist1_30m, title1_30m, 1000,0.,600.,1000,0.,1.);
}
for(Int_t i = 0; i < maxplane_30m; i++){
 sprintf(hist2_30m,"HPm_30m%d",i);
 sprintf(title2_30m,"Momentum-time distribution for pions,30m drift %d",i);
 HPm_30m[i] = new TH2F(hist2_30m, title2_30m, 1000,0.,300.,1000,0.,1.);
}
for(Int_t i = 0; i < maxplane_30m; i++){
 sprintf(hist3_30m,"HMY_30m%d",i);
 sprintf(title3_30m,"Yield of muons,30m drift %d",i);
```

```
  HMY_30m[i] = new TH1F(hist3_30m, title3_30m, 50,0.,5.);
}
for(Int_t i = 0; i < maxplane_30m; i++){
  sprintf(hist4_30m,"HPY_30m%d",i);
  sprintf(title4_30m,"Yield of pions,30m drift %d",i);
  HPY_30m[i] = new TH1F(hist4_30m, title4_30m, 50,0.,5.);
}

//Declaring the Variables
char header_10m[1000];
char header_20m[1000];
char header_30m[1000];
char header3_10m[1000];
char header3_20m[1000];
char header3_30m[1000];
Int_t maxline1_10m = 36659; // for009_10m.dat
Int_t maxline1_20m = 37732; // for009_20m.dat
Int_t maxline1_30m = 38593; // for009_30m.dat
Int_t maxline3_10m = 50; // ecalc9f_10m.dat
Int_t maxline3_20m = 51; // ecalc9f_20m.dat
Int_t maxline3_30m = 52; // ecalc9f_30m.dat
Int_t oldreg_10m;
Int_t oldreg_20m;
Int_t oldreg_30m;
Int_t evt_10m,par_10m,typ_10m,flg_10m,reg_10m;
Int_t evt_20m,par_20m,typ_20m,flg_20m,reg_20m;
Int_t evt_30m,par_30m,typ_30m,flg_30m,reg_30m;
Int_t line1_10m;
Int_t line1_20m;
Int_t line1_30m;
Int_t line3_10m;
Int_t line3_20m;
Int_t line3_30m;
Int_t icount_10m;
Int_t icount_20m;
Int_t icount_30m;
Double_t time_10m;
Double_t x_10m,y_10m,z_10m;
Double_t px_10m,py_10m,pz_10m;
Double_t bx_10m,by_10m,bz_10m;
Double_t wt_10m;
Double_t ex_10m,ey_10m,ez_10m;
Double_t arclength_10m,polx_10m,poly_10m,polz_10m;
Double_t momentum_10m;
Double_t num_10m[268];
Double_t zpos_10m[268];
Double_t time_20m;
Double_t x_20m,y_20m,z_20m;
Double_t px_20m,py_20m,pz_20m;
Double_t bx_20m,by_20m,bz_20m;
Double_t wt_20m;
Double_t ex_20m,ey_20m,ez_20m;
Double_t arclength_20m,polx_20m,poly_20m,polz_20m;
Double_t momentum_20m;
Double_t num_20m[268];
Double_t zpos_20m[268];
Double_t time_30m;
```

```
Double_t x_30m,y_30m,z_30m;
Double_t px_30m,py_30m,pz_30m;
Double_t bx_30m,by_30m,bz_30m;
Double_t wt_30m;
Double_t ex_30m,ey_30m,ez_30m;
Double_t arclength_30m,polx_30m,poly_30m,polz_30m;
Double_t momentum_30m;
Double_t num_30m[268];
Double_t zpos_30m[268];
Double_t regn_10m,Z_10m,Bz_10m;
Double_t eperp_10m,elong_10m,e6D_10m;
Double_t Ldim_10m,Pzavg_10m,beta_10m,alpha_10m;
Double_t betaL_10m,alphaL_10m,n0_10m,n1_10m,n2_10m;
Double_t Lcon_10m,sigmaE_10m,sigmaT_10m,corrE_10m,corrT_10m;
Double_t sigmaE_c_10m,xavg_10m,yavg_10m;
Double_t Dx_10m,Dy_10m,Dr_10m,Dr2_10m;
Double_t regn_20m,Z_20m,Bz_20m;
Double_t eperp_20m,elong_20m,e6D_20m;
Double_t Ldim_20m,Pzavg_20m,beta_20m,alpha_20m;
Double_t betaL_20m,alphaL_20m,n0_20m,n1_20m,n2_20m;
Double_t Lcon_20m,sigmaE_20m,sigmaT_20m,corrE_20m,corrT_20m;
Double_t sigmaE_c_20m,xavg_20m,yavg_20m;
Double_t Dx_20m,Dy_20m,Dr_20m,Dr2_20m;
Double_t regn_30m,Z_30m,Bz_30m;
Double_t eperp_30m,elong_30m,e6D_30m;
Double_t Ldim_30m,Pzavg_30m,beta_30m,alpha_30m;
Double_t betaL_30m,alphaL_30m,n0_30m,n1_30m,n2_30m;
Double_t Lcon_30m,sigmaE_30m,sigmaT_30m,corrE_30m,corrT_30m;
Double_t sigmaE_c_30m,xavg_30m,yavg_30m;
Double_t Dx_30m,Dy_30m,Dr_30m,Dr2_30m;
Double_t num_m_10m[268];
Double_t num_p_10m[268];
Double_t num_k_10m[268];
Double_t num_m_20m[268];
Double_t num_p_20m[268];
Double_t num_k_20m[268];
Double_t num_m_30m[268];
Double_t num_p_30m[268];
Double_t num_k_30m[268];
Int_t index0_10m;
Int_t index1_10m;
Int_t index2_10m;
Int_t index0_20m;
Int_t index1_20m;
Int_t index2_20m;
Int_t index0_30m;
Int_t index1_30m;
Int_t index2_30m;
Int_t region_10m;
Int_t region_20m;
Int_t region_30m;
Double_t num_n0_10m[37];
Double_t num_n1_10m[37];
Double_t num_n2_10m[37];
Double_t num_n0_20m[38];
Double_t num_n1_20m[38];
Double_t num_n2_20m[38];
```

```
Double_t num_n0_30m[39];
Double_t num_n1_30m[39];
Double_t num_n2_30m[39];

//Initialization of Variables
oldreg_10m = 1;
oldreg_20m = 1;
oldreg_30m = 1;
line1_10m = 0;
line1_20m = 0;
line1_30m = 0;
line3_10m = 0;
line3_20m = 0;
line3_30m = 0;
icount_10m = 0;
icount_20m = 0;
icount_30m = 0;
index0_10m = 0;
index1_10m = 0;
index2_10m = 0;
index0_20m = 0;
index1_20m = 0;
index2_20m = 0;
index0_30m = 0;
index1_30m = 0;
index2_30m = 0;
region_10m = 0;
region_20m = 0;
region_30m = 0;
for(Int_t i = 0; i < 268; i++){
  num_10m[i] = 0;
 }
 for(Int_t i = 0; i < 268; i++){
  num_20m[i] = 0;
 }
 for(Int_t i = 0; i < 268; i++){
  num_30m[i] = 0;
 }

 // Open ecalc9f_10m.dat file
 ifstream fin2(filein2.c_str());
 if(!fin2){
  cout << "Cannot open file" << filein2 << endl;
  return0;
 }
 else cout << "File " << filein2 << " opened " << endl;

 //Skip header lines
 for(Int_t i = 0; i < 13; i++){
  fin2.getline(header3_10m,1000);
  cout << "header3_10m 1 " << header3_10m << endl;
  line3_10m++;
  cout << "line 3 " << line3_10m << endl;  }
 while(fin2 && line3_10m < maxline3_10m){
  fin2 >> regn_10m >> Z_10m >> Bz_10m >> eperp_10m >> elong_10m >> e6D_10m;
  fin2 >> Ldim_10m >> Pzavg_10m >> beta_10m >> alpha_10m >> betaL_10m;
  fin2 >> alphaL_10m >> n0_10m >> n1_10m >> n2_10m >> Lcon_10m;
```

```
  fin2 >> sigmaE_10m >> sigmaT_10m >> corrE_10m >> corrT_10m;
  fin2 >> sigmaE_c_10m >> xavg_10m >> yavg_10m >> Dx_10m >> Dy_10m >> Dr_10m >> Dr2_10m;

  num_n0_10m[index0_10m] = n0_10m;
  num_n1_10m[index1_10m] = n1_10m;
  num_n2_10m[index2_10m] = n2_10m;
  index0_10m++;
  index1_10m++;
  index2_10m++;
  line3_10m++;
}

fin2.close();
fin2.clear();

// Open ecalc9f_20m.dat file
ifstream fin4(filein4.c_str());
if(!fin4){
  cout << "Cannot open file" << filein4 << endl;
  return0;
}
else cout << "File " << filein4 << " opened " << endl;

//Skip header lines
for(Int_t i = 0; i < 13; i++){
  fin4.getline(header3_20m,1000);
  cout << "header3_20m 1 " << header3_20m << endl;
  line3_20m++;
  cout << "line 3 " << line3_20m << endl;
}

while(fin4 && line3_20m < maxline3_20m){
  fin4 >> regn_20m >> Z_20m >> Bz_20m >> eperp_20m >> elong_20m >> e6D_20m;
  fin4 >> Ldim_20m >> Pzavg_20m >> beta_20m >> alpha_20m >> betaL_20m;
  fin4 >> alphaL_20m >> n0_20m >> n1_20m >> n2_20m >> Lcon_20m;
  fin4 >> sigmaE_20m >> sigmaT_20m >> corrE_20m >> corrT_20m;
  fin4 >> sigmaE_c_20m >> xavg_20m >> yavg_20m >> Dx_20m >> Dy_20m >> Dr_20m >> Dr2_20m;

  num_n0_20m[index0_20m] = n0_20m;
  num_n1_20m[index1_20m] = n1_20m;
  num_n2_20m[index2_20m] = n2_20m;

  index0_20m++;
  index1_20m++;
  index2_20m++;

  line3_20m++;
}

fin4.close();
fin4.clear();


// Open ecalc9f_30m.dat file
ifstream fin6(filein6.c_str());
if(!fin6){
  cout << "Cannot open file" << filein6 << endl;
```

```
 return0;
}
else cout << "File " << filein6 << " opened " << endl;

//Skip header lines
for(Int_t i = 0; i < 13; i++){
 fin6.getline(header3_30m,1000);
 cout << "header3_30m 1 " << header3_30m << endl;
 line3_30m++;
 cout << "line 3 " << line3_30m << endl;
}

while(fin6 && line3_30m < maxline3_30m){
 fin6 >> regn_30m >> Z_30m >> Bz_30m >> eperp_30m >> elong_30m >> e6D_30m;
 fin6 >> Ldim_30m >> Pzavg_30m >> beta_30m >> alpha_30m >> betaL_30m;
 fin6 >> alphaL_30m >> n0_30m >> n1_30m >> n2_30m >> Lcon_30m;
 fin6 >> sigmaE_30m >> sigmaT_30m >> corrE_30m >> corrT_30m;
 fin6 >> sigmaE_c_30m >> xavg_30m >> yavg_30m >> Dx_30m >> Dy_30m >> Dr_30m >> Dr2_30m;

 num_n0_30m[index0_30m] = n0_30m;
 num_n1_30m[index1_30m] = n1_30m;
 num_n2_30m[index2_30m] = n2_30m;

 index0_30m++;
 index1_30m++;
 index2_30m++;

 line3_30m++;
}

fin6.close();
fin6.clear();

// Open for009_10m.dat file
ifstream fin1(filein1.c_str());
if(!fin1){
 cout << "Cannot open file" << filein1 << endl;
 return 0;
}
else cout << "File" << filein1 << "opened" << endl;

//Skip header lines
fin1.getline(header_10m,1000);
cout << "header 1 " << header_10m << endl;
line1_10m++;
cout << "line 1 " << line1_10m << endl;

fin1.getline(header_10m,1000);
cout << "header 2 " << header_10m << endl;
line1_10m++;
cout << "line 2 " << line1_10m << endl;

fin1.getline(header_10m,1000);
cout << "header 3 " << header_10m << endl;
line1_10m++;
cout << "line 3 " << line1_10m << endl;
```

```
 while( fin1 && line1_10m < maxline1_10m ){

  fin1 >> evt_10m >> par_10m >> typ_10m >> flg_10m >> reg_10m;
  fin1 >> time_10m >> x_10m >> y_10m >> z_10m >> px_10m >> py_10m >> pz_10m
      >> bx_10m >> by_10m >> bz_10m >> wt_10m >> ex_10m >> ey_10m >> ez_10m;
  fin1 >> arclength_10m >> polx_10m >> poly_10m >> polz_10m;

  if(reg_10m == oldreg_10m && flg_10m == 0){
   zpos_10m[icount_10m] = z_10m;
   num_10m[icount_10m] = num_10m[icount_10m] + wt_10m;

   if(typ_10m == -2 || typ_10m == 2){
    num_m_10m[icount_10m] = num_m_10m[icount_10m] + wt_10m;
   }

   if(typ_10m == -3 || typ_10m == 3){
    num_p_10m[icount_10m] = num_p_10m[icount_10m] + wt_10m;
   }
  }

  if(reg_10m != oldreg_10m){
   cout << "z_10m " << z_10m << " oldreg_10m " << oldreg_10m << " reg_10m " << reg_10m << " icount_10m "
<< icount_10m << endl;
    icount_10m++;
    oldreg_10m = reg_10m;
  }

  momentum_10m = TMath::Sqrt(pow(px_10m,2)+pow(py_10m,2)+pow(pz_10m,2));

  if(typ_10m == -2 || typ_10m == 2){
   HMm_10m[icount_10m]->Fill(time_10m*pow(10,9),momentum_10m,wt_10m);
   HMY_10m[icount_10m]->Fill(momentum_10m,wt_10m);
  }

  if(typ_10m == -3 || typ_10m == 3){
   HPm_10m[icount_10m]->Fill(time_10m*pow(10,9),momentum_10m,wt_10m);
   HPY_10m[icount_10m]->Fill(momentum_10m,wt_10m);
  }
  line1_10m++;
 }

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

 // Open for009.dat_20m file
 ifstream fin3(filein3.c_str());
 if(!fin3){
  cout << "Cannot open file" << filein3 << endl;
  return 0;
 }
 else cout << "File" << filein3 << "opened" << endl;

 //Skip header lines
 fin3.getline(header_20m,1000);
 cout << "header 1 " << header_20m << endl;
 line1_20m++;
 cout << "line 1 " << line1_20m << endl;
```

```
fin3.getline(header_20m,1000);
cout << "header 2 " << header_20m << endl;
line1_20m++;
cout << "line 2 " << line1_20m << endl;

fin3.getline(header_20m,1000);
cout << "header 3 " << header_20m << endl;
line1_20m++;
cout << "line 3 " << line1_20m << endl;

while( fin3 && line1_20m < maxline1_20m ){

 fin3 >> evt_20m >> par_20m >> typ_20m >> flg_20m >> reg_20m;
 fin3 >> time_20m >> x_20m >> y_20m >> z_20m >> px_20m >> py_20m >> pz_20m
    >> bx_20m >> by_20m >> bz_20m >> wt_20m >> ex_20m >> ey_20m >> ez_20m;
 fin3 >> arclength_20m >> polx_20m >> poly_20m >> polz_20m;

 if(reg_20m == oldreg_20m && flg_20m == 0){
  zpos_20m[icount_20m] = z_20m;

  num_20m[icount_20m] = num_20m[icount_20m] + wt_20m;

  if(typ_20m == -2 || typ_20m == 2){
   num_m_20m[icount_20m] = num_m_20m[icount_20m] + wt_20m;
  }

  if(typ_20m == -3 || typ_20m == 3){
   num_p_20m[icount_20m] = num_p_20m[icount_20m] + wt_20m;
  }
 }

 if(reg_20m != oldreg_20m){
  cout << "z_20m " << z_20m << " oldreg_20m " << oldreg_20m << " reg_20m " << reg_20m << " icount_20m "
<< icount_20m << endl;
  icount_20m++;
  oldreg_20m = reg_20m;
 }

 momentum_20m = TMath::Sqrt(pow(px_20m,2)+pow(py_20m,2)+pow(pz_20m,2));

 if(typ_20m == -2 || typ_20m == 2){
  HMm_20m[icount_20m]->Fill(time_20m*pow(10,9),momentum_20m,wt_20m);
  HMY_20m[icount_20m]->Fill(momentum_20m,wt_20m);
 }

 if(typ_20m == -3 || typ_20m == 3){
  HPm_20m[icount_20m]->Fill(time_20m*pow(10,9),momentum_20m,wt_20m);
  HPY_20m[icount_20m]->Fill(momentum_20m,wt_20m);
 }
   line1_20m++;
}

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

// Open for009.dat_30m file
```

```
ifstream fin5(filein5.c_str());
if(!fin5){
 cout << "Cannot open file" << filein5 << endl;
 return 0;
}
else cout << "File" << filein5 << "opened" << endl;

//Skip header lines
fin5.getline(header_30m,1000);
cout << "header 1 " << header_30m << endl;
line1_30m++;
cout << "line 1 " << line1_30m << endl;

fin5.getline(header_30m,1000);
cout << "header 2 " << header_30m << endl;
line1_30m++;
cout << "line 2 " << line1_30m << endl;

fin5.getline(header_30m,1000);
cout << "header 3 " << header_30m << endl;
line1_30m++;
cout << "line 3 " << line1_30m << endl;

while( fin5 && line1_30m < maxline1_30m ){

 fin5 >> evt_30m >> par_30m >> typ_30m >> flg_30m >> reg_30m;
 fin5 >> time_30m >> x_30m >> y_30m >> z_30m >> px_30m >> py_30m >> pz_30m
    >> bx_30m >> by_30m >> bz_30m >> wt_30m >> ex_30m >> ey_30m >> ez_30m;
 fin5 >> arclength_30m >> polx_30m >> poly_30m >> polz_30m;

 if(reg_30m == oldreg_30m && flg_30m == 0){
  zpos_30m[icount_30m] = z_30m;
  num_30m[icount_30m] = num_30m[icount_30m] + wt_30m;

  if(typ_30m == -2 || typ_30m == 2){
   num_m_30m[icount_30m] = num_m_30m[icount_30m] + wt_30m;
  }

  if(typ_30m == -3 || typ_30m == 3){
   num_p_30m[icount_30m] = num_p_30m[icount_30m] + wt_30m;
  }
 }

 if(reg_30m != oldreg_30m){
  cout << "z_30m " << z_30m << " oldreg_30m " << oldreg_30m << " reg_30m " << reg_30m << " icount_30m "
<< icount_30m << endl;
  icount_30m++;
  oldreg_30m = reg_30m;
 }

 momentum_30m = TMath::Sqrt(pow(px_30m,2)+pow(py_30m,2)+pow(pz_30m,2));

 if(typ_30m == -2 || typ_30m == 2){
  HMm_30m[icount_30m]->Fill(time_30m*pow(10,9),momentum_30m,wt_30m);
  HMY_30m[icount_30m]->Fill(momentum_30m,wt_30m);
 }
```

```
 if(typ_30m == -3 || typ_30m == 3){
   HPm_30m[icount_30m]->Fill(time_30m*pow(10,9),momentum_30m,wt_30m);
   HPY_30m[icount_30m]->Fill(momentum_30m,wt_30m);
 }

 line1_30m++;
}

fin5.close();
fin5.clear();

//TGraf and TMultigraph of muons as a function of their z position in every region
//10m drift
Double_t xpm_10m[37];
Double_t ypm_10m[37];
Double_t EXm_10m[37];
Double_t EYm_10m[37];
TGraphErrors *muons_10m;

for(Int_t i = 0; i < 37; i++){
 xpm_10m[i] = zpos_10m[i];
 ypm_10m[i] = num_m_10m[i];
 EYm_10m[i] = 0.;
 EXm_10m[i] = 0.;
 }

muons_10m = new TGraphErrors(37,xpm_10m,ypm_10m,EXm_10m,EYm_10m);

TMultiGraph *Graph1_10m;
Graph1_10m = new TMultiGraph("Graph1_10m","Total number of muons, 10m drift ");
c0.Clear();
muons_10m->SetMarkerColor(1);
muons_10m->SetLineColor(1);
Graph1_10m->Add(muons_10m,"lp");
Graph1_10m->Draw("AC*");
Graph1_10m->GetXaxis()->SetTitle("z [m]");
Graph1_10m->GetYaxis()->SetTitleOffset(1.4);
Graph1_10m->GetYaxis()->SetTitleSize(0.03);
Graph1_10m->GetYaxis()->SetLabelSize(0.03);
Graph1_10m->GetYaxis()->SetTitle("muons");
c0.Print(outplots);

//20m drift
Double_t xpm_20m[38];
Double_t ypm_20m[38];
Double_t EXm_20m[38];
Double_t EYm_20m[38];
TGraphErrors *muons_20m;

for(Int_t i = 0; i < 38; i++){
 xpm_20m[i] = zpos_20m[i];
 ypm_20m[i] = num_m_20m[i];
 EYm_20m[i] = 0.;
 EXm_20m[i] = 0.;
 // cout << " xpm_20m " << xpm_20m[i] << " ypm_20m " << ypm_20m[i] << " i " << i << endl;
}
```

```
muons_20m = new TGraphErrors(38,xpm_20m,ypm_20m,EXm_20m,EYm_20m);

TMultiGraph *Graph1_20m;
Graph1_20m = new TMultiGraph("Graph1_20m","Total number of muons, 20m drift ");
c0.Clear();
muons_20m->SetMarkerColor(1);
muons_20m->SetLineColor(1);
Graph1_20m->Add(muons_20m,"lp");
Graph1_20m->Draw("AC*");
Graph1_20m->GetXaxis()->SetTitle("z [m]");
Graph1_20m->GetYaxis()->SetTitleOffset(1.4);
Graph1_20m->GetYaxis()->SetTitleSize(0.03);
Graph1_20m->GetYaxis()->SetLabelSize(0.03);
Graph1_20m->GetYaxis()->SetTitle("muons");
c0.Print(outplots);

//30m drift
Double_t xpm_30m[39];
Double_t ypm_30m[39];
Double_t EXm_30m[39];
Double_t EYm_30m[39];
TGraphErrors *muons_30m;

for(Int_t i = 0; i < 39; i++){
  xpm_30m[i] = zpos_30m[i];
  ypm_30m[i] = num_m_30m[i];
  EYm_30m[i] = 0.;
  EXm_30m[i] = 0.;
  // cout << " xpm_30m " << xpm_30m[i] << " ypm_30m " << ypm_30m[i] << " i " << i << endl;
}

muons_30m = new TGraphErrors(39,xpm_30m,ypm_30m,EXm_30m,EYm_30m);

TMultiGraph *Graph1_30m;
Graph1_30m = new TMultiGraph("Graph1_30m","Total number of muons, 30m drift ");
c0.Clear();
muons_30m->SetMarkerColor(1);
muons_30m->SetLineColor(1);
Graph1_30m->Add(muons_30m,"lp");
Graph1_30m->Draw("AC*");
Graph1_30m->GetXaxis()->SetTitle("z [m]");
Graph1_30m->GetYaxis()->SetTitleOffset(1.4);
Graph1_30m->GetYaxis()->SetTitleSize(0.03);
Graph1_30m->GetYaxis()->SetLabelSize(0.03);
Graph1_30m->GetYaxis()->SetTitle("muons");
c0.Print(outplots);

//TGraf of pions as a function of their z position in every region
//10m drift
Double_t xpp_10m[37];
Double_t ypp_10m[37];
Double_t EXp_10m[37];
Double_t EYp_10m[37];
TGraphErrors *pions_10m;

for(Int_t i = 0; i < 37; i++){
  xpp_10m[i] = zpos_10m[i];
```

```
 ypp_10m[i] = num_p_10m[i];
 EYp_10m[i] = 0.;
 EXp_10m[i] = 0.;
}

pions_10m = new TGraphErrors(37,xpp_10m,ypp_10m,EXp_10m,EYp_10m);

TMultiGraph *Graph2_10m;
Graph2_10m = new TMultiGraph("Graph2_10m"," Number of pions, 10m drift ");

c0.Clear();
pions_10m->SetMarkerColor(1);
pions_10m->SetLineColor(1);
Graph2_10m->Add(pions_10m,"lp");
Graph2_10m->Draw("AC*");
Graph2_10m->GetXaxis()->SetTitle("z [m]");
Graph2_10m->GetYaxis()->SetTitleOffset(1.4);
Graph2_10m->GetYaxis()->SetTitleSize(0.03);
Graph2_10m->GetYaxis()->SetLabelSize(0.03);
Graph2_10m->GetYaxis()->SetTitle("Pions");
c0.Print(outplots);

//20m drift
Double_t xpp_20m[38];
Double_t ypp_20m[38];
Double_t EXp_20m[38];
Double_t EYp_20m[38];
TGraphErrors *pions_20m;

for(Int_t i = 0; i < 38; i++){
 xpp_20m[i] = zpos_20m[i];
 ypp_20m[i] = num_p_20m[i];
 EYp_20m[i] = 0.;
 EXp_20m[i] = 0.;
}

pions_20m = new TGraphErrors(38,xpp_20m,ypp_20m,EXp_20m,EYp_20m);

TMultiGraph *Graph2_20m;
Graph2_20m = new TMultiGraph("Graph2_20m"," Number of pions, 20m drift ");

c0.Clear();
pions_20m->SetMarkerColor(1);
pions_20m->SetLineColor(1);
Graph2_20m->Add(pions_20m,"lp");
Graph2_20m->Draw("AC*");
Graph2_20m->GetXaxis()->SetTitle("z [m]");
Graph2_20m->GetYaxis()->SetTitleOffset(1.4);
Graph2_20m->GetYaxis()->SetTitleSize(0.03);
Graph2_20m->GetYaxis()->SetLabelSize(0.03);
Graph2_20m->GetYaxis()->SetTitle("Pions");
c0.Print(outplots);

//30m drift
Double_t xpp_30m[39];
Double_t ypp_30m[39];
Double_t EXp_30m[39];
```

```
Double_t EYp_30m[39];
TGraphErrors *pions_30m;

for(Int_t i = 0; i < 39; i++){
  xpp_30m[i] = zpos_30m[i];
  ypp_30m[i] = num_p_30m[i];
  EYp_30m[i] = 0.;
  EXp_30m[i] = 0.;
}

pions_30m = new TGraphErrors(39,xpp_30m,ypp_30m,EXp_30m,EYp_30m);

TMultiGraph *Graph2_30m;
Graph2_30m = new TMultiGraph("Graph2_30m"," Number of pions, 30m drift ");

c0.Clear();
pions_30m->SetMarkerColor(1);
pions_30m->SetLineColor(1);
Graph2_30m->Add(pions_30m,"lp");
Graph2_30m->Draw("AC*");
Graph2_30m->GetXaxis()->SetTitle("z [m]");
Graph2_30m->GetYaxis()->SetTitleOffset(1.4);
Graph2_30m->GetYaxis()->SetTitleSize(0.03);
Graph2_30m->GetYaxis()->SetLabelSize(0.03);
Graph2_30m->GetYaxis()->SetTitle("Pions");
c0.Print(outplots);

//TGraph and TMultigraph n0
//10m drift
Double_t xn0_10m[37];
Double_t yn0_10m[37];
Double_t Exn0_10m[37];
Double_t Eyn0_10m[37];
TGraphErrors *Graph_n0_10m;

for(Int_t i = 0; i < 37; i++){
  xn0_10m[i] = zpos_10m[i];
  yn0_10m[i] = num_n0_10m[i];
  Exn0_10m[i] = 0;
  Eyn0_10m[i] = 0;
  // cout << " xn0_10m = zpos_10m " << xn0_10m[i] << " yn0_10m = n0_10m " << yn0_10m[i] << endl;
}

Graph_n0_10m = new TGraphErrors(37,xn0_10m,yn0_10m,Exn0_10m,Eyn0_10m);

TMultiGraph *TGraph_n0_10m;
TGraph_n0_10m = new TMultiGraph("TGraph_n0_10m"," n0, 10m drift ");

c0.Clear();
TGraph_n0_10m->Add(Graph_n0_10m,"lp");
TGraph_n0_10m->Draw("AC*");
TGraph_n0_10m->GetXaxis()->SetTitle("z [m]");
TGraph_n0_10m->GetYaxis()->SetTitle("n0");
c0.Print(outplots);

//20m drift
Double_t xn0_20m[38];
```

```
Double_t yn0_20m[38];
Double_t Exn0_20m[38];
Double_t Eyn0_20m[38];
TGraphErrors *Graph_n0_20m;

for(Int_t i = 0; i < 38; i++){
  xn0_20m[i] = zpos_20m[i];
  yn0_20m[i] = num_n0_20m[i];
  Exn0_20m[i] = 0;
  Eyn0_20m[i] = 0;
  // cout << " xn0_20m = zpos_20m " << xn0_20m[i] << " yn0_20m = n0_20m " << yn0_20m[i] << endl;
}

Graph_n0_20m = new TGraphErrors(38,xn0_20m,yn0_20m,Exn0_20m,Eyn0_20m);

TMultiGraph *TGraph_n0_20m;
TGraph_n0_20m = new TMultiGraph("TGraph_n0_20m"," n0, 20m drift ");

c0.Clear();
TGraph_n0_20m->Add(Graph_n0_20m,"lp");
TGraph_n0_20m->Draw("AC*");
TGraph_n0_20m->GetXaxis()->SetTitle("z [m]");
TGraph_n0_20m->GetYaxis()->SetTitle("n0");
c0.Print(outplots);

//30m drift
Double_t xn0_30m[39];
Double_t yn0_30m[39];
Double_t Exn0_30m[39];
Double_t Eyn0_30m[39];
TGraphErrors *Graph_n0_30m;

for(Int_t i = 0; i < 39; i++){
  xn0_30m[i] = zpos_30m[i];
  yn0_30m[i] = num_n0_30m[i];
  Exn0_30m[i] = 0;
  Eyn0_30m[i] = 0;
  }

Graph_n0_30m = new TGraphErrors(39,xn0_30m,yn0_30m,Exn0_30m,Eyn0_30m);

TMultiGraph *TGraph_n0_30m;
TGraph_n0_30m = new TMultiGraph("TGraph_n0_30m"," n0, 30m drift ");

c0.Clear();
TGraph_n0_30m->Add(Graph_n0_30m,"lp");
TGraph_n0_30m->Draw("AC*");
TGraph_n0_30m->GetXaxis()->SetTitle("z [m]");
TGraph_n0_30m->GetYaxis()->SetTitle("n0");
c0.Print(outplots);

//TGraph and Tmultigraph n1
//10m drift
Double_t xn1_10m[37];
Double_t yn1_10m[37];
Double_t Exn1_10m[37];
Double_t Eyn1_10m[37];
```

```
TGraphErrors *Graph_n1_10m;

for(Int_t i = 0; i < 37; i++){
  xn1_10m[i] = zpos_10m[i];
  yn1_10m[i] = num_n1_10m[i];
  Exn1_10m[i] = 0;
  Eyn1_10m[i] = 0;
}

Graph_n1_10m = new TGraphErrors(37,xn1_10m,yn1_10m,Exn1_10m,Eyn1_10m);

TMultiGraph *TGraph_n1_10m;
TGraph_n1_10m = new TMultiGraph("TGraph_n1_10m"," n1, 10m drift ");

c0.Clear();
TGraph_n1_10m->Add(Graph_n1_10m,"lp");
TGraph_n1_10m->Draw("AC*");
TGraph_n1_10m->GetXaxis()->SetTitle("z [m]");
TGraph_n1_10m->GetYaxis()->SetTitle("n1");
c0.Print(outplots);

//20m drift
Double_t xn1_20m[38];
Double_t yn1_20m[38];
Double_t Exn1_20m[38];
Double_t Eyn1_20m[38];
TGraphErrors *Graph_n1_20m;

for(Int_t i = 0; i < 38; i++){
  xn1_20m[i] = zpos_20m[i];
  yn1_20m[i] = num_n1_20m[i];
  Exn1_20m[i] = 0;
  Eyn1_20m[i] = 0;
}

Graph_n1_20m = new TGraphErrors(38,xn1_20m,yn1_20m,Exn1_20m,Eyn1_20m);

TMultiGraph *TGraph_n1_20m;
TGraph_n1_20m = new TMultiGraph("TGraph_n1_20m"," n1, 20m drift ");

c0.Clear();
TGraph_n1_20m->Add(Graph_n1_20m,"lp");
TGraph_n1_20m->Draw("AC*");
TGraph_n1_20m->GetXaxis()->SetTitle("z [m]");
TGraph_n1_20m->GetYaxis()->SetTitle("n1");
c0.Print(outplots);

//30m drift
Double_t xn1_30m[39];
Double_t yn1_30m[39];
Double_t Exn1_30m[39];
Double_t Eyn1_30m[39];
TGraphErrors *Graph_n1_30m;

for(Int_t i = 0; i < 39; i++){
  xn1_30m[i] = zpos_30m[i];
  yn1_30m[i] = num_n1_30m[i];
```

```
 Exn1_30m[i] = 0;
 Eyn1_30m[i] = 0;
  }

Graph_n1_30m = new TGraphErrors(39,xn1_30m,yn1_30m,Exn1_30m,Eyn1_30m);

TMultiGraph *TGraph_n1_30m;
TGraph_n1_30m = new TMultiGraph("TGraph_n1_30m"," n1, 30m drift ");

c0.Clear();
TGraph_n1_30m->Add(Graph_n1_30m,"lp");
TGraph_n1_30m->Draw("AC*");
TGraph_n1_30m->GetXaxis()->SetTitle("z [m]");
TGraph_n1_30m->GetYaxis()->SetTitle("n1");
c0.Print(outplots);

//TGraph and TMultigraph n2
//10m drift
Double_t xn2_10m[37];
Double_t yn2_10m[37];
Double_t Exn2_10m[37];
Double_t Eyn2_10m[37];
TGraphErrors *Graph_n2_10m;

for(Int_t i = 0; i < 37; i++){
 xn2_10m[i] = zpos_10m[i];
 yn2_10m[i] = num_n2_10m[i];
 Exn2_10m[i] = 0;
 Eyn2_10m[i] = 0;
  }

Graph_n2_10m = new TGraphErrors(37,xn2_10m,yn2_10m,Exn2_10m,Eyn2_10m);

TMultiGraph *TGraph_n2_10m;
TGraph_n2_10m = new TMultiGraph("TGraph_n2_10m"," n2, 10m drift ");

c0.Clear();
TGraph_n2_10m->Add(Graph_n2_10m,"lp");
TGraph_n2_10m->Draw("AC*");
TGraph_n2_10m->GetXaxis()->SetTitle("z [m]");
TGraph_n2_10m->GetYaxis()->SetTitle("n2");
c0.Print(outplots);

//20m drift
Double_t xn2_20m[38];
Double_t yn2_20m[38];
Double_t Exn2_20m[38];
Double_t Eyn2_20m[38];
TGraphErrors *Graph_n2_20m;

for(Int_t i = 0; i < 38; i++){
 xn2_20m[i] = zpos_20m[i];
 yn2_20m[i] = num_n2_20m[i];
 Exn2_20m[i] = 0;
 Eyn2_20m[i] = 0;
}
```

```
Graph_n2_20m = new TGraphErrors(38,xn2_20m,yn2_20m,Exn2_20m,Eyn2_20m);

TMultiGraph *TGraph_n2_20m;
TGraph_n2_20m = new TMultiGraph("TGraph_n2_20m"," n2, 20m drift ");

c0.Clear();
TGraph_n2_20m->Add(Graph_n2_20m,"lp");
TGraph_n2_20m->Draw("AC*");
TGraph_n2_20m->GetXaxis()->SetTitle("z [m]");
TGraph_n2_20m->GetYaxis()->SetTitle("n2");
c0.Print(outplots);

//30m drift
Double_t xn2_30m[39];
Double_t yn2_30m[39];
Double_t Exn2_30m[39];
Double_t Eyn2_30m[39];
TGraphErrors *Graph_n2_30m;

for(Int_t i = 0; i < 39; i++){
  xn2_30m[i] = zpos_30m[i];
  yn2_30m[i] = num_n2_30m[i];
  Exn2_30m[i] = 0;
  Eyn2_30m[i] = 0;
  }

Graph_n2_30m = new TGraphErrors(39,xn2_30m,yn2_30m,Exn2_30m,Eyn2_30m);

TMultiGraph *TGraph_n2_30m;
TGraph_n2_30m = new TMultiGraph("TGraph_n2_30m"," n2, 30m drift ");

c0.Clear();
TGraph_n2_30m->Add(Graph_n2_30m,"lp");
TGraph_n2_30m->Draw("AC*");
TGraph_n2_30m->GetXaxis()->SetTitle("z [m]");
TGraph_n2_30m->GetYaxis()->SetTitle("n2");
c0.Print(outplots);

//TMultigraphs n1 and n2
//10m drift
TMultiGraph *Graph1_10m;
Graph1_10m = new TMultiGraph("Graph_1_10m "," n2 and n1, 10m drift ");

c0.Clear();
Graph_n1_10m->SetMarkerStyle(27);
Graph_n1_10m->SetMarkerColor(3);
Graph_n1_10m->SetLineColor(3);
Graph1_10m->Add(Graph_n1_10m,"lp");
Graph_n2_10m->SetMarkerColor(2);
Graph_n2_10m->SetLineColor(2);
Graph1_10m->Add(Graph_n2_10m,"lp");
Graph1_10m->Draw("a");
Graph1_10m->GetXaxis()->SetTitle("z [m]");
Graph1_10m->GetYaxis()->SetTitle("N");

leg1 = new TLegend(0.8,0.8,0.9,0.9);
leg1->AddEntry(Graph_n1_10m,"n1","lp");
```

```
leg1->AddEntry(Graph_n2_10m,"n2","lp");
leg1->Draw();
c0.Print(outplots);

//20m drift
TMultiGraph *Graph1_20m;
Graph1_20m = new TMultiGraph("Graph_1_20m "," n2 and n1, 20m drift ");

c0.Clear();
Graph_n1_20m->SetMarkerStyle(27);
Graph_n1_20m->SetMarkerColor(3);
Graph_n1_20m->SetLineColor(3);
Graph1_20m->Add(Graph_n1_20m,"lp");
Graph_n2_20m->SetMarkerColor(2);
Graph_n2_20m->SetLineColor(2);
Graph1_20m->Add(Graph_n2_20m,"lp");
Graph1_20m->Draw("a");
Graph1_20m->GetXaxis()->SetTitle("z [m]");
Graph1_20m->GetYaxis()->SetTitle("N");
Graph1_20m->GetYaxis()->SetRangeUser(0.,30.);

leg1 = new TLegend(0.8,0.8,0.9,0.9);
leg1->AddEntry(Graph_n1_20m,"n1","lp");
leg1->AddEntry(Graph_n2_20m,"n2","lp");
leg1->Draw();
c0.Print(outplots);

//30m drift
TMultiGraph *Graph1_30m;
Graph1_30m = new TMultiGraph("Graph_1_30m "," n2 and n1, 30m drift ");

c0.Clear();
Graph_n1_30m->SetMarkerStyle(27);
Graph_n1_30m->SetMarkerColor(3);
Graph_n1_30m->SetLineColor(3);
Graph1_30m->Add(Graph_n1_30m,"lp");
Graph_n2_30m->SetMarkerColor(2);
Graph_n2_30m->SetLineColor(2);
Graph1_30m->Add(Graph_n2_30m,"lp");
Graph1_30m->Draw("a");
Graph1_30m->GetXaxis()->SetTitle("z [m]");
Graph1_30m->GetYaxis()->SetTitle("N");
leg1 = new TLegend(0.8,0.8,0.9,0.9);
leg1->AddEntry(Graph_n1_30m,"n1","lp");
leg1->AddEntry(Graph_n2_30m,"n2","lp");
leg1->Draw();
c0.Print(outplots);

//TMultigraph n0 and muons
//10m drift
TMultiGraph *Graph2_10m;
Graph2_10m = new TMultiGraph("Graph_2_10m ","Total number of muons and n0, 10m drift  ");

c0.Clear();
Graph_n0_10m->SetMarkerColor(4);
Graph_n0_10m->SetLineColor(4);
Graph2_10m->Add(Graph_n0_10m,"lp");
```

```
Graph2_10m->Add(muons_10m,"lp");
Graph2_10m->Draw("a");
Graph2_10m->GetXaxis()->SetTitle("z [m]");
Graph2_10m->GetYaxis()->SetTitleOffset(1.4);
Graph2_10m->GetYaxis()->SetTitleSize(0.03);
Graph2_10m->GetYaxis()->SetLabelSize(0.03);
Graph2_10m->GetYaxis()->SetTitle("N");
Graph2_10m->GetYaxis()->SetRangeUser(0.,4000.);
leg1 = new TLegend(0.8,0.8,0.9,0.9);
leg1->AddEntry(Graph_n0_10m,"n0","l");
leg1->AddEntry(muons_10m,"muons","l");
leg1->Draw();
c0.Print(outplots);

//20m drift
TMultiGraph *Graph2_20m;
Graph2_20m = new TMultiGraph("Graph_2_20m ","Total number of muons and n0, 20m drift ");

c0.Clear();
Graph_n0_20m->SetMarkerColor(4);
Graph_n0_20m->SetLineColor(4);
Graph2_20m->Add(Graph_n0_20m,"lp");
Graph2_20m->Add(muons_20m,"lp");
Graph2_20m->Draw("a");
Graph2_20m->GetXaxis()->SetTitle("z [m]");
Graph2_20m->GetYaxis()->SetTitleOffset(1.4);
Graph2_20m->GetYaxis()->SetTitleSize(0.03);
Graph2_20m->GetYaxis()->SetLabelSize(0.03);
Graph2_20m->GetYaxis()->SetTitle("N");
Graph2_20m->GetYaxis()->SetRangeUser(0.,4000.);
leg1 = new TLegend(0.8,0.8,0.9,0.9);
leg1->AddEntry(Graph_n0_20m,"n0","l");
leg1->AddEntry(muons_20m,"muons","l");
leg1->Draw();
c0.Print(outplots);

//30m drift
TMultiGraph *Graph2_30m;
Graph2_30m = new TMultiGraph("Graph_2_30m ","Total number of muons and n0, 30m drift ");

c0.Clear();
Graph_n0_30m->SetMarkerColor(4);
Graph_n0_30m->SetLineColor(4);
Graph2_30m->Add(Graph_n0_30m,"lp");
Graph2_30m->Add(muons_30m,"lp");
Graph2_30m->Draw("a");
Graph2_30m->GetXaxis()->SetTitle("z [m]");
Graph2_30m->GetYaxis()->SetTitleOffset(1.4);
Graph2_30m->GetYaxis()->SetTitleSize(0.03);
Graph2_30m->GetYaxis()->SetLabelSize(0.03);
Graph2_30m->GetYaxis()->SetTitle("N");
Graph2_30m->GetYaxis()->SetRangeUser(0.,4000.);
leg1 = new TLegend(0.8,0.8,0.9,0.9);
leg1->AddEntry(Graph_n0_30m,"n0","l");
leg1->AddEntry(muons_30m,"muons","l");
leg1->Draw();
c0.Print(outplots);
```

```
// Combine graphs of Total muons and n0 for all three drifts
TMultiGraph *CGraph1;
CGraph1 = new TMultiGraph("CGraph1","Number of muons n and n0 for 10m, 20m and 30m drifts");

c0.Clear();
muons_10m->SetMarkerColor(1);
muons_10m->SetLineColor(1);
CGraph1->Add(muons_10m,"lp");
muons_20m->SetMarkerColor(2);
muons_20m->SetLineColor(2);
CGraph1->Add(muons_20m,"lp");
muons_30m->SetMarkerColor(7);
muons_30m->SetLineColor(7);
CGraph1->Add(muons_30m,"lp");
Graph_n0_10m->SetMarkerColor(4);
Graph_n0_10m->SetLineColor(4);
CGraph1->Add(Graph_n0_10m,"lp");
Graph_n0_20m->SetMarkerColor(3);
Graph_n0_20m->SetLineColor(3);
CGraph1->Add(Graph_n0_20m,"lp");
Graph_n0_30m->SetMarkerColor(6);
Graph_n0_30m->SetLineColor(6);
CGraph1->Add(Graph_n0_30m,"lp");
CGraph1->Draw("a");
CGraph1->GetXaxis()->SetTitle("z [m]");
CGraph1->GetYaxis()->SetTitleOffset(1.4);
CGraph1->GetYaxis()->SetTitleSize(0.03);
CGraph1->GetYaxis()->SetLabelSize(0.03);
CGraph1->GetYaxis()->SetTitle("N");
CGraph1->GetYaxis()->SetRangeUser(0.,5000.);
leg1 = new TLegend(0.8,0.7,0.90,0.9);
leg1->AddEntry(muons_10m,"n-10m","l");
leg1->AddEntry(muons_20m,"n-20m","l");
leg1->AddEntry(muons_30m,"n-30m","l");
leg1->AddEntry(Graph_n0_10m,"n0-10m","l");
leg1->AddEntry(Graph_n0_20m,"n0-20m","l");
leg1->AddEntry(Graph_n0_30m,"n0-30m","l");
leg1->Draw();
c0.Print(outplots);

// Combine graphs of n1 and n2 for all three drifts
TMultiGraph *CGraph2;
CGraph2 = new TMultiGraph("CGraph2 ","n1 and n2 for 10m, 20m and 30m drifts");

c0.Clear();
Graph_n1_10m->SetMarkerStyle(1);
Graph_n1_10m->SetMarkerColor(1);
Graph_n1_10m->SetLineColor(1);
CGraph2->Add(Graph_n1_10m,"lp");
Graph_n1_20m->SetMarkerColor(2);
Graph_n1_20m->SetLineColor(2);
CGraph2->Add(Graph_n1_20m,"lp");
Graph_n1_30m->SetMarkerColor(28);
Graph_n1_30m->SetLineColor(28);
CGraph2->Add(Graph_n1_30m,"lp");
Graph_n2_10m->SetMarkerColor(4);
```

```
Graph_n2_10m->SetLineColor(4);
CGraph2->Add(Graph_n2_10m,"lp");
Graph_n2_20m->SetMarkerColor(8);
Graph_n2_20m->SetLineColor(8);
CGraph2->Add(Graph_n2_20m,"lp");
Graph_n2_30m->SetMarkerColor(6);
Graph_n2_30m->SetLineColor(6);
CGraph2->Add(Graph_n2_30m,"lp");
CGraph2->Draw("a");
CGraph2->GetXaxis()->SetTitle("z [m]");
CGraph2->GetYaxis()->SetRangeUser(0.,50.);
CGraph2->GetYaxis()->SetTitle("N");
leg1 = new TLegend(0.8,0.7,0.90,0.9);
leg1->AddEntry(Graph_n1_10m,"n1-10m","lp");
leg1->AddEntry(Graph_n1_20m,"n1-20m","lp");
leg1->AddEntry(Graph_n1_30m,"n1-30m","lp");
leg1->AddEntry(Graph_n2_10m,"n2-10m","lp");
leg1->AddEntry(Graph_n2_20m,"n2-20m","lp");
leg1->AddEntry(Graph_n2_30m,"n2-30m","lp");
leg1->Draw();

c0.Print(outplots);

//Ploting the histograms
//Muons' momentum as a function of time
//10m drift
for(Int_t i = 0; i < maxplane_10m; i++){
  c0.Clear();
  HMm_10m[i]->SetMarkerStyle(20);
  HMm_10m[i]->SetMarkerSize(0.5);
  HMm_10m[i]->SetYTitle("Momentum [GeV]");
  HMm_10m[i]->SetXTitle("Time [ns]");
  HMm_10m[i]->SetMarkerSize(0.5);
  HMm_10m[i]->SetLineColor(2);
  HMm_10m[i]->Draw();
  c0.Print(outplots);
}

//20m drift
for(Int_t i = 0; i < maxplane_20m; i++){
  c0.Clear();
  HMm_20m[i]->SetMarkerStyle(20);
  HMm_20m[i]->SetMarkerSize(0.5);
  HMm_20m[i]->SetYTitle("Momentum [GeV]");
  HMm_20m[i]->SetXTitle("Time [ns]");
  HMm_20m[i]->SetMarkerSize(0.5);
  HMm_20m[i]->SetLineColor(2);
  HMm_20m[i]->Draw();
  c0.Print(outplots);
}

//30m drift
for(Int_t i = 0; i < maxplane_30m; i++){
  c0.Clear();
  HMm_30m[i]->SetMarkerStyle(20);
  HMm_30m[i]->SetMarkerSize(0.5);
  HMm_30m[i]->SetYTitle("Momentum [GeV]");
```

```
 HMm_30m[i]->SetXTitle("Time [ns]");
 HMm_30m[i]->SetMarkerSize(0.5);
 HMm_30m[i]->SetLineColor(2);
 HMm_30m[i]->Draw();
 c0.Print(outplots);
}

//Pions' momentum as a function of time
//10m drift
for(Int_t i = 0; i < maxplane_10m; i++){
 c0.Clear();
 HPm_10m[i]->SetMarkerStyle(20);
 HPm_10m[i]->SetMarkerSize(0.5);
 HPm_10m[i]->SetYTitle("Momentum [GeV]");
 HPm_10m[i]->SetXTitle("Time [ns]");
 HPm_10m[i]->SetMarkerStyle(20);
 HPm_10m[i]->SetMarkerSize(0.5);
 HPm_10m[i]->SetLineColor(2);
 HPm_10m[i]->Draw();
 c0.Print(outplots);
}

//20m drift
for(Int_t i = 0; i < maxplane_20m; i++){
 c0.Clear();
 HPm_20m[i]->SetMarkerStyle(20);
 HPm_20m[i]->SetMarkerSize(0.5);
 HPm_20m[i]->SetYTitle("Momentum [GeV]");
 HPm_20m[i]->SetXTitle("Time [ns]");
 HPm_20m[i]->SetMarkerStyle(20);
 HPm_20m[i]->SetMarkerSize(0.5);
 HPm_20m[i]->SetLineColor(2);
 HPm_20m[i]->Draw();
 c0.Print(outplots);
}

//30m drift
for(Int_t i = 0; i < maxplane_30m; i++){
 c0.Clear();
 HPm_30m[i]->SetMarkerStyle(20);
 HPm_30m[i]->SetMarkerSize(0.5);
 HPm_30m[i]->SetYTitle("Momentum [GeV]");
 HPm_30m[i]->SetXTitle("Time [ns]");
 HPm_30m[i]->SetMarkerStyle(20);
 HPm_30m[i]->SetMarkerSize(0.5);
 HPm_30m[i]->SetLineColor(2);
 HPm_30m[i]->Draw();
 c0.Print(outplots);
}

//Muon Yield
//10m drift
for(Int_t i = 0; i < maxplane_10m; i++){
 c0.Clear();
 HMY_10m[i]->SetMarkerStyle(20);
 HMY_10m[i]->SetMarkerSize(0.5);
 HMY_10m[i]->SetYTitle("Yield");
```

```
  HMY_10m[i]->SetXTitle("Momentum [GeV]");
  HMY_10m[i]->SetMarkerStyle(20);
  HMY_10m[i]->SetMarkerSize(0.5);
  HMY_10m[i]->SetLineColor(2);
  HMY_10m[i]->Draw();
  c0.Print(outplots);
 }

 //20m drift
 for(Int_t i = 0; i < maxplane_20m; i++){
  c0.Clear();
  HMY_20m[i]->SetMarkerStyle(20);
  HMY_20m[i]->SetMarkerSize(0.5);
  HMY_20m[i]->SetYTitle("Yield");
  HMY_20m[i]->SetXTitle("Momentum [GeV]");
  HMY_20m[i]->SetMarkerStyle(20);
  HMY_20m[i]->SetMarkerSize(0.5);
  HMY_20m[i]->SetLineColor(2);
  HMY_20m[i]->Draw();
  c0.Print(outplots);
 }

 //30m drift
 for(Int_t i = 0; i < maxplane_30m; i++){
  c0.Clear();
  HMY_30m[i]->SetMarkerStyle(20);
  HMY_30m[i]->SetMarkerSize(0.5);
  HMY_30m[i]->SetYTitle("Yield");
  HMY_30m[i]->SetXTitle("Momentum [GeV]");
  HMY_30m[i]->SetMarkerStyle(20);
  HMY_30m[i]->SetMarkerSize(0.5);
  HMY_30m[i]->SetLineColor(2);
  HMY_30m[i]->Draw();
  c0.Print(outplots);
 }

 //Pion Yield
 //10m drift
 for(Int_t i = 0; i < maxplane_10m; i++){
  c0.Clear();
  HPY_10m[i]->SetMarkerStyle(20);
  HPY_10m[i]->SetMarkerSize(0.5);
  HPY_10m[i]->SetYTitle("Yield");
  HPY_10m[i]->SetXTitle("Momentum [GeV]");
  HPY_10m[i]->SetMarkerStyle(20);
  HPY_10m[i]->SetMarkerSize(0.5);
  HPY_10m[i]->SetLineColor(2);
  HPY_10m[i]->Draw();
  c0.Print(outplots);
 }

 //20m drift
 for(Int_t i = 0; i < maxplane_20m; i++){
  c0.Clear();
  HPY_20m[i]->SetMarkerStyle(20);
  HPY_20m[i]->SetMarkerSize(0.5);
  HPY_20m[i]->SetYTitle("Yield");
```

```
  HPY_20m[i]->SetXTitle("Momentum [GeV]");
  HPY_20m[i]->SetMarkerStyle(20);
  HPY_20m[i]->SetMarkerSize(0.5);
  HPY_20m[i]->SetLineColor(2);
  HPY_20m[i]->Draw();
  c0.Print(outplots);
 }

 //30m drift
 for(Int_t i = 0; i < maxplane_30m; i++){
  c0.Clear();
  HPY_30m[i]->SetMarkerStyle(20);
  HPY_30m[i]->SetMarkerSize(0.5);
  HPY_30m[i]->SetYTitle("Yield");
  HPY_30m[i]->SetXTitle("Momentum [GeV]");
  HPY_30m[i]->SetMarkerStyle(20);
  HPY_30m[i]->SetMarkerSize(0.5);
  HPY_30m[i]->SetLineColor(2);
  HPY_30m[i]->Draw();
  c0.Print(outplots);
 }
 c0.Print(Form("%s]",outplots));
}
```