# Acoustic Detection of Ultra High Energy Neutrinos

Simon Bevan

Supervised by Dr. David Waters

March 26, 2004

**Abstract**

This report investigates the possibility of using acoustic techniques to search for ultra-high-energy neutrinos. Using Geant4, hadronic showers were simulated up to energies of 100TeV and the energy structure of the shower analysed. The dE/dr and dE/dz components of the shower were parameterised. Using this parameterised form for an hadronic shower, 50000 events were generated over the energy range $10^{17}$eV - $10^{21}$eV, the energy range over which the the theoretical GZK flux is expected to be detected. The efficiencies of different array designs were investigated by reconstructing an acoustic pulse at each hydrophone due to each hadronic shower. For each array a predicted rate is given. Reconstruction techniques for the original position of the shower are also discussed.

## Acknowledgements

I'd like to thank Dr. David Waters for all his time and patience during the last six months.

# Contents

# List of Figures

# List of Tables

# Executive Summary

This paper will be split into three sections, with each section representing different aspects of the research that was undertaken into simulating an acoustic UHE neutrino detector. Section 1 details the reasons and motivation into developing UHE neutrino detectors.

The second section details the research into the structure of UHE hadronic showers using Geant4, with the aim of generating UHE hadronic showers without using Geant4. Using Geant4, showers were simulated over the energy range 1TeV - 100TeV (the maximum energy possible using Geant4) and the de/dz and dE/dr components analysed. From these studies, equations representing dE/dz and dE/dr were parameterised and UHE hadronic showers reconstructed.

Acoustic pulses were then generated for both the Geant4 and modelled showers. The modelled hadronic showers proved successful, and a simulation was made to reconstruct the acoustic pulses generated by these showers.

The third section details the work that was carried out using the above-described model to investigate arrays of detectors. Research was carried out into finding the most efficient array shape, and reconstructing the position and direction of the original neutrino, with the aim of using this information to point at the original source of the neutrino, a neutrino telescope.

For each array a predicted rate is calculated. The maximum time resolution and real detectors are also discussed.

# 1 UHE neutrino telescopes

## 1.1 Theory

Most modern techniques rely on the use of the electromagnetic spectrum (radio, infrared, visible, ultraviolet, x-ray, and most recently gamma rays) to study the universe, but new techniques are being developed to study the universe via an alternative particle, the neutrino. This method is still in its infancy, but if the new techniques prove to be successful many of the unanswered questions in astrophysics today could be answered.

## 1.2 History of the neutrino

The idea of the neutrino first arose in 1930 when there appeared to be a problem in the study of nuclear beta decay (a nucleus A is transformed into a lighter nucleus B with the emission of an electron. It is characteristic of two-body decays that the outgoing energies are kinetically determined in the centre of mass frame. If A is at rest, so that B and e come out back to with equal and opposite momenta, then the conservation of energy dictates that the electron energy is

$$E = \left( \frac{\left(ma^2 - mb^2 + me^2\right)}{2ma} \right) c^2 \tag{1}$$

Hence, the electron energy is fixed if the masses of A and B are known, but this was not the case. When experiments were performed it was found that the energy of the electron varied considerably, with equation 1 only giving the maximum energy. This proved a problem, as seemingly the conservation laws were broken. However, Pauli suggested that along with B and e there was another, electrically neutral, particle emitted. For the electron to achieve the energies that were found in the study beta decay, this particle had to be extremely light or massless. It was Fermi that named this particle the neutrino..

Until the mid-fifties there was compelling theoretical evidence for the neutrino, but no direct experimental evidence, it left no tracks, it didn't decay, it just proved a handy book-keeping tool for the conservation laws. Cowen and Reines finally detected this elusive particle at the Savannah River nuclear reactor in South Carolina using inverse beta decay. These results proved unambiguous conformation of the neutrino's existence [1].

Now that the existence of the neutrino had been discovered, the next obvious question was is the neutrino its own anti-neutrino? The photon is its own antiparticle, as is the neutral pion. This was answered in the late fifties by Davis and Harmer [1].

$$\nu + n \rightarrow p^+ + e^- \tag{2a}$$
$$\bar{\nu} + n \rightarrow p^+ + e^- \tag{2b}$$

2a was observed and 2b was never observed. This established that the neutrino and anti-neutrino were two unique particles.

The next chapter in the story of the neutrino is the story of the neutrino flavours. Experimentally, the decay $\mu^- \rightarrow e^- + \gamma$ is never observed, but it is consistent with the conservation of charge and the conservation of lepton number. In particle physics, what is not forbidden is mandatory. This therefore suggests a conservation of mu-ness, but then how do you explain reactions like

$$\mu \rightarrow e + \nu + \bar{\nu} \tag{3}$$

The answer was to have a neutrino associated with the muon, and a neutrino associated with the electron. This again proved amazingly successful in theory, and Lederman, Schwartz, and Steinberg successfully distinguished both neutrinos in 1962. The tau neutrino was eventually added to the list in 2000.

The final and unfinished chapter in the history of the neutrino is neutrino oscillation and mass.

## 1.3 The neutrino

As discussed above the neutrino is electrically weak and has a very small mass. The cross section of the neutrino is also extremely small, and hence interacts with matter very rarely. For a neutrino at 1MeV the mean free path is about $10^6$km.

The neutrino is second most abundant particle in the universe, with a number density of $3.10^6/\mathrm{m}^3$ (number one, the photon has a number density of $1.10^7/\mathrm{m}^3$, and the proton only $0.5/\mathrm{m}^3$). Although being so abundant, the neutrino is a very difficult particle to work with and the potential of neutrino telescopes are only now being investigated.

Whereas nuclear energy (MeV) neutrino astronomy has been established, eg SuperK, the low cross section and poor angular resolution of MeV neutrinos has always been a problem in advancing the techniques to much beyond the Sun and nearby supernovae. A proposed solution to this problem is to detect ultra-high energy (UHE) neutrinos (Energy $> 10.^{10}$ GeV).

Using UHE neutrinos has the advantages that, i) the increase of the neutrino cross-section, and angular resolution with energy, and ii) the opportunity to use large natural target material (water, salt, ice). Detection at energies of this magnitude is above the limit of conventional optical neutrino detectors. In order to detect these high energies new detectors are being designed.

There are many methods of detecting UHE neutrinos that are being investigated. This report will focus solely on the detection of UHE neutrinos by the detection of the acoustic pulse generated when the neutrino interacts with water - acoustic detection.

## 1.4 Other Detection Methods

There are also other types of neutrino detectors, namely optical Cerenkov, optical shower detectors, and radio Cerenkov. Radio Cerenkov is hoping for the same energies as acoustic detection, but the optical Cerenkov is at lower energies. Shower detectors hope to go to as high energies as acoustic but the charged particles suffer greatly from attenuation. Fig 1 shows the energy range for these other types of detectors.

## 1.5 What are the aims of neutrino astronomy?

High-energy astronomy has been much advanced with invention of the gamma ray telescope, but the range of modern high-energy gamma ray telescopes is limited. This is because of attenuation of gamma rays from distant sources. At a few hundred TeV, gamma rays do not survive the journey from the centre of our galaxy. Therefore the view into the early universe is hidden from photon astronomy due to the photon-matter coupling barrier. With neutrino astronomy we could look to cosmological interesting distances, as neutrinos suffer no such attenuation.

With UHE neutrino astronomy we could also look into the densest places of the universe hidden to current techniques. UHE neutrino techniques could help prove/dismiss many of astrophysics current problems. For example we would be able to see the to-date unseen details of the ultimate energy release during the collapse of matter, origin of cosmic rays, and could even give crucial insights into the nature of dark matter. Observations of UHE neutrinos from cosmologically interesting distances may also give scientists crucial information about heavy element formation, and help explain neutrino mass (the standard model predicts they should be mass less).

Another important advantage of using neutrinos, is the ability to distinguish between electromagnetic and hadronic processes. Accelerated electrons and protons both result in the production of high energy rays, but only hadronic processes result in the production of neutrinos.

## 1.6 Potential sources UHE neutrinos

The origin and mechanism for accelerating UHE cosmic rays remains one of the major unknown phenomena in astrophysics. Isolated sources of UHE neutrinos are believed to be produced by protons accelerated in highly energetic environments colliding with the ambient gas and radiation surrounding the accelerating environments through proton-proton or proton-photon interactions via pion production and decay. Such accelerators might be an active galactic nuclei, AGN, powered by a super massive black hole, gamma ray bursts, or in shock fronts, Fermi acceleration [2], of supernovae remnants, micro-quasars, and magnetars.

## 1.7 Predictions of expected flux

Predictions have been made for the expected flux of UHE neutrinos based on the standard model of particle physics (although neutrino mass is beyond the standard model!); namely Waxman-Bachall limit and the GZK cut off.

**Waxman-Bachall limit**[3] is $2.10^{-8}$ / $E^2(GeVcm^2secsr)^{-1}$ - Derived from photo-meson interactions. The value quoted here is the upper limit, and is based on assumptions described in [3].

Mannheim et al corrected Waxmann Bachall limit [4] and extended the analysis to include cases where the emitted protons suffer adiabatic losses on the large scale, and magnetic fields and absorption processes within the sources. This results in more general bounds allowing much higher extra galactic neutrino fluxes.

**GZK cut-off** [5] is the limit of maximum energy of protons of cosmological origin because of the finite ( 50Mpc) inelastic collision length of such particles in the cosmic background radiation. The highest energy cosmic rays are energetic enough to have photo-production reactions with the cosmic background radiation. The secondary mesons produced in the photo-production decay into gamma rays and neutrinos. Using a full model of neutrino production due to proton interactions on CMB photons over the full lifetime of the universe, calculations of the flux of GZK neutrinos can be made, fig 1.

$$p + \gamma_{cmb} \rightarrow \pi^+ + n \tag{4a}$$

$$n \rightarrow p + e^- + \bar{\nu}_e \tag{4b}$$

$$\pi^+ \rightarrow \mu + \nu_{mu} \tag{4c}$$

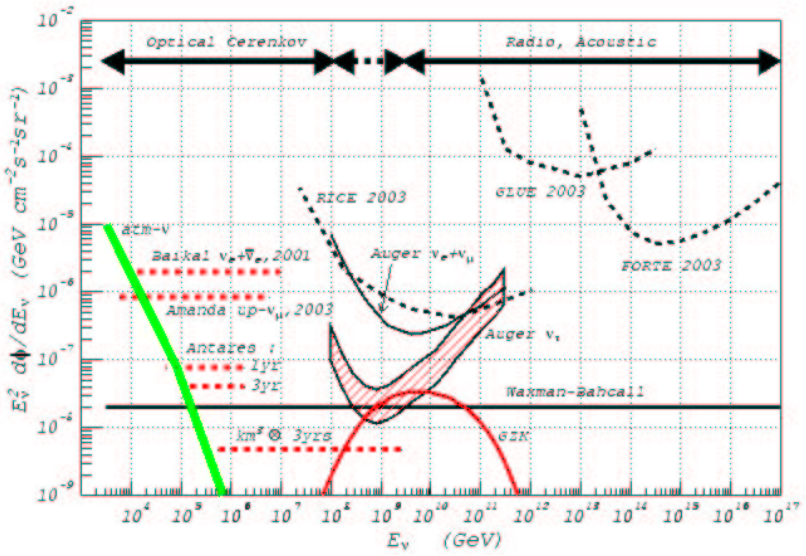



Figure 1: Showing the GZK, Waxman-Bachall, and the atmospheric neutrino flux. The fig also shows the upper limits of other neutrino detectors and the dates they are expected to come on-line.

$$\mu \rightarrow e^- + \nu_{mu} + \nu_e \tag{4d}$$

GZK neutrinos are of significant interest for the early studies of UHE neutrinos, as the measurement of the detected flux can be directly compared with theory. For energies above the GZK cut-off point, comparing the measured flux with theory becomes more difficult, due to lack of theory. Although fig 1 shows an upper limit for the Waxman-Bahcall flux extending to $1.10^{17}$GeV, this is only generated on general arguments about Fermi acceleration in astrophysical objects. This also must have a cut-off at some point, possibly around $10^{21}$ or $10^{22}$ eV, but the actual value is unknown. Other possible models for the neutrino flux at energies above the GZK limit are models in which neutrinos are generated by the decay of extremely massive particles (M $>$ 1.0E14 GeV). These massive particles are thought to be remnants from the very early universe, and have never been detected. Hence any flux calculated from there decay is highly theoretical and based on no actual measurements.

Models based on the decay of massive particles are called "top-down" models. "Bottom-up" models try to explain the extremely high-energy cosmic ray data through more conventional acceleration mechanisms in astrophysical objects.

GZK neutrinos also act as good particles to use in neutrino telescopes. This is because the neutrinos travel in a straight line from their point of origin, and the protons from which they were created are of sufficiently high energy as to also travel in a straight line from their point of origin. For these reasons this investigation is going to focus on the detecting GZK neutrinos, $10^{17}$eV $<$ E $<$ $10^{21}$eV.

Observations of fluxes above these limits would indicate either independent sources or physics beyond the standard model.

## 1.8 Dark Matter

The fundamental problem with all accepted models in astrophysics is the predictions of dark matter and dark energy, so I have included a brief section on how neutrino astronomy can help resolve this debate. These neutrinos are at TeV energies, and are too low for detection with acoustic techniques. Other neutrino detectors detecting at lower energies, fig 1, will be able

to detect them. From studying the dynamics of galaxies and clusters it has been found that there is a missing mass problem. As this 'missing' mass cannot be detected with any modern telescopes, it has been named dark matter. There are various candidates for dark matter, but GUTs suggest that weakly interacting massive particles (WIMPS) should dominate the dark matter. Neutrinos are also a possible candidate for dark matter, but the known upper limit on the neutrino mass, $\Sigma m_\nu < 0.7$eV, is too small to be a valid candidate with current theories. Neutrinos are also relativistic at decoupling, which means that they can't explain the small-scale structure, (which grow to become galaxies). Neutrino astronomy would be able to detect the flux of neutrinos expected from the decay or annihilations of the WIMPS, allowing determination of their mass and density.

Recent results from WMAP [6] suggest that the universe consists of 73% dark energy, 23% dark matter, and 4% baryonic matter. This means that theoretically dark matter and energy exist, but has never been directly detected - remind you of a period in the neutrinos history!

## 1.9 Detecting UHE neutrinos

At UHE neutrino energies the cross section of the neutrinos is sufficiently high for the Earth to become opaque to the neutrinos. The neutrinos interact with the water, and via deep inelastic scattering a hadronic shower is produced.

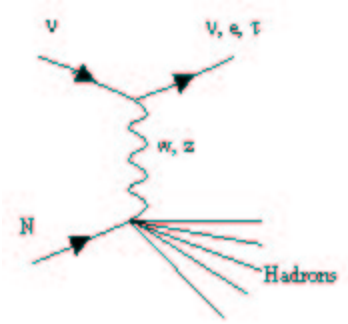


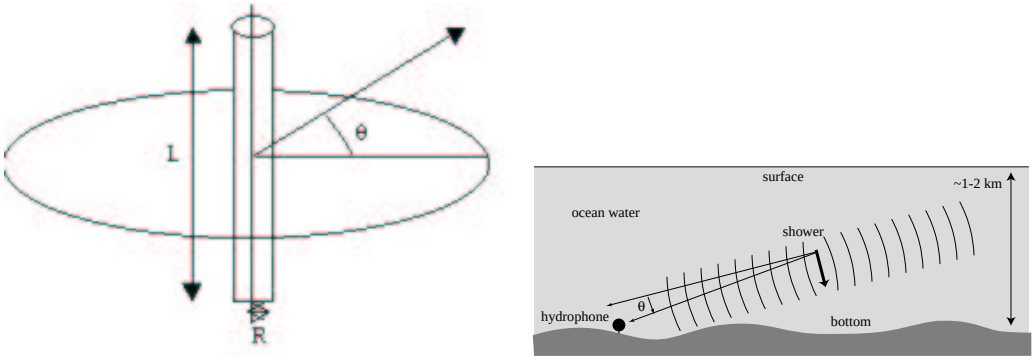

Figure 2: Deep inelastic scattering

Hadronic and electromagnetic cascades, produced by the interactions of UHE neutrinos, can deposit sufficient thermal energy, via ionisation losses, in the surrounding medium fig 3. The heat dissipates only slowly, so the bulk effect is an expansion of the region where the energy was deposited. This in turn produces a bi-polar acoustic pulse with leading compression. The speed of sound in water is much less than the speed of the shower propagation, and hence the energy deposition along the length of the shower can be considered instantaneous, and hence the radiation is coherent and adds. This produces a pancake shape acoustic pulse, fig 3.

## 1.10 Arrays of Detectors

The height of the pressure pulse falls rapidly out of the plane perpendicular to the shower axis, and most of the acoustic energy from a shower is concentrated in a small angular range ($\theta < |6|^\circ$). Hence a single detector will detect only a small angular range. To over come this problem an array of detectors could be used.

At large distances from the shower axis, and in the plane of propagation, the acoustic emission is coherent, giving rise to large amplitude pressure pulses. An important advantage of acoustic detectors in water is that the attenuation length is of the order of kilometres, (10km for components of 10 kHz to about 1km for components of 30 kHz). Therefore the main constraint of the array design is the angular resolution.




(a) The pancake shaped acoustic pulse

(b) A schematic of the acoustic pulse propagation through the water

Figure 3: acoustic arrays

Having an array over large volumes of water will improve the counting rate potential by orders of magnitudes compared to that of a single hydrophone. Using an array also has the advantage that the data can be extrapolated to reconstruct the cascade position and the neutrino direction, and can hence be used as a pointer to look for the origin of the UHE neutrinos. This is a particular import advantage of UHE neutrino telescopes over shower detectors, as directional information is lost for charged particles below $10^{10}$ GeV due to deflection in cosmic magnetic fields.

## 2 Simulating acoustic pulses

As described above, there are three possible large natural resources that could be used for acoustic detection of UHE neutrinos - water, ice, and salt. This paper concentrates on the detection in water. For the purpose of this investigation, all that is required is the pressure pulse at a given point, the hydrophone (the chosen detector to detect the acoustic pulse) due to the energy deposition of the neutrino.

The pressure measured at hydrophone location r from the neutrino cascade is given by: -

$$p\left(\vec{r},t\right)=\int_{v}\rho_{E}\left(\vec{r'}\right)G\left(\left|\vec{r}-\vec{r'}\right|,t\right)d^{3}\vec{r'} \tag{5}$$

$$G\left(r,t\right)=-\frac{\beta}{4\pi C_{p}}\frac{\left(t-\frac{r}{c}\right)}{r\sqrt{2\pi}\tau^{3}}exp\left(-\frac{\left(t-\frac{r}{c}\right)^{2}}{\left(2\tau^{2}\right)}\right) \tag{6}$$

$$\tau=\sqrt{\frac{r}{\left(\omega_{0}c\right)}} \tag{7}$$

$\rho_E$ = thermal energy density; $\beta$ = coefficient of thermal expansivity / $(0.000201344°C^{-1})$; Cp = specific heat capacity / $(3980.43)JK^{-1}kg^{-1}$; c = speed of sound / $(1505.19\ ms^{-1})$; $\omega_0$ = attenuation frequency - $(2.5.10^{10})$; t = time / s; r = radius / m; where the numbers represent values predicted for the Rona array.

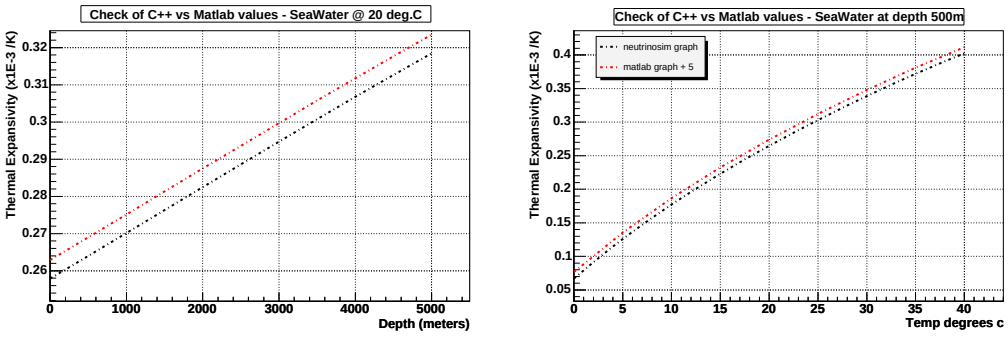
### 2.1 Simulating water properties, $\beta$, Cp, and c

**Code - appendix B.1**

Equation 6 has a dependence on $\beta$, Cp, and c, which are in turn dependent on temperature, depth and the salinity of the water. Equation 6 also shows the dependence of the amplitude of the acoustic pulse on $\beta$ and inversely on Cp. The temperature, depth, and salinity are user definable. The temperature depends on where in world the hydrophones are going to be placed, and hence a measured temperature can be used. This simulation uses 13°C, the measured temperature for the water surrounding the Rona hydrophones. The salinity varies from 0 psu (practical salinity units) for fresh water, to 35 psu for salt water. Although the salinity is seasonally dependent, it only changes by 2 or 3 psu, and has an average of 35 psu. This paper investigates the pressure pulses in salt water using 35psu, and in fresh water using 0 psu.

All acoustic pulses were generated using a depth of 300m. 300m has been chosen as the depth as it is the depth of the Rona hydrophones [1]. This adds a slight inconsistency to the simulation. This inconsistency occurs as the volume of water that the neutrinos are being simulated has a radius of 10km, and hence neutrinos are being generated above the surface of the water. The programme could very easily be adapted to account for this, having the depth be a function of the hydrophone location, but one of the main aims of the simulation is to investigate the conditions around the Rona hydrophones.

The programme that was created takes the temperature, depth, and salinity as an input parameter and calculates $\beta$, Cp, and c. To calculate $\beta$, Cp, and c a MatLab programm was adapted.




(a) thermal expansivity as a function of depth

(b) thermal expansivity as a function of temp

Figure 4: Thermal expansivity

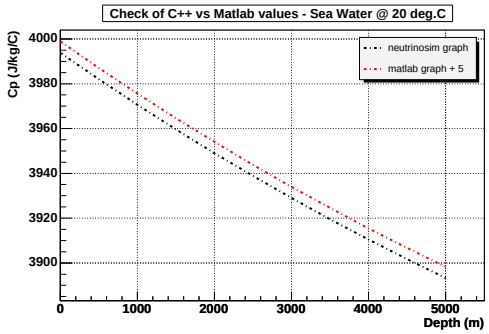
## 2.2 Simulating random, energy, position, direction

**Code - randomPosition.hh - appendix B.2**
**Code - randomEnergy.hh - appendix B.3**

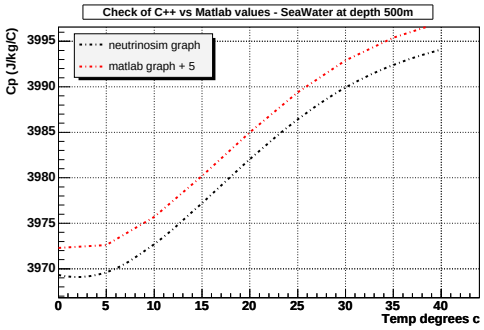
For the studies into UHE neutrinos, the energy region of interest is the GZK region, $10^{17}$eV - $10^{21}eV$, as discussed in §1.7

To make sure of an even distribution over this region, a log scale was used. If a linear scale was used, powers of $10^{21}$ would dominate, and when plotted on a log scale the energy will not be evenly distributed To ensure of an even distribution in the log scale, the energy is randomly

[1] The Rona hydrophones are a MOD owned array of hydrophones off the island of Rona. They are to be used to study the effect of noise, pulse recognition techniques, data acquisition, and any flux predictions. If the studies on the Rona arrays prove successful, designing a full size UHE neutrino detector can be considered.

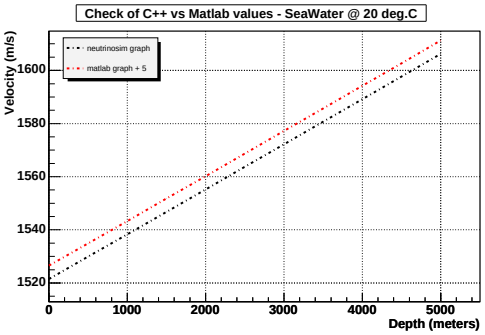



(a) specific heat capacity as a function of depth

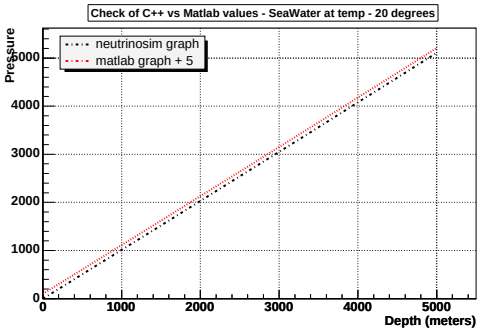



(b) specific heat capacity as a function of temp

Figure 5: Specific heat capacity




(a) velocity




(b) pressure

Figure 6: velocity and pressure as functions of depth

generated between the log of the maximum and minimum, and transformed back into the eV scale.

The final quantity that is required to calculate the pressure pulse is the distance of the shower from the hydrophone. To find this, the hydrophone position, §3.4 and shower position are required. The shower position and direction, as with energy, were randomly generated using a random number generator.

Spherical polar co-ordinates best describe the position in a sphere. To generate random positions in a sphere, r, $\theta$, and $\phi$ were generated randomly.

$$dV = r^2 \sin(\theta) dr d\theta d\phi \tag{8}$$

A maximum radius was defined, 10,000m, and a random r generated between the centre of the sphere, 0m, and the maximum radius. To get an even distribution of r in a sphere, a random $r^3$ must be defined, and r being the cube root of this number. If r was generated on a linear scale, when considering a sphere, there is a larger volume at greater r, and hence the density of particles per unit volume would less at larger r, and not an even distribution. Generating in r3 ensures that the particle density is even throughout the volume of water, as would be the case in a real system.

For $\theta$, equation 8 must be considered. It shows that $\theta$ cannot be generated on a linear scale either. $\theta$ must be generated on a scale of dsin$\theta$ or /cos$\theta$ between -1 and 1.

$\phi$ is simpler, this can simply be generated between 0 and $2\pi$.

r, $\theta$, and $\phi$ were then transformed into Cartesian co-ordinates. To change into Cartesian co-ordinates: -

$$x = r \sin(\theta) \cos(\phi) \tag{9}$$
$$y = r \sin(\theta) \sin(\phi) \tag{10}$$
$$z = r \cos(\theta) \tag{11}$$

**Turning into shower concentric co-ordinates** The co-ordinate system described so far assumes that a hydrophone is at the centre of the sphere, and the shower is generated in a volume around this hydrophone. This system is OK if one hydrophone is to be studied, but for arrays of hydrophones this is not a practical way of describing the system. For many hydrophones, the most practical system is to describe the hydrophones relative to the shower. To transform a hydrophone centred co-ordinate system into a shower centred co-ordinate system, the following equations must be used.

$$sensor_x = r \cos(\theta) \tag{12}$$
$$sensor_y = 0.0 \tag{13}$$
$$sensor_z = r \sin(\theta) \tag{14}$$
$$radius = [(shower\_position_x - hy\_location_x)^2 + \tag{15}$$
$$(shower\_position_y - hy\_location_y)^2 + \tag{16}$$
$$(shower\_position_z - hy\_location_z)^2]^{0.5} \tag{17}$$

$\theta$ must also be changed.

$$L = H - S \tag{18}$$

$$\theta = acos\left(\frac{d \cdot L}{|d|\,|L|}\right) \tag{19}$$

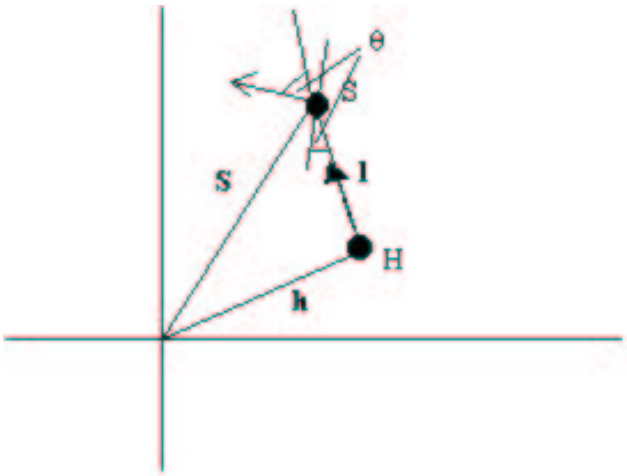



Figure 7: Calculateing theta

## 2.3 Simulating hadronic showers

Equation 6 generates an acoustic pulse due to a point source, but an hadronic shower is not a point source. For a non-point source, equation 5 must be considered. This reconstructs an acoustic pulse due to a region of energy, or many point sources by integrating over the entire energy range. The region of energy in this case being the shower. It is therefore important that the shower represents the true energy deposition. To accurately study particle showers a programme called Geant4 [7] was used.

### 2.3.1 Geant4

Geant4 is programme which simulates particle showers. The programme allows the user to define the initial particle, the energy of the initial particle, and the material that the shower is to develop in. Geant4 then details the position, energy, and decay mode of every step in the shower. The output of Geant4 is also very user definable. For the studies in this project, the position and energy of each step were written to a Root ntuple for further analysis.

Geant4 has a very basic interface that needed to be adapted before any research could be performed. Firstly Geant4 had to specifically be told what physics to use. This was achieved by adding the appropriate headers and the corresponding code in the main body of the programme. Of particular importance to this experiment was the addition of hadronic physics.

The next step was to define the material that the shower was to develop in. For this project two materials were defined. They were water and salt-water. Geant4 has a list of all elements but no definition of any compounds. To define a material the density and mass had to be given.

For water, a density of 1.000 g/cm$^3$ was used with a ratio of one oxygen to two hydrogen. Where oxygen has an atomic weight of 16.00 and hydrogen and atomic weight of 1.01. For sea-water a salinity of 35 psu was used. Salinity is the total amount of salts dissolved in water; grams of salts per kilo of water (g/kg) or parts per thousand. This implies for a salinity of 35 psu, 35g of salts to 1000g of water. Which is 3.38% salts to 96.62% $H_20$. The main constituent of the salts in seawater is NaCl, about 90% [8], where the atomic weight of sodium is 22.990 g/mole, and of chlorine is 35.453 g/mole (all atomic masses are quoted from [9]). To calculate the density of NaCl the following method was used.

$$density = \frac{mass}{volume} = \frac{mass\_per\_unit\_cell}{volume\_per\_unit\_cell} \tag{20}$$

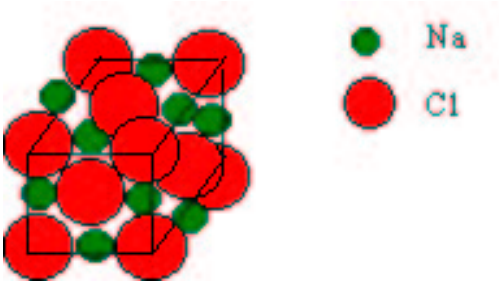



Figure 8: a salt lattice

Fig 8 shows per unit cell there are 14 Cl atoms and 13 Na atoms. This gives a mass of (14Cl +13Na) ⇒

$$mass = (14Cl + 13Na)\,(g/mol)\,(atoms/unit\_cell)\,\left(1mol/6.022.10^{23}atoms\right) \tag{21}$$

$$\textbf{mass} = \mathbf{1.321.10^{-21}\ g/unit\ cell}$$

The volume,

$$V = (2rCl + 2rNa)^3 \tag{22}$$

Where rCl = 1.81Åand rNa = 1.02Åare the radii of the corresponding atoms.

$$\mathbf{V = 1.823E\text{-}22\ cm3/unit\ cell}$$

$$Density = (1.321E-21)(g/unitcell)/(1.813E-22)(cm3/unitcell) \tag{23}$$

$$\textbf{Density} = \mathbf{7.28\ gcm^{-3}}$$

Using the above for the density of the salts, and a density of 1027kgcm$^{-3}$ [10] for seawater, seawater was defined.

Once the detector medium was defined, the size of the detector had to be defined. For the initial studies a detector size of 2m, 2m, 10m (x, y, z) was used. At higher energies this proved too small, with some of the shower leaving the detector. As the whole shower was to be analysed, as long as the detector was greater than the shower dimensions, the size of the detector was arbitrary. It was hence decided to make the detector much larger than the showers - 10m, 10m, 100m.

The final alteration to the initial programme was to define a random number generator. The initial programme used the same seed for each shower, and hence got identical showers. To achieve the stochastic effects that were to be studied, a random number generator was used to seed each shower.

Having defined a suitable shower generator, many showers were generated from 10GeV - 20TeV. However beyond 20TeV the file sizes became too large for a single root ntuple. The code then had to be re-adapted so that Geant4 made multiple Root output files. Showers were then generated for 100TeV pions

### 2.3.2 Neutrinosim_Geant

**Code - neutrinosim_geant.cc - appendix B.4**

To model UHE neutrinos interacting with the water a programme called neutrinosim_geant was created. The programme uses equations 5 and 6 to recreate an acoustic pulse that a

hydrophone, a distance r from a shower, would receive. For the studies into the shower shape, only one hydrophone was used with the position of the shower fixed. The shower was generated at 1000m, at varying $\theta$. For most of the studies this angle was 0°.

Equation 5 recreates an acoustic pulse from an energy distribution. The programme was therefore required to take the showers generated from the Geant4 simulations and produce an acoustic pulse from these energy distributions.

As described §2.1 to calculate the seawater properties temperature, depth, and salinity are required. neutrinosim_geant also took as arguments the minimum angle, maximum angle, the name of the root input file (the output from the Geant4 simulation), and the name of the desired output file.

The minimum and maximum angle gives the range of angles, of the shower relative to the hydrophone, the acoustic pulse is to be calculated for. neutrinosim_geant loops over this range in integer steps calculating the acoustic pulse for each step.

As the Geant4 simulations were randomly generated, the energy distributions were never exactly the same. This meant that neutrinosim_geant had to be able to handle stochastic effects. The first stochastic effect that needed to be considered was finding the number of steps in each shower. Each shower was different, so if every step was to be considered a fixed number of steps couldn't be used. As all the shower data was stored in a Root ntuple, the number of steps was the number of entries in the ntuple, and hence could be found by asking ntuple for the number entries.

The effect that next needed to be considered was the varying shower centres. The shower centre is the z corresponding to the maximum energy. This needed to be found as equation 5 assumes a centre of energy, from which the angle is defined, and calculates the acoustic pulse in the positive and negative direction. The Geant4 shower, and hence the ntuple, is defined from the beginning of the shower. To find the centre of energy the ntuple was looped over and a graph of de/dz plotted. From this, the centre of energy was found by finding the z that corresponded to the maximum bin.

The next variable that needed to be considered was the time window that the pulse was to be calculated for. Using equation 24

$$time = \frac{distance}{speed} \tag{24}$$

the centre of the time window could be found. The distance is the distance to the hydrophone, 1000m, and speed is that calculated in the water properties code §2.1.

To calculate the acoustic pulse neutrinosim_geant takes the time window, splits it into a defined number of intervals and calculates the pressure pulse for each of these intervals. It is therefore very important that the time window and the number of intervals be defined in the most efficient way as to produce an accurate acoustic pulse using as less CPU time as possible.

The number of intervals gives the 'smoothness' of the acoustic pulse. The more intervals the more accurate the representation of the acoustic pulse, but also much more time consuming in calculating it. Also, the frequency that the Rona hydrophones operated at needed to be considered.

For the time window, the window couldn't be too large as time would be wasted calculating the pulse at irrelevant times, and the interval would be diluted making the acoustic pulse less accurate while still taking the same time to calculate. For a too small time window, information could be lost.

The window that was decided upon was $\pm 0.2$ ms with 512 intervals. This gives a frequency of 1.28 MHz.

## 2.4 Results and Analysis

### 2.4.1 Missing energy

neutrinosim_geant kept a sum of the energy in the shower. It was found that there was about 10% less energy than was expected from the input energy of the Geant4 simulations. This was because some of the contributions to the total absorption did not give rise to an observable signal. Examples are nuclear excitation and leakage of secondary muons and neutrinos [11].

It was not possible to test for nuclear excitation from the output of Geant4, but the log files of the showers contained detailed information on each step in the shower. To see how much energy was lost through neutrinos and muons escaping the detector the log file of 100GeV was analysed. The energy of the muons and neutrinos was noted and summed. This was compared to the recorded energy.

For the 100TeV pion the total energy in the shower was found to be 86.7847TeV, which gives a 'missing' energy of 13.2153TeV. The energy calculated from the neutrinos and muons was 12.48833TeV. This accounts for 94.5% of the total missing energy, with only 0.726973TeV unaccounted for. The unaccounted energy could be due to nuclear excitations, error in rounding, or ignoring <MeV energies.

This suggests that for future studies 10% of the energy will be lost to neutrinos and muons escaping the detector.

### 2.4.2 Comparison of shower shape

**Code - geant_ntuple_anal - appendix B.5** This Root macro reads in the relevant Geant4 output file and plots a 2D histogram as described above. To do this the macro finds the number of entries in the ntuple and loops over all of these entries. For each entry, the ntuple contains the position for the x, y, and z co-ordinate of the beginning and end of each step. The ntuple also contains the energy deposited for the step. An assumption was made that all energy is deposited at the midpoint of the step equations 25 and histograms for dz/de, dr/de, dphi/dE, and a 2D histogram for rz/dE were filled.

$$x = \frac{(x_{end} - x_{beginning})}{2} \tag{25a}$$

$$y = \frac{(y_{end} - y_{beginning})}{2} \tag{25b}$$

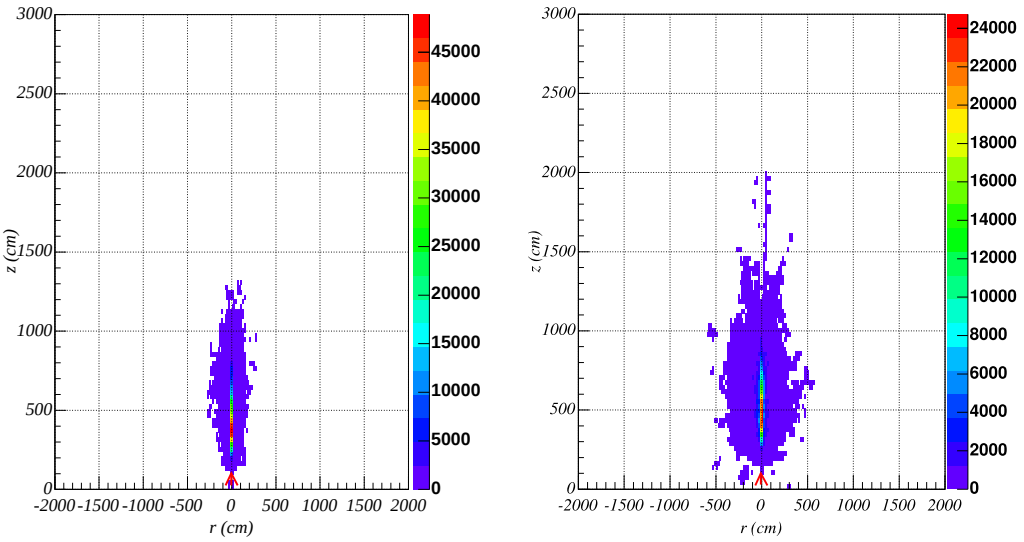
$$z = \frac{(z_{end} - z_{beginning})}{2} \tag{25c}$$

$$r = \left(x^2 + y^2\right)^{0.5} \tag{26}$$

$$\phi = a\tan\left(\frac{x}{y}\right) \tag{27}$$

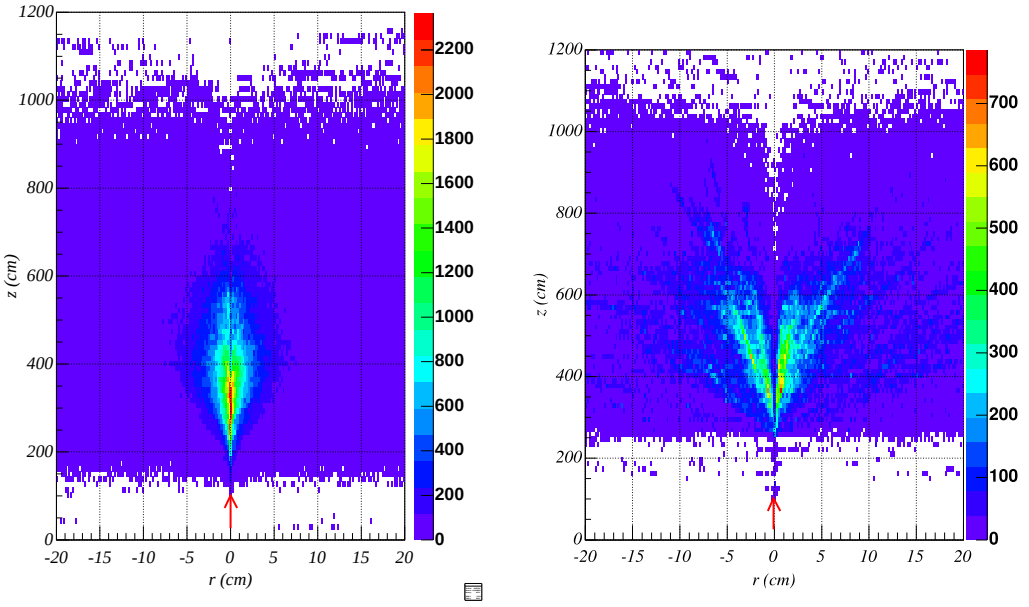
For dE/dz, a histogram was filled with z weighted by dE, for dE/dr, a histogram was filled with r weighted by dE, and for dE/d$\phi$, a histogram was filled with $d\phi$ weighted by dE. For the 2D histogram the histogram was filled with, r and z weighted by dE, but to make symmetrical about the shower origin, r was plotted positive for positive $\phi$, and negative for negative $\phi$.

**Analysis** To study the shower shapes, 2D histograms were plotted. The 2D histogram plots bins of r against z and weights the bins with the energy in each of the bins. Histograms of this type were plotted for all showers, for the total for both the shower and the core, -20cm < r < 20cm, of the shower.




(a) 1TeV electron (b) 1TeV pion

Figure 9: Entire showers




(a) 1TeV electron (b) 1TeV pion

Figure 10: Core of shower

**Comparison of electromagnetic and hadronic showers** The major visual difference is that at lower energies, for the core of the shower, electromagnetic showers are symmetric about the origin, and vary very little from shower to shower. This cannot be said for hadronic showers, which suffer greatly from stochastic effects. However, as the energy increases the core becomes more compact and much more symmetric, see figs 2.4.2

As would be expected, the energy of electromagnetic showers gets less as r and z increase. Again due to stochastic effects this is not true for hadronic showers at lower energies, where the energy can be quite similar over whole core. Again the stochastic effects become less prominent at higher energies. The consequence of this will be discussed in more detail in §2.5 when studying the shapes of the acoustic pulses.

Hadronic shower cores are bigger than the cores from electromagnetic showers. Although the core is bigger less energy, as a percentage of total energy is in the core. The hadronic showers also develop either side of r = 0 along the z-axis. This leaves an energy 'hole' in the centre of the shower. This can clearly be seen in fig 10.

The position of where the shower begins to develop varied between electromagnetic and hadronic showers. The Hadronic showers take about 2.0 meters from where the pion entered the detector to start showering. The electromagnetic showers begin where the electron entered the detector. This is important to note for later, where the shower is to be reconstructed using formulae.

**Comparison of hadronic showers for increasing energy** Figs 2.4.2 show the shape of the showers for increasing energy. The first comparison of the shape that should be noted is the increases in the length of the shower with energy, from 20m for a 1TeV shower to 30m for a 100TeV shower. This is exactly what is expected and is no surprise to find so. Similarly the width of the shower increases with increasing energy, but by very little. For 1TeV the width of the shower is a few cm under 5m and for 100TeV the width is a few cm over 5m. For increasing energies it can also be seen that the amount of energy scattered backward increases. The stochastic nature of the showers is also illustrated in these histograms, where high-energy particles can be seen to leaving the leaf shape of the shower in random directions.

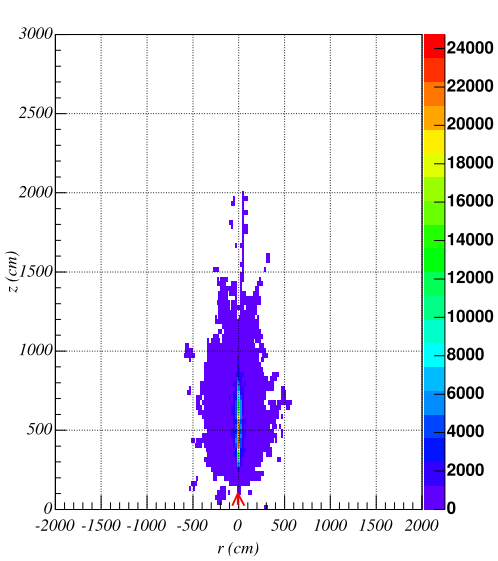
Of more importance is the core of the shower. It is the core of the shower where the acoustic pulse is generated, with the rest of the shower adding very little to shape of the acoustic pulse. Using figs 2.4.2 an estimate of the size of the core was made, -20cm $<$ E $<$ 20cm. Histograms were then plotted for the core. Figs 12 represent the results.

As the energy increases, the core of the shower becomes more condensed, and has greater symmetry both spatially and in energy. The 1TeV shower has a width of 20cm, the 20TeV shower has a width of 15cm, and the 100TeV a width of 8cm. For the 1TeV shower there appears to be very little structure, where for 100TeV a clear shape can be seen, with the energy distribution within this shape also having clear structure. This leads to a greater energy density at the centre of the shower. Also, as the energy increases the central hole, where the shower develops either side of r=0 becomes less prominent, and by 100TeV is hardly noticeable. The effects of this discussion will be discussed in greater detail in §2.5.

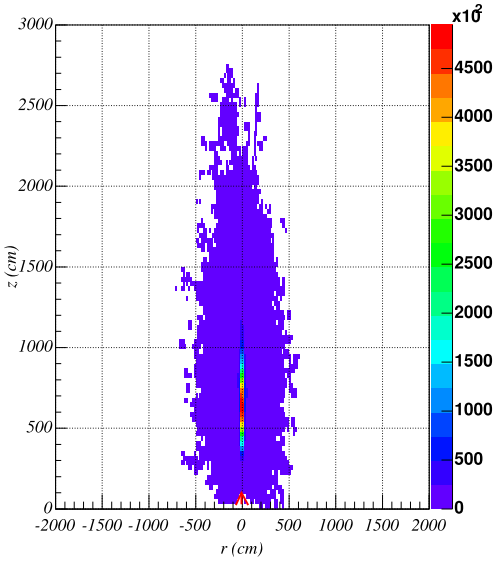
### 2.4.3 de/dz analysis

#### Fitting dE/dz

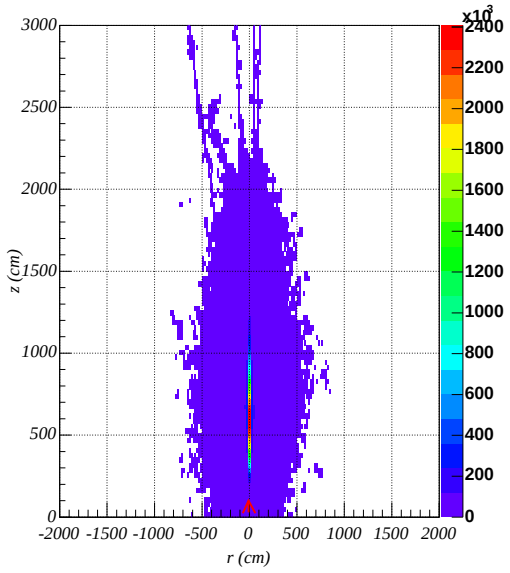
**Code - dedzfitter.C - appendix B.6** This macro loops through all the Geant4 output files for all energies and creates a histogram of dE/dz for each file. For every histogram errors were added to all bins. As no real errors exist, large errors were added to each bin so as to get a better fit on the actual data. Various errors were tested, from 0 - 750, and it was found that an error of 750 gives the best results. For points greater than one meter (as the pion entered the detector at 1m) the histogram was fitted with equation 29 below. For all histograms, the parameters of the fit were recorded in a log file.




(a) 1TeV

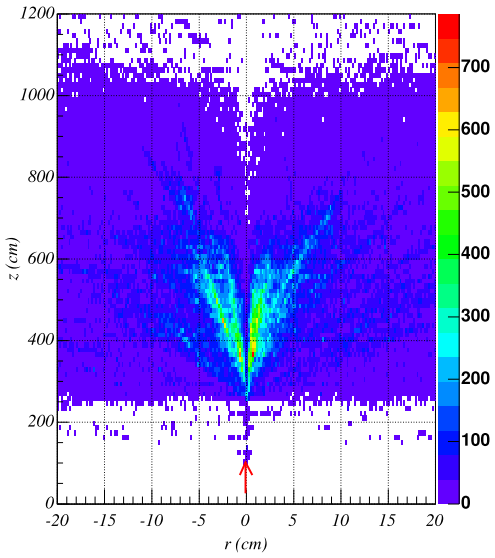



(b) 20TeV

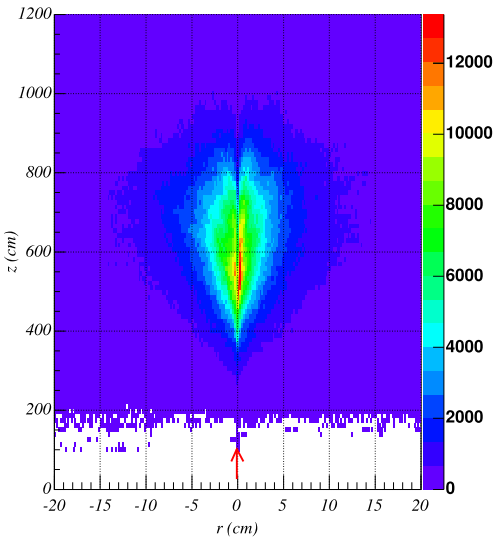



(c) 100TeV

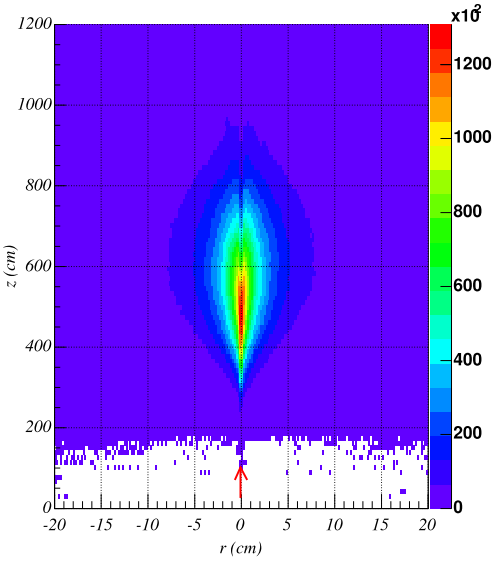
Figure 11: Full showers




(a) 1TeV




(b) 20TeV




(c) 100TeV

Figure 12: Full showers

To fit dE/dz equation 29 [12] is

$$\frac{dE}{dz} = k\left(wt^{a-1}e^{-bt} + (1-w)\,l^{c-1}e^{dl}\right) \tag{28}$$

t = dEpth, starting from shower origin; l = depth, starting from shower origin; ds = step depth; a, b, c, d = parameters fitted from data; w = realtive weight of the electric component; (1-w) = relative weight of the hadronic component; k = normalisation

Equation 28 has two parts, an electromagnetic part and a hadronic part. When investigating this equation it was found that the electromagnetic part had no effect on describing the structure of the dE/dz component of the shower. Equation 28 can hence be written as

$$\frac{dE}{dz} = p0\,(x-1-P3)^{p1}\,exp\,(p2\,(x-1-p3)) \tag{29}$$

Where p0, p1, p2, and p3 are the varying parameters which were used to best describe the shape and size of all dE/dz. (x-1) was used rather than simply x as it describes that the pion entered the detector at z = 1. p3 was added to allow for the varying starting positions of the shower, as discussed above in §2.3.1

### Parameterising the fit function

**Code - Paramfitter.C - appendix B.6** The log file created by dedzfitter.C give parameters for each fit, but for a general equation these parameters need to be paramatised. To investigate if equation 29 could be paramatised a Root macro was created to analyse these values. Paramfitter.C calculated the scatter on the parameters for each shower energy, and used this as the error for the parameter at that energy. The parameters were then plotted against the log of the energy.

Figs 13 show that for p1 and p2 the graphs reach a plateau at higher energies, where the stochastic effects become less important. The values for p1 and p2, obtained by taking the value of the line of best fit through the points from 10TeV and above, were found to be :-

**p1 = 9.00**
**p2 = -1.75**

Equation 29 becomes now a constant function of dE/dz, and hence zmax could be found by plotting equation 29 with p1 and p2 and finding the maximum bin.

**zmax = 5.22**

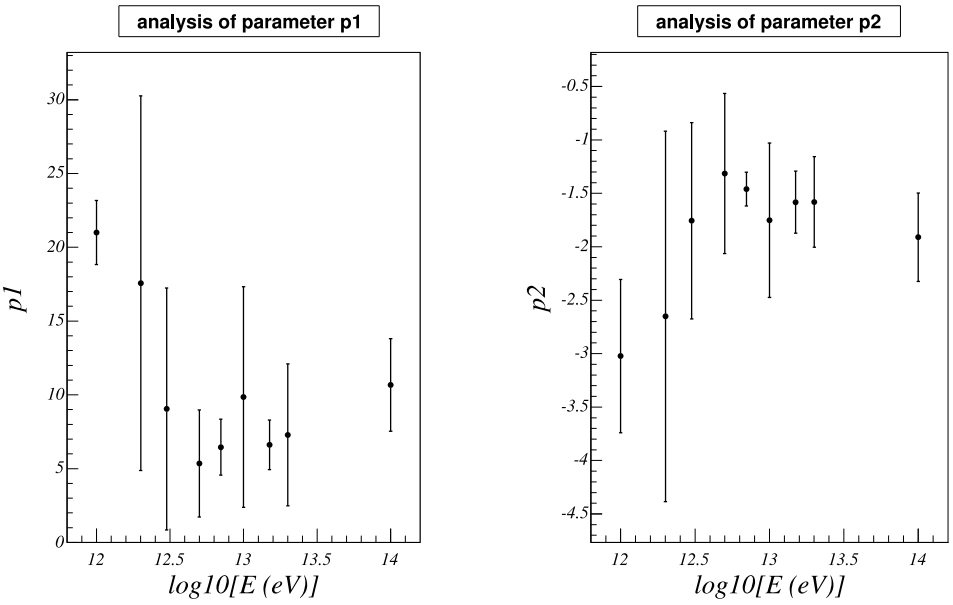
This value was used as the centre of the shower.

p0 has not been considered as it is the normalisation parameter. For recreating acoustic pulses the normalisation is performed on the reconstructed acoustic pulse, rather than the structure of the pulse. This will be discussed in more detail in section 2.5.

### General analysis

**Code - dedzcomp.C - appendix B.7** As neutrinosim_geant calculated the acoustic pulse, it also created histograms for dE/dz. dedzcomp.C looped through the output files of neutrinosim_geant and extracted all relevant dE/dz histograms. As with dedzfitter.C, it added errors to these histograms and fitted with equation 29.

Fig 15 show the comparison of dE/dz for showers of the same energy. These show that there is variation in dE/dz for all energies, with the scatter not becoming less for higher energies. The fig also shows that the value of 5.22 is a good approximation of the average value of zmax. As the scatter is still large for 100TeV it must be considered in the reconstructed pulse. As with normalisation, the sactter in dE/dz will only be considered on the reconstructed pulse.




(a) p1 (b) p2

Figure 13: calculating the parameters

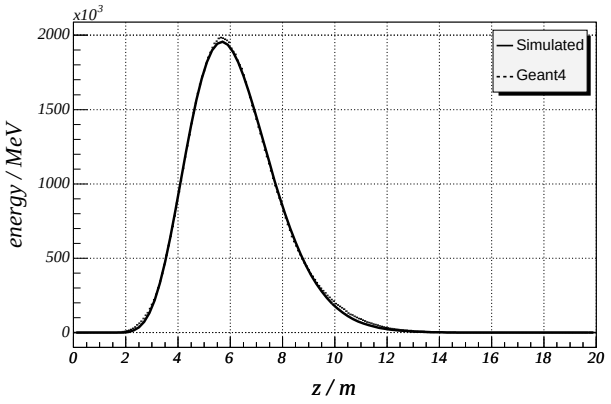



Figure 14: Simulated and reconstructed fit on dE/dz for 20TeV. This shows that equation 29 is a very good approximation of dE/dz

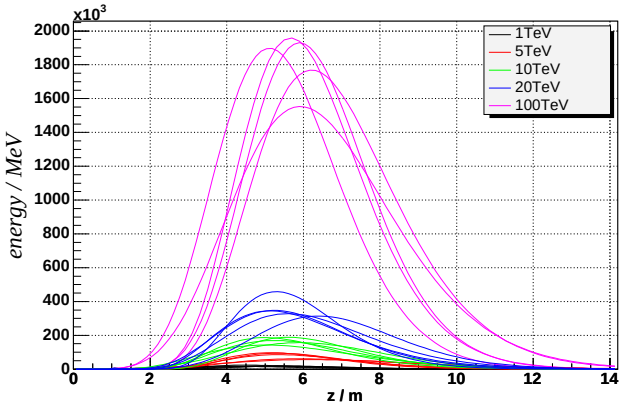



Figure 15: all dE/dz for all energies

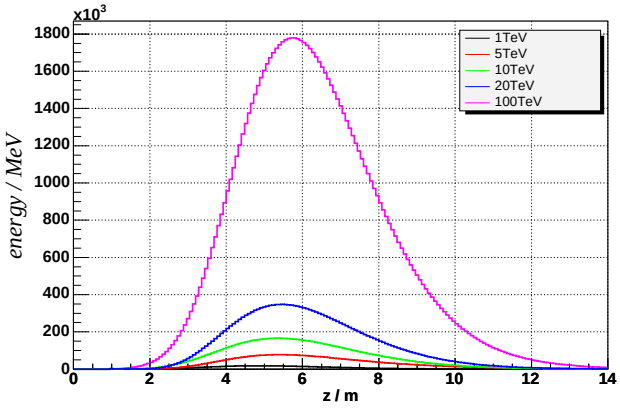



Figure 16: average dE/dz

Fig 16 shows the comparison of the average dE/dz at varying energies. The figs show the energy normalisation very clearly with the 100TeV curve having twice the maximum height as the 20TeV and 20TeV having four times the maximum height of the 5TeV curve. The curves also show that taking an average result still produces a smooth dE/dz curve, with very little variation in zmax between the energies. This is in good agreement with using a fixed parameter equation for dE/dz.

### 2.4.4 dE/dr analysis

#### Fitting dE/dr

**Code - dedrfitter.C - appendix B.8** This macro loops through all the Geant4 output files for all energies and creates a histogram of dE/dr for each file. For every histogram errors were added to all bins. As no real errors exist, large errors were added to each bin so as to get a better fit on the actual data. Various errors were tested, from 0 - 500, and it was found that an error of 500 gives the best results.

For points greater than 2cm (as fitting only the exponential decays of showers) the histogram was fitted with equation 31 below. For all histograms, the parameters of the fit were recorded in a log file.

The function that was used to fit the dE/dr histograms [[13] is

$$\frac{dE}{dr} = p0e^{\frac{-r}{p1}} + p2e^{\frac{-r^2}{p3^2}} \tag{30}$$

$$\frac{dE}{dr} = p0e^{\frac{-(r-p4)}{p1}} + p2e^{\frac{-(r-p5)^2}{p3^2}} \tag{31}$$

As with equation 29, equation 30 has two parts, one which describes small values of r, and the other describing large r. In order for equation 32 to be normalisable it must behave linearly with respect to one another and with respect to energy.

$$\frac{dE}{dr} = p0'\left[e^{\frac{-(r-p4)}{p1}} + p2'e^{\frac{-(r-p5)^2}{p3^2}}\right] \tag{32}$$

$$p0' = p0 \tag{33}$$

$$p0'p2' = p2 \tag{34}$$

$$p2' = p2/p0 \tag{35}$$

Hence if both parts behave linearly with respect to one another p2' must be constant. To investigate this the parameters for the fit of dE/dr were found using equation 32.

**Parameterising the fit function** From these parameters equation 33 could be investigated.

Fig 17 shows that p2' is not constant. As both parts could not be described using one normalisation constant, only one part of equation 32 could be used in the simulation. The acoustic pulse is generated from the core of the shower, so the exponential which describes small r was used. The final form of the fitting equation for dE/dr becomes: -

$$\frac{dE}{dr} = p0\left(e^{\frac{-(r-p4)}{p1}}\right) \tag{36}$$

As with dE/dz, p0 does not have to be parameterised as the normalisation and will be done after the acoustic pulse has been generated. Using figs 18, and taking the line of best fit for higher energies the parameters were found to be: -

| log10(E eV) | p2 | p0 | p2' |
|---|---|---|---|
| 12.00 | 9036.61 | 8704 | 1.04 |
| 12.30 | 4.65E8 | 24306.8 | 19139.05 |
| 12.48 | 3.54E7 | 29620 | 1194.78 |
| 12.70 | 7.24E8 | 120861 | 5990.353 |
| 12.85 | 3.21E10 | 459884 | 69821.96 |
| 13.00 | 5.72E8 | 53419.7 | 10707.66 |
| 13.18 | 2.58E9 | 143501 | 17978.97 |
| 13.30 | 1.02E10 | 270467 | 37712.55 |
| 14.00 | 1.31E11 | 927958 | 141170.2 |

Table 1:

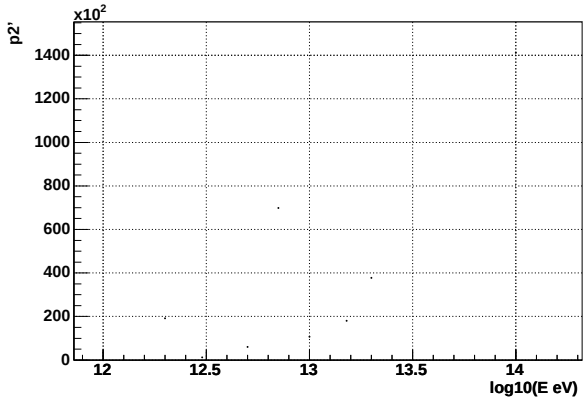
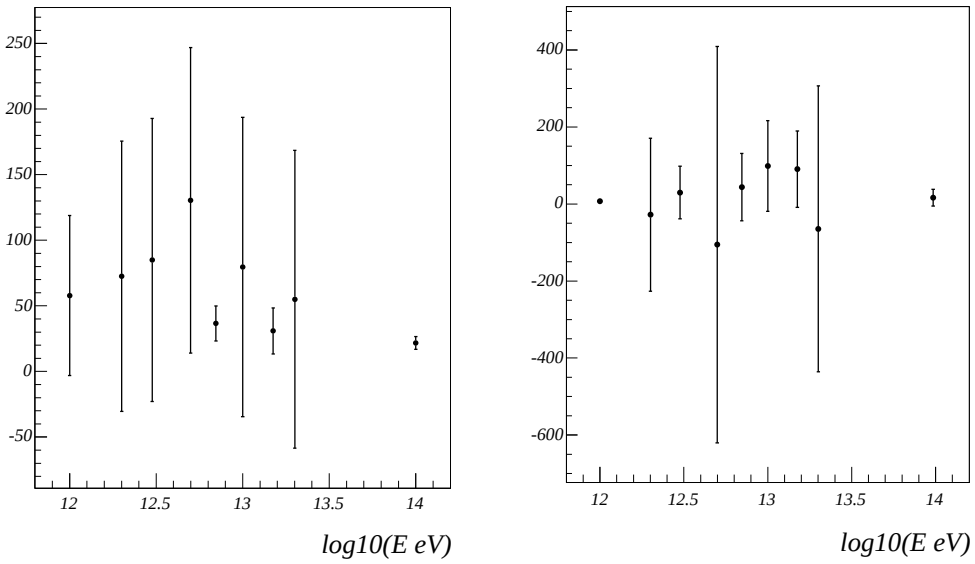



Figure 17: test for use of second exponential




(a) p1 (b) p4

Figure 18: calculating p1 and p4

$$\mathbf{p1 = 24.6522;}$$
$$\mathbf{p4 = 9.56813}$$

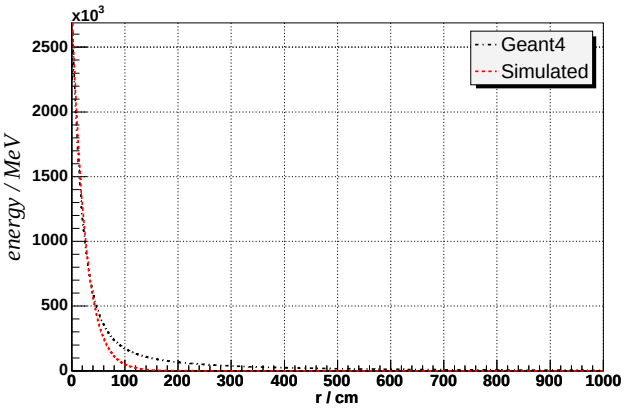


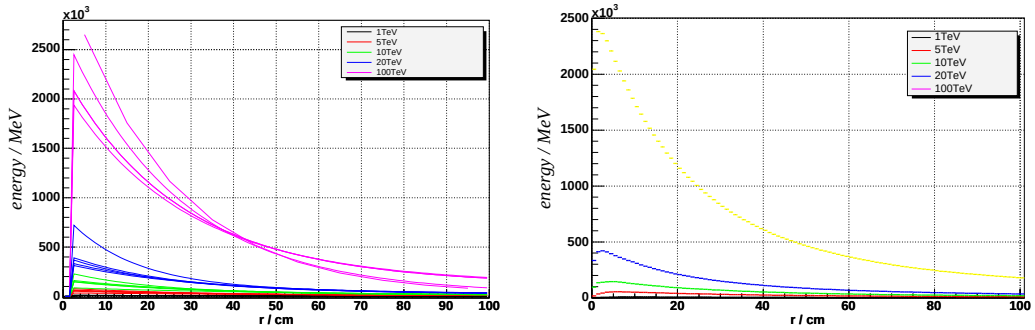

Figure 19: simulated and reconstructed fit on dE/dr

Fig 19 shows that the fit on dE/dr is only valid up to 70cm. This is because as stated in equation 36 the second exponential was not considered. This is because only the core of the shower generates an acoustic pulse, fig 24. For r $<$ 70cm the fit is very accurate

**Code - dedrcomp.C - B.9** As neutrinosim_geant calculated the acoustic pulse, it also created histograms for dE/dz. dedzcomp.C looped through the output files of neutrinosim_geant and extracted all relevant dE/dz histograms. As with dedzfitter.C, it added errors to these histograms and fitted with equation 36 .

**Code - dedrdzmaker.C B.10** This macro behaves very similarly to geant_ntuple_anal.C. The macro loops through all Geant4 output files and reads in the ntuples. For every step in the ntuple the x, y, z, and r are calculated using equation 25. Unlike geant_ntuple_anal.C, instead of filling a histogram with all r, z is taken into account. For each step z is analysed, and histograms filled according to the value of z. Histograms were created for z from 0-10 in integer steps, e.g. 0-1, 1-2, and the corresponding histogram filled with r, weighted by de.

**Code - dedrdzplotter.C B.11** This macro is an extension of dedrdzmaker.C. Once the histograms are created by dedrdzmaker.C, this macro reads in the output file extracts the figs and plots them. The macro also creates average histograms for each energy.




(a) all dE/dr for all energies

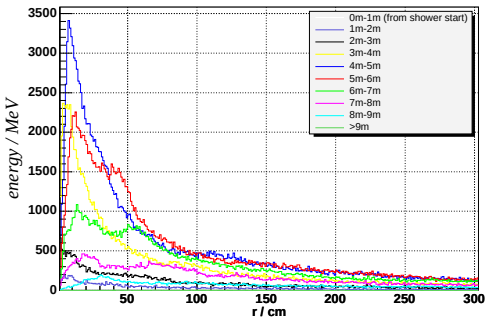
(b) average dE/dr

Figure 20: dedr

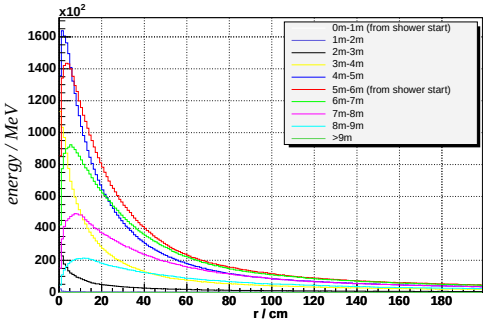
**Analysis** Fig 20 shows the comparison of dE/dr for increasing shower energies. For each shower energy five showers are shown. The scatter for each shower energy is up to ±50% of the average value, and does not decrease with increasing energy. Although the scatter is large the average values clearly shows the normalisation of energy.

The interesting point to note about the shape of dE/dr is the non zero maximum. This reflects the earlier studies into the shape shower, §2.4.2, where it was noted that the showers develop around a central 'hole.' For higher shower energies the hole becomes less prominent. This is reflected in fig 20 where it can be seen that the non-zero peak appears at lower energies for increasing shower energies. This quantifies the assumption that at UHE neutrino energies, the maximum energy of the shower peaks is r = 0. This also means that the assumption of using an exponential description of dE/dr, equation 36, can be assumed to be true for all values of r.

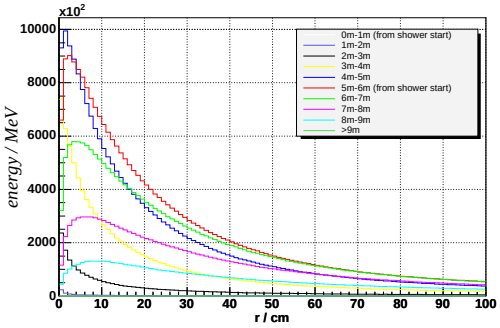
Fig 21 shows that the slice corresponding to z = 4-5 metres is the most prominent, followed by the slice for 5-6 meters. This is agreement with the maximum shower energy at z = 5.2 meters, and hence the shower centre at z = 5.2 meters and r = 0.0m. Fig 22 also shows the non-zero peak disappearing as the energy of the shower increases.




(a) 1TeV




(b) 20TeV




(c) 100TeV

Figure 21: dE/dr as a function of z

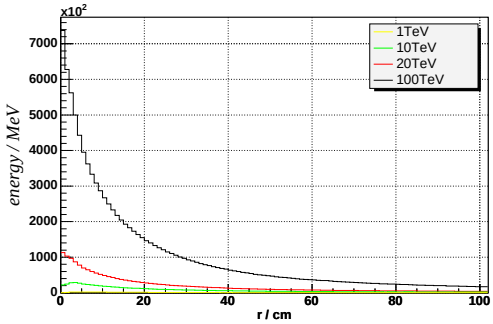
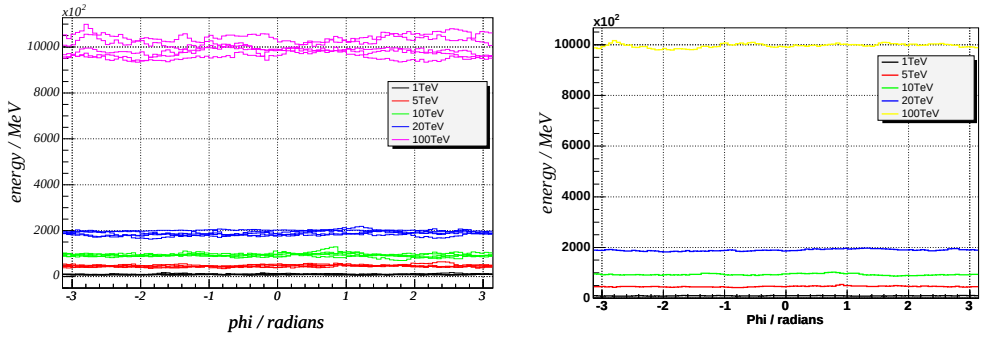



Figure 22: showing only slice 4

### 2.4.5 dE/d$\phi$ analysis

**Code - dedphicomp.C - B.12** As neutrinosim_geant calculated the acoustic pulse, it also created histograms for dE/d$\phi$. dedphicomp.C looped through the output files of neutrinosimgeant and extracted the dE/d$\phi$ histograms. The macro also created average histograms for every shower energy.




(a) all dE/d$\phi$ (b) average dE/d$\phi$

Figure 23: dedphi

Equations 36 and 29 assume that the shower is symmetric in $\phi$. Fig 23 shows the comparison of dE/d$\phi$ for increasing shower energies. For all shower energies five showers are shown.

Fig 23 shows that the assumption that showers are symmetric in phi is a good assumption, and equations 36 and 29 still hold true. It can also be seen that dE/dphi is not a perfect straight line, as would be expected from stochastic effects.

## 2.5 Pulse analysis

**Code - pulse_comp.C - B.13**

### 2.5.1 Comparison of acoustic pulse generated from the whole shower and the core of the shower

Acoustic pulses were generated using all of the steps from the output of the Geant4 simulation, and steps just from the core of the same shower. It was found that just analysing the core, -20cm < E <20cm provided an accurate representation of the shower, fig 24, but also contained the majority of steps the total shower steps.

| E TeV | steps in whole shower | steps in core of shower | percentage in core |
|---|---|---|---|
| 1 | 3.20E6 | 2.77E6 | 87 |
| 10 | 3.20E7 | 2.97E7 | 93 |
| 20 | 6.59E7 | 5.99E7 | 91 |
| 100 | 3.29E8 | 2.93E8 | 89 |

Table 2:

This shows that the acoustic pulse is generated from the core of the shower, and the analysis of §2.4.2 must be considered when discussing the variations in acoustic pulses. This also shows that using a single exponential to describe dE/dr was a good assumption.

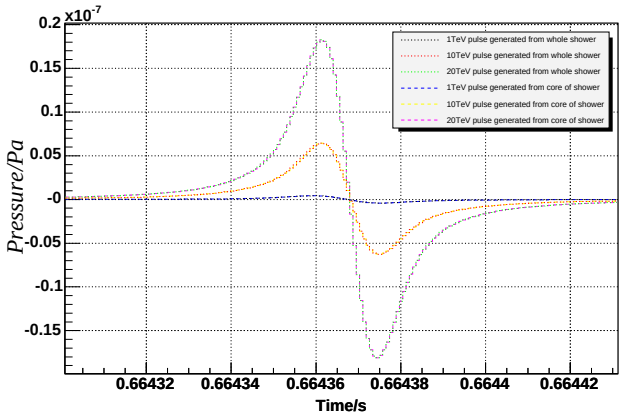



Figure 24: Acoustic pulse generated from the core and the whole shower

### 2.5.2 Comparison of 1TeV electromagnetic and hadronic showers

Acoustic pulses were generated for electromagnetic showers and hadronic showers in sea water at 1TeV.

Figs 25 show the comparison of the individual acoustic pulses for electromagnetic and hadronic showers. It can be seen that the electromagnetic pulses give a very defined pulse, where the hadronic pulses suffer greatly from stochastic effects. This leads to a scatter in pulse height and time of peak pressure. For 1TeV hadronic pulses, the peak pressure varies from the average peak pressure, $0.5.10^{-8}$Pa, by about 20%.

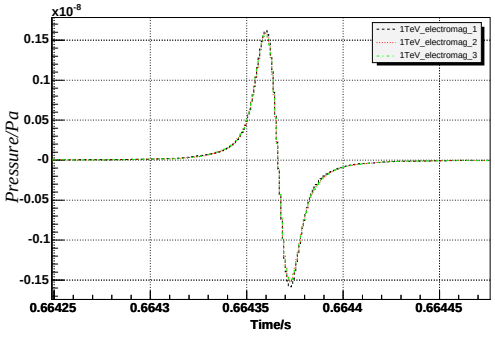
As discussed above, the acoustic pulse is generated in the core of the shower. From the analysis of §2.4.2 it was found that 1TeV electromagnetic shower cores are compact and symmetric about the origin, with very little variation between showers. Hadronic shower cores are larger, un-symmetric, and have an even energy distribution over the core. The results shown in figs 25 are what would be expected from these observations, with very predictable electromagnetic acoustic pulses, and stochastically varying acoustic pulses.

Another result that would be expected from the discussion in §2.4.2 is that the electromagnetic shower would produce an acoustic pulse with a greater peak pressure. This is because hadronic shower cores are bigger than the cores from electromagnetic showers, with less energy, as a percentage of total energy is in the core. The hadronic showers also have an energy 'hole' in the centre of the shower. It can be seen in fig 25 that the maximum pressure of the electromagnetic pulse is three times larger than the hadronic pulse.

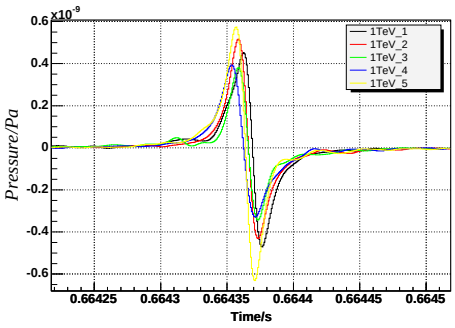
### 2.5.3 Comparison of 1TeV hadronic acoustic pulses in seawater and freshwater

Showers were generated at 1TeV in seawater and in freshwater. The acoustic pulses were calculated for these showers.

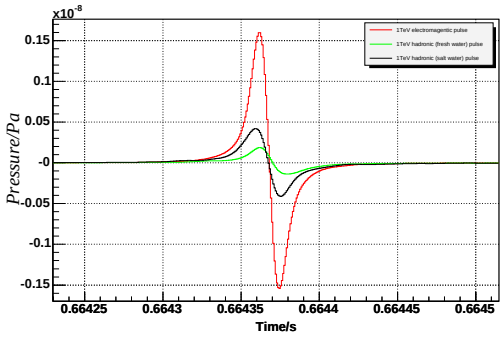
Fig 25 show the results. The time window fro fresh water is different to the time window for salt water. This is due to the different water properties for fresh and salt water. For ease of comparison the freshwater time window has been moved to the saltwater time window.The more important comparison that must be made is the shape of pulses. The acoustic pulses generated in fresh water have less smooth pulses, and appear to have a smaller average maximum pressure. However, no fair conclusions can be drawn from these results. Only three pulses were generated for freshwater, compared to the five for seawater, and only 1TeV showers were generated. To make any comparisons for different types of water, more results need to be taken for fresh water.




(a) pulse generated from an electromagnetic shower



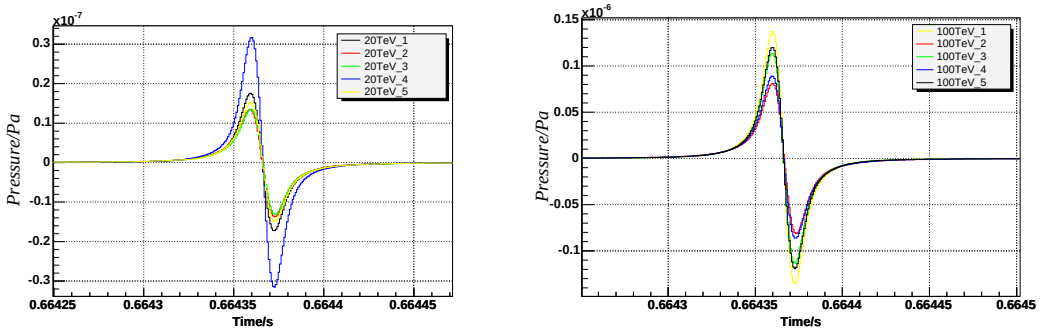

(b) pulse generated from an hadronic shower




(c) comparison of all showers

Figure 25: 1TeV

### 2.5.4 Comparison of the acoustic pulses from showers of the same energy




(a) 20TeV

(b) 100TeV

Figure 26: Hadronic Showers

Figs 26 show the results for energies of 100, 20, 10, 5, and 1TeV for hadronic showers generated in seawater. The 1TeV pulses have been briefly described in§2.5.2, using the analysis of §2.4.2. For the analysis of acoustic pulses at varying energies, the analysis of §2.4.2 will be used also. The first comparison to be made is the variation of maximum pressure for each shower at increasing energy. Using fig 26 an error can be estimated for the variation of the pulse heights compared to the average pulse height, fig 26.

1TeV - 20%
5TeV - 50%
20TeV - 40%
10TeV - 40%
100TeV - 25%

These values show that, for the energies investigated, there is no correlation between energy and error. There is also no correlation between the errors. An average error can be estimated at 35%. This value will be used to scatter the acoustic pulses generated using equation 5.

It can also be noted that as the energy increases, the influence of stochastic effects on the acoustic pulses becomes less important. This is as expected from the studies of §2.4.2. Also as the energy increases, the spread in the width of the pulse becomes insignificant, with only the spread in the maximum height of the pulse important.

### 2.5.5 A comparison of the average acoustic pulse at varying energies

The average pulse was found for each shower. The figs show that as the energy increases the time of the pressure maximum stays constant. This is as expected, as the properties of the seawater were constant for all showers. For higher energies the pulse height scales linearly; for 20TeV the average height is 0.02E-6, and for 100TeV the average pulse height is 0.1E-6. This is an important observation, as this will be assumed when scaling acoustic pulses using equation 5.

### 2.5.6 A comparison of the acoustic pulse for slices in z

**Code - neutrinosim_geant.cc - appendix B.4** This is an extension of neutrinosim_geant.c. The extension to the code allowed acoustic pulses to be calculated for slices of z. Acoustic pulses

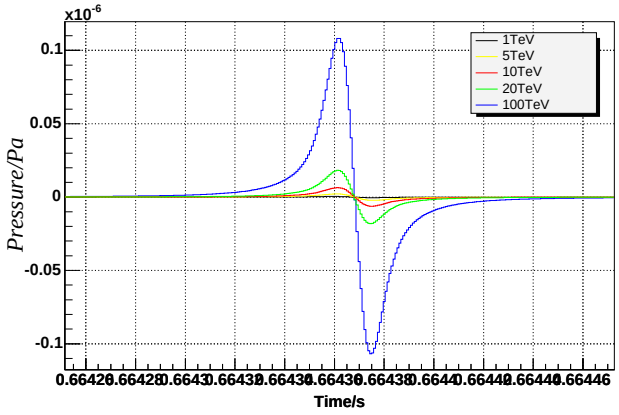



Figure 27: average hadronic showers

were generated for z from 0-10 in integer steps, e.g. 0-1, 1-2. Counts of the number of steps in each slice were also found.

### Code - pulsededrdz.C - appendix B.14

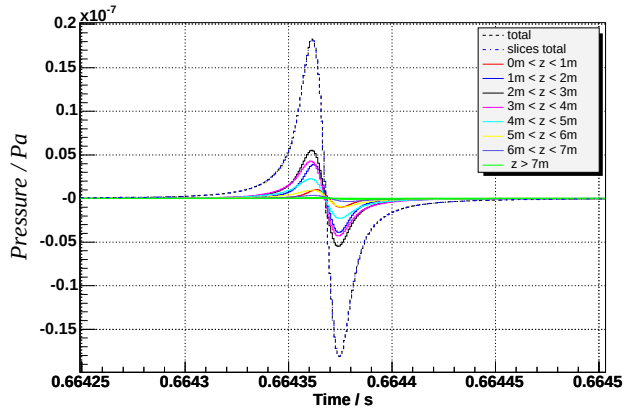



Figure 28: acoustic pulse as a function of z

**analysis** Slice 4-5 is the most dominant. There are also only 4/5 slices that actually add up to give the pulse -2 - 2

## 2.6 Angular dependence of peak pressure

Fig 29 shows an example of the variation of peak pressure with angle. As expected, the peak is at 0 degrees, and the fig falls either side of this. By $-6° < \theta < 6°$ the peak pressure has fallen to almost zero. $|6°|$ was hence used as a cut in the calculation of acoustic pulses. If a shower was generated at an angle greater than $|6°|$ to the hydrophone, the acoustic pulse was not calculated. This saved computer time, and more showers could be generated. It can also be seen that the graph is asymmetric.

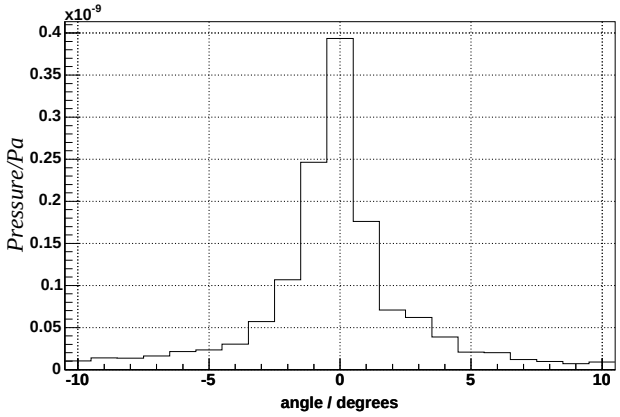



Figure 29: showing the angular dependence peak pressure

## 2.7 Acoustic Pulse reconstruction

**Code - pulse_reproducer.C B.15** Using the analysis from §2.4.2 the core of the shower was defined to be a cylinder of 0.3m in r by 10m in z. Using the energy distributions as defined in equation 36 the cylinder was filled with the required energy distribution. From this the acoustic pulse was generated using equation 5, in a similar way to generating acoustic pulses using the simulated showers. The pulse was then scaled accordingly for the required energy.

$$E(r,z) = E_r(z)\, xE_z(z) \tag{37}$$

$$E = \Sigma de \tag{38}$$

$$Scale = \frac{energydesired}{sumofE(r,z)} \tag{39}$$

During the reconstruction of the acoustic pulse, this and a further factor of 0.74 was used to scale each individual pulse calculation. The further 0.74 comes from various factors. It was noted in §2.4.1 that about 10% of the energy was lost due to escaping neutrinos and muons. It was also noted that the peak of dE/dr was not at 0, and hence an exponential would be an over estimate of energy for low r, §2.4.4. The pulse therefore had to be scaled by this additional factor, which was derived by comparing the simulated pulse to the reconstructed 100TeV pulse.

Graph 30 show the results for acoustic pulses generated using Geant4, and acoustic pulses generated using equation 5. The acoustic pulse agrees very well for 100TeV, but for lower energies the equation-generated pulse is an over estimate. For higher the energies, the lower the error for the equation-generated pulse against the Geant4 generated pulse. Therefore, this argument can be extrapolated to higher energies and can assumed that any equation-generated pulse will be an under estimate of the true pulse.

## 2.8 Summary

At UHE neutrino energies, the cross section of the neutrino is sufficiently large as to make the Earth opaque. The neutrinos interact with nucleons on the Earth, via deep inelastic scattering, fig2, and shower hadronically. The hadronic shower can be considered as an instantaneous 'dump' of energy. In seawater, this energy is sufficiently large as to expand the water surrounding the shower, creating a di-polar acoustic pulse, fig 30. Simulating hadronic showers using Geant4, acoustic pulses were created and studied.

Using Geant4 hadronic showers were simulated up to 100TeV. From analysing the structural form of these showers, a parameterised form of the shower was developed.

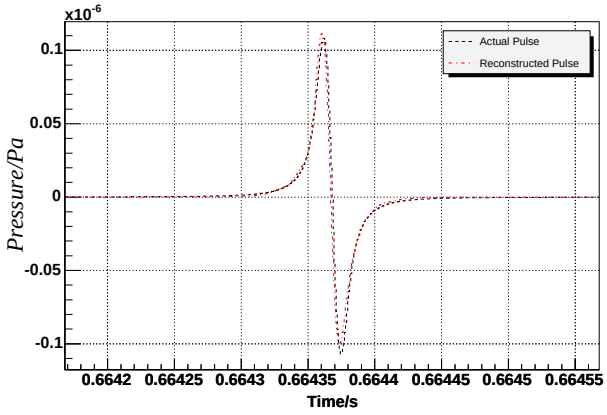



Figure 30: Reconstructed and simulated pulse

$$\frac{dE}{dr} = \left(e^{\frac{-(r-9.56813)}{24.6522}}\right) \tag{40}$$

$$\frac{de}{dz} = (x)^{9.00}\, exp\,(-1.75x) \tag{41}$$

It was also noted that 10% of the energy was lost due to neutrinos and muons escaping the showering region.

Using the Geant4 showers and equations 5 and **??**, acoustic pulses were generated. Equations 5 and **??** require the thermal expansivity of water, the specific heat capacity, and the velocity of the acoustic pulse in the water. These in turn depend on salinity, depth, and temperature of the water. A programme was written that allowed the seawater properties to be calculated as a function salinity, depth and temperature.

The study of these acoustic pulses showed that by 100TeV the pulse shape is very defined, fig 26, with only the peak pressure varying, by 35%.

Using equations 40, hadronic showers were recreated and the corresponding acoustic pulses found. Fig 30 shows the results. Fig 30 shows that the parameterised acoustic pulse is a very accurate representation of the actual acoustic pulse.

The angular variation of peak pressure was also investigated, 29. It is suggested only for $\theta < |6°|$ should the acoustic pulse be calculated.

# 3 Array Studies

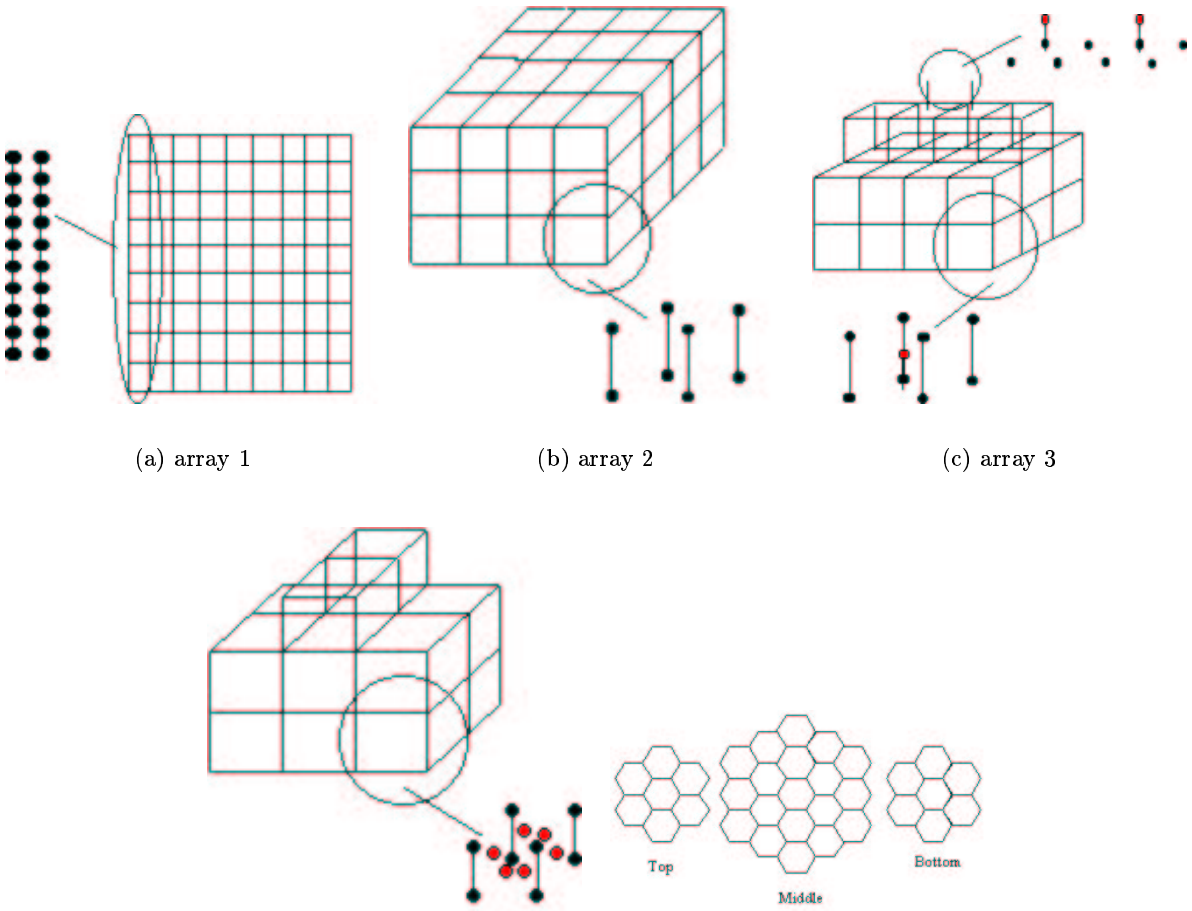
## 3.1 Introduction

This section of the report uses the analysis of section 1 to simulate a large number of hadronic showers, and investigate the efficiencies of various hydrophone array designs. The efficiency of an array is determined by the number of events detected for coincidences of 3, 4, and 9 hydrophones for varying minimum pulse amplitudes. Coincidences rather than single events determine the efficiency, as the arrays are to be used to reconstruct the position of the shower. Coincidences will, in the actual detector, help extract the acoustic pulse from the noise.

For each array design 50,000 events were generated in a sphere of radius 10,000m with the arrays positioned at the centre of the sphere.

## 3.2 Array designs

The arrays were designed using 100 hydrophones, a realistic number for an actual hydrophone array. Five array designs were investigated, fig 31.




(a) array 1 (b) array 2 (c) array 3

(d) array 4 (e) array 5

Figure 31: arrays

## 3.3 Array spacing

The five arrays designs above were initially investigated using a spacing of 100m between the hydrophones. 100m is an arbitrary number and does not represent the maximal array design. Using the results from the 100m spaced arrays, the array that performed best was chosen and the hydrophone spacing investigated.

Ideally this test should be performed on all arrays, and only from these results the best design chosen. Due to the computational time of generating 50000 events per array per hydrophone spacing, this was not possible.

## 3.4 Neutrinosim

**Code - neutrinosim.cc - appendix B.16** This is the main programme for the study of arrays. This programme generates a shower at a random position, in a random direction, and at a random energy, §2.2 , and calculates the acoustic pulse at given a hydrophone, §2

Neutrinosim is similar in many ways to neutrinosim_geant, but also varies in many ways. Unlike neutrinosim_geant, which calculated only one event per run, neutrinosim allowed for many many events to be generated per run. Neutrinosim also allows the speed of the simulation to be set. This is determined by setting the number of integrations over the shower for which neutrinosim calculates the acoustic pulse. The greater the number of integrations the more accurate the representation of the acoustic pulse, but the slower the programme. For the study into arrays 10000 integrations were used. This number was determined by starting at a high number to get an accurate pulse and decreasing the number until the pulse is no longer acceptable.

As with neutrinosim_geant, temperature, depth, and salinity were given for the calculation of the water properties. Again the numbers used were the values of the Rona array, T = 13°C, salinity = 35psu, and a depth of 300m.

Similar to neutrinosim_geant, neutrinosim asks an input file and the name of the desired output file. The only difference being that the input file is now an array design instead of a Geant4 output file.

### 3.4.1 Defining arrays

Above it was stated that neutrinosim takes as an argument an array file. An array file is text file that defines an array of hydrophones. The arrays were defined with the centre of the array lying at 0,0,0 and each line of the text file giving the x, y, and z co-ordinate of a hydrophone in the array. From this neutrinosim was able to read in each hydrophone separately and calculate the acoustic pulse at the hydrophone. Using this method data files were created for figs *******

### 3.4.2 Writing a log file

For any analysis, the results of neutrinosim had to output in a suitable format. It was decided that the results were to be placed in a log file. This log file had to be filled in a suitable format so the results could be read and analysed by another programme.

For every array, and hence log file, there were constant values. These were the hydrophone co-ordinates and the water properties. Although not relevant to this part of the studies, for completeness, the temperature, depth, salinity, velocity of the acoustic wave in water, the thermal expansion co-efficient, and the specific heat capacity of the water were written to the top of log file.

The next lines in the log file were the array data. Firstly neutrinosim opened the array file, counted the number of detectors, and printed this number to the log file. Knowing the number of hydrophones, neutrinosim was then able read in the array and print the x, y, and z co-ordinates of the hydrophones to the log file. The remainder of the log file was the results of the simulation.

As discussed above, for every event a random position, direction, and energy was generated. Using these values neutrinosim looped through every hydrophone in the array and calculated the distance of the hydrophone from the shower and the angle of the wave front with respect to the hydrophone. A decision was then made whether to calculate the acoustic pulse. For -6.0°C $< \theta <$ 6.0°C (§2.6), if the distance between the shower and the hydrophone was less than 1000m, the acoustic pulse was calculated.

If the acoustic pulse was calculated the peak pressure and the time of the peak pressure was written to the log file. If the acoustic pulse was not calculated 0's were written to the log file. Thus, for every line in the log file, one event, the log file contained the shower details and then details of the pressure pulse at every hydrophone in the array.

## 3.5 Results

To study each array, 50,000 events were generated. For consistency, all events were simulated with the same parameters and random number seeds.

### 3.5.1 Minimum peak pressure at the hydrophone

For hydrophone arrays built in seawater, background noise from the seawater environment will make the detection UHE neutrino acoustic pulses more difficult. If the amplitude of the signal is too small, the pulse will be indistinguishable from the noise. Noise is expected to come from biological sources as well as from point sources. Point sources of noise will be produced mainly from maritime vehicles. The nature of the noise will not be fully understood until studies are carried out using the Rona array. It needs to be determined if the noise can be described using a Gaussian function, if the noise isotropic, and how point sources, eg ships, will effect coincidences.

The consideration of noise also poses the question of where to place the array. As discussed in section §2, as the temperature of the water increases the coefficient of thermal expansivity increases, fig 4, but so does the biological noise.

The effect noise will not be known until measurements are made using the Rona hydrophones, but must still be considered in the study of arrays. To study the effect of noise on the detection of UHE neutrino acoustic pulses it was assumed that only pulses above a minimum amplitude could be extracted from the noise, and pulses below this amplitude were indistinguishable. This minimum amplitude will be referred to as the hydrophone cut. For these studies hydrophone cuts of 0.025Pa - 0.3Pa were used. 0.3Pa is a very severe cut, but is non-the less interesting to see what effects this has on the detection of the acoustic pulses.

### 3.5.2 Coincidences at varying hydrophone cuts

**Code - array_log_anal.C - appendix B.17** Loops over all of the array log files and creates a histogram for coincidences from 0-100, where each bin represents a coincidence, for all hydrophone cuts

**Code- array_anal_plotter.C - appendix B.18** Manipulates the histogram created array_log_anal.C to make graphs 32

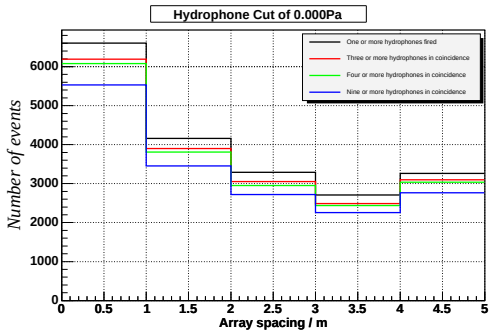
**Analysis** Considering an error of $\sqrt{n}$, figs 32 allow the efficiency of the arrays to be studied. For a hydrophone cut of 0.025Pa, array one performs the best for all coincidences.

Once the hydrophone cut is increased, the arrays performance begins to even out. For 0.05Pa, array 1 is still the most efficient for coincidences of 3 and 4, but for a coincidence of 9 array 3 appears to perform better. Considering an error of $\sqrt{n}$, all arrays lie within the error bars of the other figs. It no longer becomes statistically relevant to single out one fig, and all arrays can be said to perform equally. For a cut of 0.1Pa, array1 again appears to be the most efficient, but again considering an error of $\sqrt{n}$ it is no longer a convincing winner. As the cut is increased further the number of events detected decreases to a point at a cut of 0.3Pa where no conclusions can be drawn from the results as the number of detected events is too low.

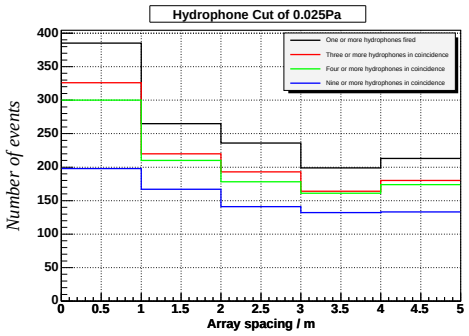
Array 1 appears to be dominant for these early studies, but only by a tiny margin. Other factors also need to be considered in choosing the final array design. These include how well the array reconstructs the position and direction, and engineering considerations. No conclusion about the optimal array design can be drawn from these results, as more study needs to be performed into hydrophone spacing and clusters of arrays.

### 3.5.3 Performance of arrays at varying energy

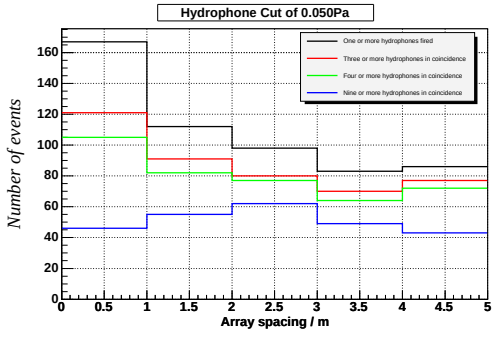
**Macro - array_log_further_anal.C** Loops over all arrays and creates a histogram of the number of events detected over the entire energy range for all cuts and coincidences.



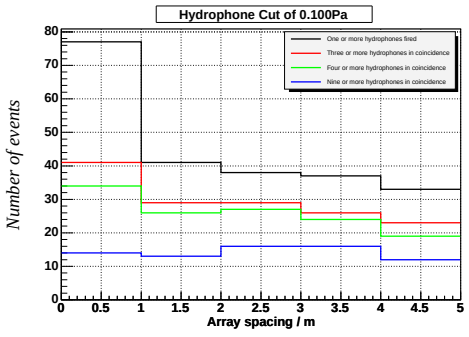

(a) no cut



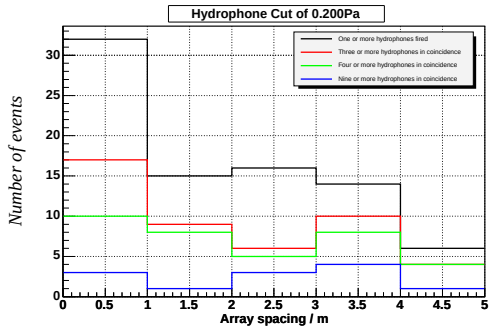

(b) cut of 0.025Pa



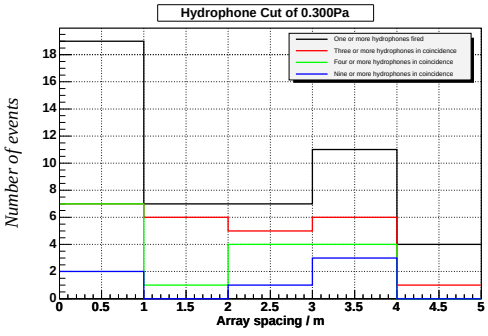

(c) cut of 0.050Pa




(d) cut of 0.100Pa




(e) cut of 0.200Pa




(f) cut of 0.300Pa

Figure 32: array efficiencies at varying hydrophone cuts

**Macro array_further_anal_plotter.C** Manipulates the histogram created array_log_further_anal.C to make figs 33

**Analysis** As discussed above, array 1 performs best over the entire GZK energy region, but do the various array shapes perform differently over different energy regions? To look at this, histograms were plotted over the energy region $1.10^{17}$ - $1.10^{20}$ for varying hydrophone cuts and for coincidences of 3, 4, and 9 for all arrays.

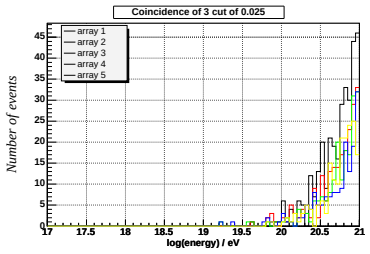



Figure 33: array efficiency as a function of energy

For a full set of results see appendix A.1

Fig 33 shows that all arrays behave very similarly in this energy range, with array one again showing its dominance. The interesting point that this fig shows is that for a hydrophone cut of only 0.025Pa no acoustic pulses from showers with energy less than 1E20 are detected. This means that either this method will only be able to detect the highest energy neutrinos or that the hydrophone cut must be lowered.

As discussed in section §1.7, detection over the entire GZK flux would be preferable, which means that the hydrophone cut must be investigated. This will be something that studies using the Rona arrays will address.

### 3.5.4 Performance of arrays at varying shower distances

**Code - array_log_further_anal.C** Loops over all arrays and creates a histogram of the number of events detected over the entire radius of the sphere for all cuts and coincidences.

**Cope - array_further_anal_plotter.C B.19** Manipulates the histogram created array_log_further_anal.C to make figs 34

**Analysis** For a full set of results see appendix A.2

As with the energy, an interesting question to ask about the arrays is the ability of the array to detect showers at varying radii. For arrays 2-4 fig 34 shows a parabolic curve. This is as expected. As the radius increases the effective volume of the sphere increases and hence more events are generated at larger r, but also as the distance of the shower to the hydrophone increases the amplitude of the pulses decrease. The shape describes the optimum distance of detection. This will change depending on the hydrophone spacing.

For all arrays there appears to be fine structure in the curve described above. This has not been investigated further, but looks like it is an artifact of the hydrophone spacing.

Fig 34 also shows that by 10km almost zero events are detected. This means that choosing a sphere of radius 10km was the correct choice.

For array one, this curve doesn't appear to exist. This due to the cut of 1000m not being the correct cut to use.

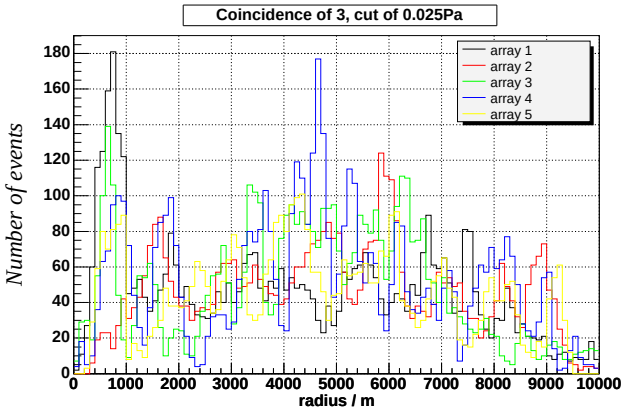



Figure 34: array efficiency as a function of radius

### 3.5.5 Performance of array at varying hydrophone separations

**Analysis**

### 3.5.6 Reconstruction of r

**Macro - recon.C - appendix B.20** This macro loops over all arrays and creates a fig of reconstructed distance against actual distance. To do this the log files of the 50000 event simulations were read in and the numbers stored in arrays. For each event, the maximum peak pressure is found and used as a reference. From this reference hydrophone two more hydrophones on the same string are found. For each of these hydrophones, the hydrophone number (and hence its position), the time of maximum peak, and the radius are recorded. The macro then checks that there are three hydrophones in coincidence and arranges them in ascending order. Using equations 42 and 43 the reconstructed radius is calculated and a plot made.

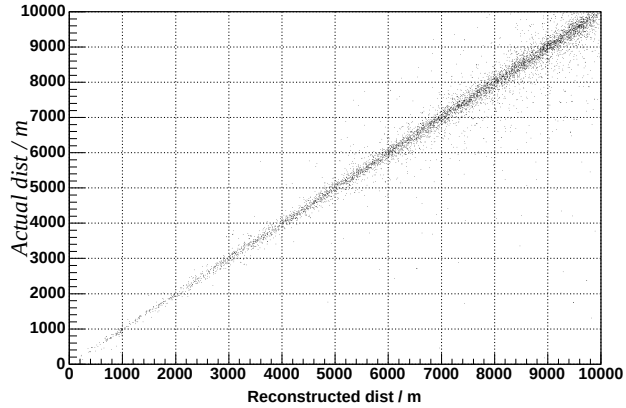



Figure 35: radius reconstruction

**Analysis** Graph 35 shows the reconstructed distance, the distance of the shower from the hydrophone calculated from the difference in arrival times of the acoustic pulse, plotted against

the actual distance, the distance from the log file. Fig 35 shows that these simple formulae prove very successful in reconstructing the radius.

Considering

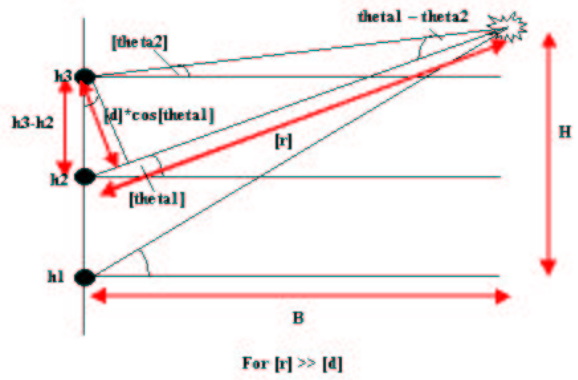



Figure 36: radius reconstruction schematic

The calculated distance was found using

$$sin\theta_1 = \frac{v\,(t_1 - t_2)}{\bar{h}_1} \tag{42}$$

$$r_2 = \frac{-\left(\frac{\bar{h}_2+\bar{h}_1}{2}\right)}{sin\theta_2\,\left(\frac{\cos\theta_1}{\cos\theta_2} - \frac{\sin\theta_1}{\sin\theta_2}\right)} \tag{43}$$

### 3.5.7 Rate calculation

**Code - rateCalc.C - appendix B.23** Evaluates equation 44 for the rate over a given energy range and then calculates the fraction of the number of events detected in that energy range from the results of the array studies and returns a rate for a given array.

**analysis** As discussed in section 1.7, the energy region of interest is the GZK region. The reason that this area is of particular interest is that any neutrinos detected in this energy range will be directly comparable with theory. The theory can also be used in this report to calculate a rate of neutrinos expected to be detected by the various array designs.

Fig 1 shows the flux prediction for GZK neutrinos.

For centre of mass energies above 10TeV there are no direct measurements from neutrino scattering or deep inelastic scattering experiments of the neutrino nucleon cross section. The cross section used in these calculations is an extrapolation of recent analysis that uses structure function measurements. The rate of interactions per $cm^3$ of water is given by:

$$R = 4\pi N \int_{E_\nu^{min}}^{E_\nu^{max}} \frac{d\phi}{dE_\nu}\sigma_{\nu N}\,(E)\,dE_\nu \tag{44}$$

N is the number of nucleons per $cm^3$ of water and full solid angle coverage is assumed. $\phi$ represents the GZK flux. Evaluating for $1.10^{17}$ - $1.10^{21}$

$$\mathrm{R} = 8.74.10^{-24}\ \mathrm{cm}^{-3}\mathrm{s}^{-1}$$

This translates per year for a 10km radius sphere to

$$R = 1165 \text{ yr}^{-1}$$

But as discussed earlier, at energies of this magnitude the Earth becomes opaque to the neutrinos. The calculations so far have assumed full solid angle coverage, and therefore the rates have to be divided by two. This gives a final predicted rate of

$$R = 550 \text{ yr}^{-1}$$

This rate assumes that every neutrino interaction is detected, but as previous results show, this is not true. Fig 33 shows that for a hydrophone cut of 0.025Pa only neutrinos with energy greater than $1.10^{20}$ were detected.

The rate for $1.10^{20}$eV - $1.10^{21}$eV is

$$R = 4.62 \text{ yr}^{-1}$$

Therefore by considering only events in this energy region, for a coincidence of four hydrophones and calculating the ratio of the number of events detected per number of events in this energy range a rate predicted rate can be found. The percentage of events detected are

Array 1 = 2.6%
Array 2 = 1.7%
Array 3 = 1.5%
Array 4 = 1.3%
Array 5 = 1.4%

For each array this translates to a rate per year of

**Array 1 = 0.12**
**Array 2 = 0.08**
**Array 3 = 0.07**
**Array 4 = 0.06**
**Array 5 = 0.06**

These rates are clearly too low to make a usable telescope from and work has to be done in bringing these numbers to useful rate. There are two main ways to increase the rate. The first is the hydrophone cut has to be decreased. Fig 37 shows the effect of decreasing the hydrophone cut to 0.0025Pa.

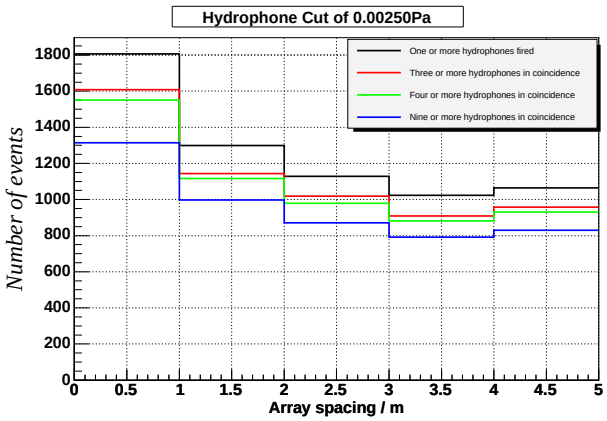



Figure 37: A hydrophone cut of 0.0025Pa

This translates to a rate of 0.52. This graph also shows that energies below $1.10^{20}$eV. The second important factor that must be considered is the effective area of the array. By increasing

the area of detection, most neutrinos are going to be detected. This has not been investigated further in this report, but factors that must be considered are the number of hydrophones that are to be used, clusters of hydrophones for reconstruction purposes, the optimum spacing of the hydrophones, and the actual practicality of having super size arrays.

### 3.5.8 Reconstruction of position, direction

To get the direction of the pulse is not a simple matter of geometry. The shower could be far away and have high energy or close and have a smaller energy. To infer the direction of the shower, the pancake shape of the acoustic pulse must be considered. From studying the position of the hydrophones that detected the pulse, and the values of the peak pressure, both the direction and the position can be calculated. This technique has not been investigated in this report.

For the reconstruction of the position consider: -

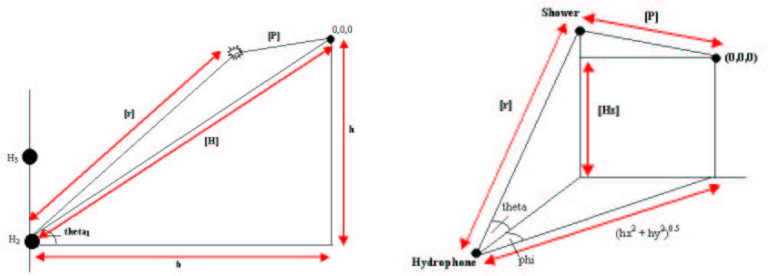



Figure 38: Position Reconstruction

Know from 3D trigonometry

$$r_1 = \left((x - h_1x)^2 + (y - h_1y)^2 + (z - h_1z)^2\right)^{\frac{1}{2}} \tag{45}$$

$$r_2 = \left((x - h_2x)^2 + (y - h_2y)^2 + (z - h_2z)^2\right)^{\frac{1}{2}} \tag{46}$$

$$r_3 = \left((x - h_3x)^2 + (y - h_3y)^2 + (z - h_3z)^2\right)^{\frac{1}{2}} \tag{47}$$

$$\tag{48}$$

Where r is the radius as described in §3.5.6

By expanding and rearranging

$$r^2 - \left(H_x^2 + H_x^2 + H_z^2\right) = \left(x^2 + y^2 + z^2\right) - 2\left(H_x x + H_y y + H_z z\right) \tag{49}$$

Where it can be seen that the left hand side of the equation is known and the right unknown. Assuming that $\left(x^2 + y^2 + z^2\right)$ can be calculated by considering fig 38 this can be written in the matrix form.

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} h_{1x} & h_{1y} & h_{1z} \\ h_{2x} & h_{2y} & h_{2z} \\ h_{3x} & h_{3y} & h_{3z} \end{pmatrix} (xyz) \tag{50}$$

where c is a constant and is defined by: -

$$c_1 = r^2 - \left(hx^2 + hy^2 + hz^2\right) - \left(x^2 + y^2 + z^2\right) \tag{51}$$

To calculate $\left(x^2+y^2+z^2\right)$ consider fig 38
z is simply

$$z = r_1 \sin\theta - h_z \tag{52}$$

but
$\left(x^2+y^2\right)$ is not so simple. Here $\phi$ must be considered. Using the cos rule

$$\left(x^2+y^2\right) = (r\cos\theta)^2 + \left(H_x^2+H_x^2\right) - 2r\cos\theta\left(H_x^2+H_x^2\right)^{0.5}\cos\phi \tag{53}$$

and hence

$$\left(x^2+y^2+z^2\right) = (r\cos\theta)^2 + \left(H_x^2+H_x^2\right) - 2r\cos\theta\left(H_x^2+H_x^2\right)^{0.5}\cos\phi + (r_1\sin\theta - h_z)^{0.5} \tag{54}$$

Which is hence

$$\left(x^2+y^2+z^2\right) = const1 - const2\cos\phi \tag{55}$$

But $\phi$ is unknown. To calculate $\phi$ with no uncertainty, 3 equations are needed, i.e 3 hydrophone positions and 3 r. Both r and the hydrophone position is known, and is hence an iterative method to pinpoint $\left(x^2+y^2\right)$. Knowing this x, y, and z can be found.

The question that this asks is what is the minimum number of hydrophones. In this report only hydrophones on the same string were able to be resolved to give a radius, but this does not mean that hydrophones on different strings cannot be used. To do thi different maths would have to be used, which was not investigated in this report. Therefore for the maths that exists in this report, 3 strings of 3 hydrophones would be required. This is large number of hydrophones, and is not really practical. Assuming that hydrophones on different strings can be resolved to give a radius, the minimum number of hydrophones to get 3 separate r would be 4 hydrophones. This is a much more sensible answer.

### 3.5.9 Problems with arrays in a moving body

All analysis done in section 3 has considered that the exact position of the hydrophones is known. In the ocean this is not true. The hydrophones will mot be attached to a solid rods, they will be attached to strings. The ocean is a moving body, and hence the strings will move with the ocean. This will add errors to the reconstruction techniques, but has not been investigated in this report.

### 3.5.10 Minimum frequency of analogue to digital converter

**code - rangauss.hh - B.24** A more realistic sampling rate is 200kHz, which corresponds to a time resolution of 5micro seconds. Figs 40 shows that both 1MHz and 200kHz reconstruct the radius very accurately. Larger time resolutions were tested until the reconstruction was no longer accurate.

**Macro - recontimescatter.C** This macro behave very much like recon.C. The only difference is in the code added to scatter the time. To do this a Gaussian function was used, as described above. The code gets the times as described in recon.C, scatters it according to the Gaussian and returns the scattered time. It is this time that is used in equation 42. The macro loops over various widths of the Gaussian.

**Analysis** From the results generated, an interesting question to ask is what is the minimum time resolution required to still be able to successfully reconstruct the radius. The description of the recon.C shows how the radius was reconstructed by considering the time difference between the hydrophones detecting a pulse. neutrinosim.cc generates 512 acoustic pulses in a time window of 0.0004s, which corresponds to a frequency of $1.28^6$ or a time resolution of $7.81^{-7}$s.

By using a Gaussian curve, the times generated in neutrinosim.cc were smeared. Plots of reconstructed vs actual were then remade.

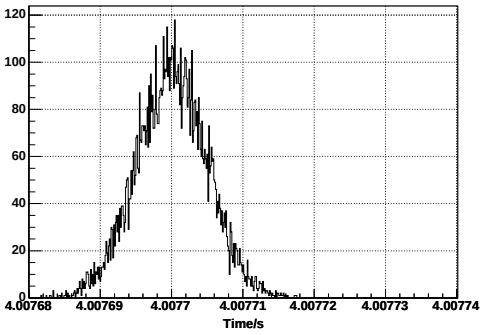
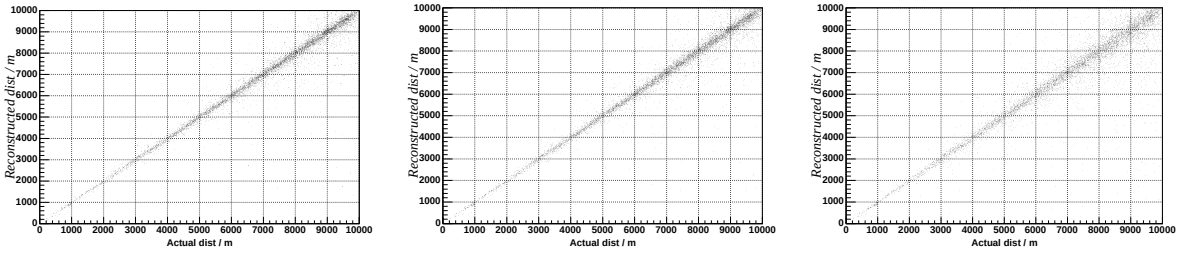
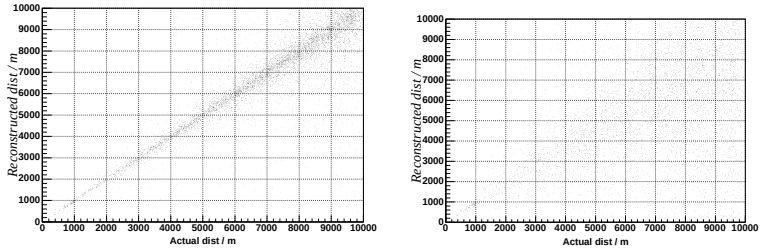



Figure 39: An example of the Gaussian curve that was used to smear the times




(a) no scatter (b) 5E-6s (c) 3E-5s




(d) 5E-5s (e) 5E-4s

Figure 40: time resolution

Figs 40 show the results for the varying time resolutions. The interpretation of the acceptable time resolution is arbitrary, but is suggested in this report that the acceptable time resolution is 30 microseconds and below.

There are two reasons why finding the minimum sampling frequency is interesting. The first is that the higher the frequency you want to sample at, the more expensive the equipment. The second reason is, the higher the frequency that you sample at, the more data there is to process, and hence computer time must be considered.

### 3.6 Summary of Array Studies

Using the acoustic pulse reconstruction that was demonstrated in section 1, arrays of detectors were investigated. Five arrays were studied. Coincidences of 3, 4, and 9 were investigated for varying hydrophone cuts. For low hydrophone cuts it was found that array1 was the most efficient. For higher cuts it was found that all arrays performed very similarly, and very poorly.

The energy range detected by each array was also investigated. It was found that only energies $>1.10^{20}$ were detected. Investigating the radius, it was found that by 10km no events were detected.

Using these studies rates were calculated for each array.

$$\textbf{Array 1} = \textbf{0.12}$$
$$\textbf{Array 2} = \textbf{0.08}$$
$$\textbf{Array 3} = \textbf{0.07}$$
$$\textbf{Array 4} = \textbf{0.06}$$
$$\textbf{Array 5} = \textbf{0.06}$$

For a practical telescope, these rates are too low. To increase these rates it is suggested that work must be done in lowering the hydrophone cut, and increasing the array effective area.

Reconstruction techniques were also investigated, and the radius successfully reconstructed, fig 40, a method on the reconstruction of the position was also suggested.

Using the reconstruction of the radius, the maximum time resolution was investigated. It is suggested that only resolutions of $<3.10^{-5}$ be used.

## 4 Conclusion

Using Geant4, hadronic showers were simulated for energies up to 100TeV. By analysing the structural form in r and z of these showers, the parameterised form the energy distribution was found to be

$$\frac{dE}{dr} = \left(e^{\frac{-(r-9.56813)}{24.6522}}\right) \tag{56}$$

$$\frac{de}{dz} = (x)^{9.00}\, exp\,(-1.75x) \tag{57}$$

Equations 56 were used to generate hadronic shower energy distributions for $1.10^{17} < \mathrm{E} < 1.10^{21}$, the GZK region. From these energy distributions, acoustic pulses were reconstructed.

Five hydrophone arrays were defined, and 50000 events simulated in a 10km radius sphere. By investigating the number of events detected at each hydrophone the efficiency of each array was investigated. It was found that a simple 2D array was the most efficient. Using the theoretical GZK flux, the 2D array is predicted to detect at a rate of 0.12 UHE neutrinos per year.

Much future work has to be completed in order to draw any sensible conclusions about the most efficient array design. All calculations in this report have assumed that the parameterised hadronic shower at 100TeV can be extrapolated to the higher GZK energies. Hadronic showers need to be simulated at these high energies and the parameterised equations tested. Also, at higher energies the neutrino cross section has not been experimentally confirmed, and hence the rate calculation may not be accurate. Many more array shapes need to be investigated,

including the spacing between the hydrophones. It was shown that events can be detected up to 10km away, and hence large effective volumes can be covered with few hydrophones.

It was shown that the rate can be increased by lowering the hydrophone cut, but this is dependant on studies in extracting the signal from the noise. Before these studies are completed the value can only be estimated. Reconstruction techniques have been suggested but also need further investigation.

For these studies, it was assumed that all of the energy of the neutrino went to the hadronic shower. This is of course not true. As fig 2 shows, there is also an electronic component to the shower. The cross section for both of these is about equal, and the two types of shower would be expected. For future simulations this need to be considered.

These early studies into detecting UHE neutrinos appear promising, and with more research a sensible detection rate should be achieved.

# References


[1] Griffis. *Introduction to Elementary Particles*, chapter 2. Wiley, 1987.

[2] J. G. Learned and Karl Mannheim. High energy neutrino astrophysics. *Annual Review Nuclear and particle Science*, 50:679–749, 2000.

[3] E. Waxman and J. Bachcall. High energy astrophysics. *Phys. Rev.*, D59, 1999.

[4] Mannheim, Protheroe, and Rachen. Waxman-bachall limit. *Ap. J*, 199, 2000, astro-ph/9812398.

[5] D. Waters. Acoustic detection of cosmic ray neutrinos.

[6] G. Patanchon, J. Delabrouille, and J.-F. Cardoso. Astrophysical data sets from the wmap satellite. *astro-ph*, 4, 2004, astro-ph/0403363.

[7] wwwasd.web.cern.ch/wwwasd/geant4/geant4.html.

[8] www.seas-upenn.edu/ chem101/sschem/metallusolids.html.

[9] Serway and Beichner. *Physics 5th Edition.* Wiley, 1995.

[10] www.globec.whoi.edu/globec-dir/sea_water_density_description.html.

[11] BR Martin and G Shaw. *Particle Physics*, chapter 4, page 169. Wiley, 1998.

[12] Bock Vasilescu. *The particle detector briefbook.* Bock, 1993.

[13] author. title. *Nuclear instruments and methods in physics research A*, pages 316–317, 1992.


# A Further Array Study Results

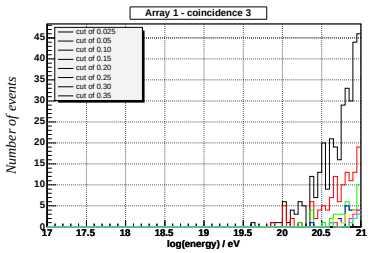
## A.1 Energy

## A.2 Radius

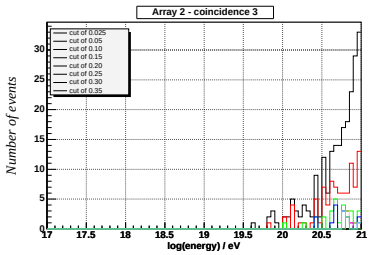
# B Code

## B.1 water_properties.hh

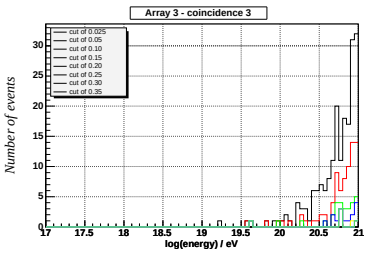
```
// ---------------------------------------- //
// Simon Bevan, 20/10/2003                  //
// C++ translation of MatLab code for calculating various properties of water  //
// Properties calculated here are: -        //
//                                          //
//      Pressure at given depth - double pressure(double depth)  //
```



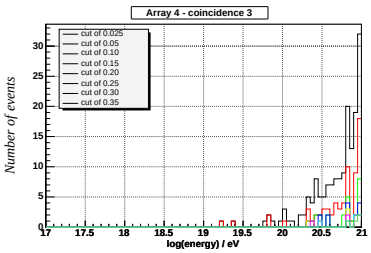

(a) array 1



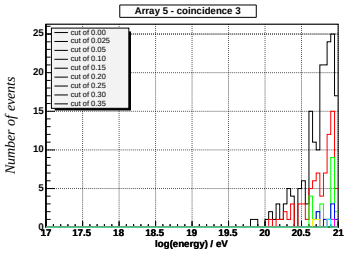

(b) array 3




(c) array 3

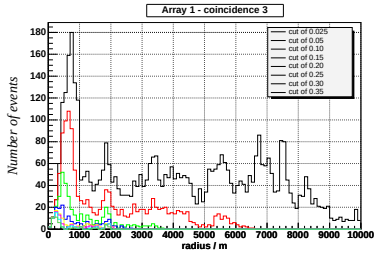



(d) array 4

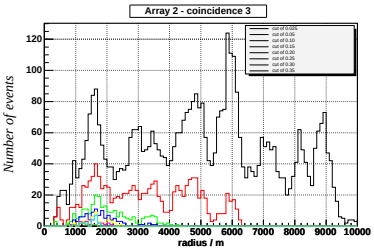



(e) array 5

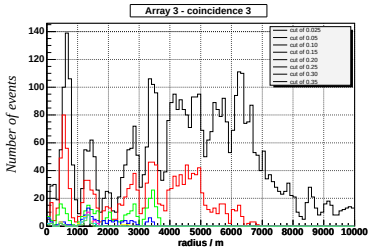
Figure 41: Efficiency as a function of energy for a coincidence of 4 hydrophones



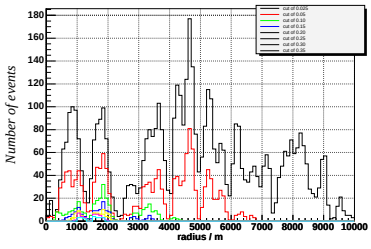

(a) array 1



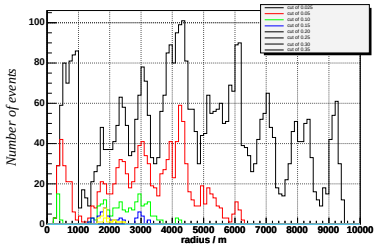

(b) array 3




(c) array 3




(d) array 4




(e) array 5

Figure 42: Efficiency as a function of energy for a coincidence of 4 hydrophones

```
//      Thermal Expensivity  -  double alpha(double temp, double depth, double salinity) //
//      Velocity             -  double vel(double temp, double depth, double salinity) //
//      Specific Heat        -  double cp(double temp, double depth, double salinity) //
// ------------------------------------------------------------------------------//

// calculation of pressure, in dB, at given depth,as needed for calculation velocity and thermal expansivity
// depth - meters, temp - degrees c, salinity - psu
// ==============================================

double pressuredepth(double depth) {

  double lat   = 60.0; //latitude hard wired at 60 degress, can be changed for specific locations
  double DEG2RAD = M_PI /180;
  double X     = sin((lat)*DEG2RAD);
  double C1    = 5.92E-3 + pow(X,2)*(5.25E-3);
  double pres  = ((1-C1)-pow(pow((1-C1),2)-8.84E-6*depth,0.5))/(4.42E-6);

  return pres;
}

//------------------------------------------------------------------------

// calculate adiabatic temp gradient, needed in calculations of thermal expansivity
// ==============================================================================

double sw_adtg(double salinity, double temp, double pres){

  //define constants
  //-------
  double a0 = 3.5803E-5;
  double a1 = +8.5258E-6;
  double a2 = -6.836E-8;
  double a3 = 6.6228E-10;

  double b0 = +1.8932E-6;
  double b1 = -4.2393E-8;

  double c0 = +1.8741E-8;
  double c1 = -6.7795E-10;
  double c2 = +8.733E-12;
  double c3 = -5.4481E-14;

  double d0 = -1.1351E-10;
  double d1 = 2.7759E-12;

  double e0 = -4.6206E-13;
  double e1 = +1.8676E-14;
  double e2 = -2.1687E-16;

  //calculate adiabatic temp gradient
  //-------------
  double ADTG = a0 + (a1 + (a2 + a3*temp)*temp)*temp +
    (b0 + b1*temp)*(salinity-35) +
    ((c0 + (c1 + (c2 + c3*temp)*temp)*temp) +
     (d0 + d1*temp)*(salinity-35))*pres +
    (e0 + (e1 + e2*temp)*temp )*pres*pres;
  return ADTG;
}
```

```
//————————————————————————————————

// calculation of theraml expansivity at given depth, temp, and salinity
// depth - meters, temp - degrees c, salinity - psu
// ========================================================================== 70

double alpha(double temp, double depth, double salinity) {

  //first calculate pressure
  double press = pressuredepth(depth);

  // have to turn temp into potential temperature
  //————————————————

  // theta1                                                                  80
  double del_P = -press;
  double del_th = del_P*sw_adtg(salinity,temp,press);
  double th   = temp + 0.5*del_th;
  double q    = del_th;

  // theta2
  del_th = del_P*sw_adtg(salinity,th,press+0.5*del_P);
  th     = th + (1 - 1/sqrt(2))*(del_th - q);
  q      = (2-sqrt(2))*del_th + (-2+3/sqrt(2))*q;
                                                                             90
  // theta3
  del_th = del_P*sw_adtg(salinity,th,press+0.5*del_P);
  th     = th + (1 + 1/sqrt(2))*(del_th - q);
  q      = (2 + sqrt(2))*del_th + (-2-3/sqrt(2))*q;

  // theta4
  del_th = del_P*sw_adtg(salinity,th,press+del_P);
  double PT  = th + (del_th - 2*q)/6;

  //calculate the thermal expansion saline contraction ratio adb           100
  //————————————————————————
  double c3 = -0.678662e-5;
  double c5 = 0.512857e-12;
  double c6 = -0.302285e-13;
  double sm35 = salinity-35;

  double AONB = -0.255019e-7*pow(PT,4)+ 0.298357e-5*pow(PT,3) + -0.203814e-3*pow(PT,2) + 0.170907e-1*PT +
    0.665157e-1 + sm35*( -0.846960e-4 +0.378110e-2 + -0.251520e-11*pow(press,2)+ -0.164759e-6*press) +
    pow(sm35,2)*c3 + press*(+0.791325e-8*pow(PT,2)+ -0.933746e-6 *PT +0.380374e-4) +
    c5*pow(press,2)*pow(PT,2) + c6*pow(press,3);                           110

  // Now calaculate the thermal expansion saline contraction ratio adb
  double cc4 = 0.515032e-8;
  double cc7 = 0.121551e-17;

  double BETA = -0.415613e-9*pow(PT,3)+ 0.555579e-7*pow(PT,2)+ -0.301985e-5*PT + 0.785567e-3 + sm35*( 0.788212e-

  double thermexpan = AONB*BETA;
  return thermexpan;
}                                                                          120

//————————————————————————-
```

```
// Calculate velocity of sound at given depth, temp, salinity
// temp - degrees C, salinity - PSU, pressure - kg/cm2
// Range of validity: temperature 0 to 30 řC, salinity 30 to 40
//parts per thousand, pressure 0 to 1000 kg/cm2, where 100 kPa=1.019716 kg/cm2.
// ============================================================================

double vel(double temp, double depth, double salinity) {

  // calculate the pressure in db at a given depth convert db to bars as used in UNESCO routines
  double presb = (pressuredepth(depth))/10;


  //constants
  //———
  double c00 = 1402.388; double c01 = 5.03711; double c02 = −5.80852e−2; double c03 = 3.3420e−4;
  double c04 = −1.47800e−6; double c05 = 3.1464e−9; double c10 = 0.153563; double c11 = 6.8982e−4;
  double c12 = −8.1788e−6; double c13 = 1.3621e−7; double c14 = −6.1185e−10; double c20 = 3.1260e−5;
  double c21 = −1.7107e−6; double c22 = 2.5974e−8; double c23 = −2.5335e−10; double c24 = 1.0405e−12;
  double c30 = −9.7729e−9; double c31 = 3.8504e−10; double c32 = −2.3643e−12; double a00 = 1.389;
  double a01 = −1.262e−2; double a02 = 7.164e−5; double a03 = 2.006e−6; double a04 = −3.21e−8;
  double a10 = 9.4742e−5; double a11 = −1.2580e−5; double a12 = −6.4885e−8; double a13 = 1.0507e−8;
  double a14 = −2.0122e−10; double a20 = −3.9064e−7; double a21 = 9.1041e−9; double a22 = −1.6002e−10;
  double a23 = 7.988e−12; double a30 = 1.100e−10; double a31 = 6.649e−12; double a32 = −3.389e−13;
  double b00 = −1.922e−2; double b01 = −4.42e−5; double b10 = 7.3637e−5; double b11 = 1.7945e−7;
  double d00 = 1.727e−3; double d10 = −7.9836e−6;


  double Cw =c00 + c01*temp + c02*pow(temp,2) + c03*pow(temp,3) + c04*pow(temp,4) + c05*pow(temp,5) +
    (c10 + c11*temp + c12*pow(temp,2) + c13*pow(temp,3) + c14*pow(temp,4))*presb +
    (c20 + c21*temp + c22*pow(temp,2) + c23*pow(temp,3) + c24*pow(temp,4))*pow(presb,2) +
    (c30 + c31*temp + c32*pow(temp,2))*pow(presb,3);

  double A =
    a00 +
    a01*temp + a02*pow(temp,2) + a03*pow(temp,3) + a04*pow(temp,4) +
    (a10 + a11*temp + a12*pow(temp,2) + a13*pow(temp,3) + a14*pow(temp,4))*presb +
    (a20 + a21*temp + a22*pow(temp,2) + a23*pow(temp,3))*pow(presb,2) +
    (a30 + a31*temp + a32*pow(temp,2))*pow(presb,3);

  double B = b00 + b01*temp + (b10 + b11*temp)*presb;

  double D = d00 + d10*presb;

  double svel = Cw + A*salinity + B*salinity*pow(salinity,0.5) + D*pow(salinity,2);

  return svel;
}

//————————————————————————————————————-

//Calculate specific heat:
//=======================

double cpf(double temp, double depth, double salinity){

  double presba = (pressuredepth(depth))/10;
```

```
double dec = 4217.4; double cpc1 = −3.720283; double cpc2 = 0.1412855;
double cpc3 = −2.654387e−3; double cpc4 = 2.093236e−5;

double cpa0a = −7.64357; double cpa1a = 0.1072763; double cpa2a = −1.38385e−3;

double cpb0a = 0.1770383; double cpb1a = −4.07718e−3; double cpb2a = 5.148e−5;

double cpa0 = −4.9592e−1; double cpa1 = 1.45747e−2; double cpa2 = −3.13885e−4;
double cpa3 = 2.0357e−6; double cpa4 = 1.7168e−8;                                    190

double cpb0 = 2.4931e−4; double cpb1 = −1.08645e−5; double cpb2 = 2.87533e−7;
double cpb3 = −4.0027e−9; double cpb4 = 2.2956e−11;

double cpc0a = −5.422e−8; double cpc1a = 2.6380e−9;
double cpc2a = −6.5637e−11; double cpc3a = 6.136e−13;

double cpd0 = 4.9247e−3; double cpd1 = −1.28315e−4; double cpd2 = 9.802e−7;
double cpd3 = 2.5941e−8; double cpd4 = −2.9179e−10;
                                                                                     200
double cpe0 = −1.2331e−4; double cpe1 = −1.517e−6; double cpe2 = 3.122e−8;

double cpf0 = −2.9558e−6; double cpf1 = 1.17054e−7;
double cpf2 = −2.3905e−9; double cpf3 = 1.8448e−11;

double cpg0 = 9.971e−8;

double cph0 = 5.540e−10; double cph1 = −1.7682e−11; double cph2 = 3.513e−13;

double cpj1 = −1.4300e−12;                                                           210

double S3_2 = salinity*pow(salinity,0.5);

//calculate tempory values
//−−−−−−−−−−
double Cpst0 = dec + cpc1*temp + cpc2*temp*temp + cpc3*pow(temp,3) + cpc4*pow(temp,4) +
  (cpa0a + cpa1a*temp + cpa2a*temp*temp)*salinity +
  (cpb0a + cpb1a*temp + cpb2a*temp*temp)*salinity*pow(salinity,0.5);

double del_Cp0t0 = (cpa0 + cpa1*temp + cpa2*temp*temp + cpa3*pow(temp,3) + cpa4*pow(temp,4))*presba +   220
  (cpb0 + cpb1*temp + cpb2*temp*temp + cpb3*pow(temp,3) + cpb4*pow(temp,4))*presba*presba +
  (cpc0a + cpc1a*temp + cpc2a*temp*temp + cpc3a*pow(temp,3))*pow(presba,3);

double del_Cpstp = ((cpd0 + cpd1*temp + cpd2*pow(temp,2) + cpd3*pow(temp,3) + cpd4*pow(temp,4))*salinity +
                    (cpe0 + cpe1*temp + cpe2*temp*temp)*S3_2)*presba +
  ((cpf0 + cpf1*temp + cpf2*temp*temp + cpf3*pow(temp,3))*salinity +
   cpg0* S3_2)*presba*presba +
  ((cph0 + cph1*temp + cph2*temp*temp)*salinity +
   cpj1*temp*S3_2)*pow(presba,3);
                                                                                     230
//calculate heat capacity
//−−−−−−−−−−
double heatcap = Cpst0 + del_Cp0t0 + del_Cpstp;
return heatcap;
    }
```

## B.2 randomPosition.hh

```
#include <math.h>

TVector3 randomPosition(TRandom3* ranGen) {

  // Generate a random position in a sphere.

  double rMax = 10500.0;
  double rMaxCube = pow(rMax,3.0);
                                                                          10
  // Random radius :
  double rCube = (ranGen->Rndm())*rMaxCube;
  double r = pow(rCube,(1.0/3.0));

  // Random phi :
  double phi = (ranGen->Rndm())*(2*M_PI);

  // Random theta :
  double cosTheta = (ranGen->Rndm())*2.0-1.0;
  double theta = acos(cosTheta);                                          20

  return TVector3(r*sin(theta)*cos(phi),
                  r*sin(theta)*sin(phi),
                  r*cos(theta));
}
```

## B.3 randomEnergy.hh

```
#include <math.h>
#include "TRandom3.h"

//double cs(double energy) {

//  // v-N cross section. From Terry Sloan and a ruler.
//  // log10(sigma/cm^2) = -32.60 + (0.362*(log10(E/GeV)-7))
//  // INPUT : energy in eV
//  // OUTPUT : cross section in cm^2                                       10
//  double egev = energy/1E09;
//  double log10cs = -32.60 + (0.362*(log10(egev)-7.0));
//  return pow(10,log10cs);
// }

  double randomEnergy(double& eMin, double& eMax, TRandom3* ranGen) {

   double E = 0.0;

   double logeMin = log10(eMin);                                          20
   double logeMax = log10(eMax);

double log10E = (ranGen->Rndm()*(logeMax - logeMin))+ logeMin;
```

```
//  // Generate a random energy according to the following distribution :
//  // dF/dE prop 1/E**2 convoluted with SM v-cross section.

//  // double eMin = 1E19;
//  // double eMax = 1E22;

//  double eMinInv = 1.0/eMin;
//  double eMaxInv = 1.0/eMax;

//  double csMin = cs(eMin);
//  double csMax = cs(eMax);

//  // cout << "eMin, csMin = " << eMin << " , " << csMin << endl;
//  // cout << "eMax, csMax = " << eMax << " , " << csMax << endl;

//  bool energyFixed = false;
//  while (!energyFixed) {
//    double invE = (ranGen->Rndm()*(eMinInv - eMaxInv))+eMaxInv;
//     E    = 1.0/invE;
//    // This is the energy according to 1/E**2 distribution.
//    // Now unweight according to the cross section dependence :
//    if ((ranGen->Rndm()*csMax) < cs(E)) {
//      energyFixed = true;
//    }
//    // cout << "E, energyFixed = " << E << " , " << energyFixed << endl;
//  }

 E = pow(10,log10E);

  return E;

}
```

---

## B.4 neutrinosim_Geant.cc

---

```
// ------------------------------------------ //
// Dave Waters, 7/6/2003                      //
// Edited by Simon Bevan, 4/11/2003           //
// Edited to incorporate water_properties.hh and geant simulation data //
//                                            //
// Edited by Simon Bevan, 4/12/2003           //
// Edited to incorporate test for shower shapes at varying z //

// C++ implementation of acoustic neutrino simulation //
// Originally impelemented in Octave.         //
//                                            //
// Need to set ROOTSYS and LD_LIBRARY_PATH *** gcc versions required *** //
// Compile :                                  //
// > g++ -I$ROOTSYS/include -c neutrinosim.cc -o neutrinosim.o //
// Link :                                     //
// > g++ -o neutrinosim neutrinosim.o -Llib -lCore -lCint -lHist -lGraf //
//     -lGraf3d -lTree -lMatrix -lRint -lm -ldl -L$ROOTSYS/lib //
```

```
//     -lpthread -rdynamic                                          //
// ------------------------------------------------------------- //
                                                                                20
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TFile.h"
#include "TNtuple.h"                                                            30
#include "TLeaf.h"
#include "TH1F.h"
#include "TH2F.h"
#include "TVector3.h"

//relic of old code, but needed for dp.hh, doesn't slow code down, so left alone so old code still works
enum {
  uhe = 1,
  dump = 2
};                                                                              40

#include "dp.hh"
#include "scaledVectorAdd.hh"
#include "water_properties.hh"
#include "scaledPulse.hh"

#define M_PI 3.14159265358979323846

using std::cout;
using std::vector;                                                              50


int main(int argc, char *argv[]) {


  int narg = argc-1;
  if (narg <10 ||narg >10) {
    cout << "usage : neutrinosim <num-events> <seed> <nint> <temp> <depth> <salinity> <angle_min> <angle_max> <.root i
    return 1;
  }                                                                             60

  //input variables
  //------
  int     num_events = atoi(argv[1]);
  int     seed       = atoi(argv[2]);
  int     nint       = atoi(argv[3]);
  double temp     = atof(argv[4]);
  double depth    = atof(argv[5]);
  double salinity  = atof(argv[6]);
                                                                                70

  // WATER PROPERTIES - from water_properties.hh :
  // ============================================

  // Temp, depth, salinity from input from neutrinosim
```

```
// coefficient of thermal expansion (bulk/volume expansivity) :
double beta = alpha(temp, depth, salinity);
// specific heat :
double cp  = cpf(temp, depth, salinity);                                    80
// attenuation coefficient :
double omega = 2.5E10;
// Speed of sound in water :
double c   = vel(temp, depth, salinity);


printf("\n neutrinosim : \n");
printf(" ============= \n");

                                                                            90
// Type of event to generate :
// ————————————
int atype;
//UHE neutrino cascades in water :
atype = uhe;
//atype = dump;


//get max and min angles from arguments
//————————————————-                                                         100
double angle_min = atof(argv[7]);
double angle_max = atof(argv[8]);
int     angle_bin = (int)((angle_max-angle_min)+1);

TH1F anglepres("pressure against angle","pressure against angle",angle_bin,(angle_min-0.5),(angle_max+0.5));


//extracting title.root and outputfile.root from input argument, but only if arguments entered
//————————————————————————————————————————
                                                                            110
char wavename[100];
sprintf(wavename,argv[9]);
char output[100];
sprintf(output, argv[10]);
char filename[100];
sprintf(filename,output);


// Make output file :
// ————————                                                                 120
TFile* histoFile = new TFile(filename,"RECREATE","neutrinosim geant histos");


//Find z of maximum energy (for shower centre)
//==========================================

// Read in GEANT shower information :
//————————————————-

TFile geantFile(wavename);                                                  130
geantFile.cd(); geantFile.ls();
TNtuple* geant = (TNtuple*)(geantFile.Get("ntuple"));
int nstep = (int)(geant->GetEntries());
```

```
//Book histogram finding z of max energy
//-----------------
TH1F dedz("dedz","dedz",100,0.0,20.0);

for (int i = 0; i < nstep; i++) {
  geant->GetEvent(i);

  double x0m  = geant->GetLeaf("x0")->GetValue()/1000.0;
  double y0m  = geant->GetLeaf("y0")->GetValue()/1000.0;
  double z0m  = geant->GetLeaf("z0")->GetValue()/1000.0;
  double x1m  = geant->GetLeaf("x1")->GetValue()/1000.0;
  double y1m  = geant->GetLeaf("y1")->GetValue()/1000.0;
  double z1m  = geant->GetLeaf("z1")->GetValue()/1000.0;
  double dem  = geant->GetLeaf("de")->GetValue()/1000.0;

  // Take point of energy deposition to be midway through the step :
  double x_depm = (x0m+x1m)/2.0;
  double y_depm = (y0m+y1m)/2.0;
  double z_depm = (z0m+z1m)/2.0;

  double r_depm = sqrt(pow(x_depm,2)+pow(y_depm,2));
  double phi_depm = atan2(x_depm,y_depm);

  dedz.Fill(z_depm,dem);

}

histoFile->cd(); dedz.Write();

//Extract the maximum z from histogram
//-----------------

double zmaxen = dedz.GetBinCenter(dedz.GetMaximumBin());
cout << zmaxen <<endl;


//=============================================================================


//Calculate acoustic pulses
//=======================

//variables for check on number of steps in slices
//-------------------

double totE = 0.0;
double z_count1 = 0.0;
double z_count2 = 0.0;
double z_count3 = 0.0;
double z_count4 = 0.0;
double z_count5 = 0.0;
double z_count6 = 0.0;
double z_count7 = 0.0;
double z_count8 = 0.0;
double z_count9 = 0.0;
```

```
double z_count10 = 0.0;
double z_count11 = 0.0;
double z_count12 = 0.0;
double z_count13 = 0.0;
double z_count14 = 0.0;
double z_count15 = 0.0;


//loop over all angles                                                    200
//--------

for (double angstep = angle_min; angstep <= angle_max; angstep+=1.0) {

  double angle = angstep;

  printf("\n Angle               : %f \n" ,angle);


  // Define hydrophone location :                                         210
  // ------------
  double radius = 1000.0;
  double x_sensor = radius*cos(M_PI*angle/180.0);
  double y_sensor = 0.0;
  double z_sensor = radius*sin(M_PI*angle/180.0);


  // Time-window for pressure pulse :
  // ---------------
  double start_time = (radius/c) - 0.0002;                                220
  double end_time = (radius/c) + 0.0002;
  int num_times    = 512;
  vector<double> times(num_times);

  for (int i = 0; i < num_times; ++i) {
    times[i] = start_time + (i*(end_time-start_time)/num_times);
  }


  // Define Integrated pressure pulse arrays :                            230
  // ------------------
  vector<double> pressure(num_times);
  vector<double> pressure1(num_times);
  vector<double> pressure2(num_times);
  vector<double> pressure3(num_times);
  vector<double> pressure4(num_times);
  vector<double> pressure5(num_times);
  vector<double> pressure6(num_times);
  vector<double> pressure7(num_times);
  vector<double> pressure8(num_times);                                    240
  vector<double> pressure9(num_times);
  vector<double> pressure10(num_times);
  vector<double> pressure11(num_times);
  vector<double> pressure12(num_times);
  vector<double> pressure13(num_times);
  vector<double> pressure14(num_times);
  vector<double> pressure15(num_times);

  vector<double> pinc(num_times);
```

```
                                                                                250

  //loop over all geant steps
  //----------.

  for (int istep = 0; istep < nstep; istep++) {
    geant->GetEvent(istep);


    if ((istep%1000)==0) cout << "istep = " << istep << endl;
                                                                                260
    // x0,y0,z0 : start of GEANT step (mm)
    // x1,y1,z1 : end of GEANT step (mm)
    // de       : energy deposited (MeV)

    double x0    = geant->GetLeaf("x0")->GetValue();
    double y0    = geant->GetLeaf("y0")->GetValue();
    double z0    = geant->GetLeaf("z0")->GetValue();
    double x1    = geant->GetLeaf("x1")->GetValue();
    double y1    = geant->GetLeaf("y1")->GetValue();
    double z1    = geant->GetLeaf("z1")->GetValue();                            270
    double de    = geant->GetLeaf("de")->GetValue();

    totE += de;

    // unit conversion (need distances in metres, energies in Joules) :
    x0 = x0/1000.0;
    y0 = y0/1000.0;
    z0 = z0/1000.0;
    x1 = x1/1000.0;
    y1 = y1/1000.0;                                                             280
    z1 = z1/1000.0;
    de = de*1.0E06*1.602E-19;

    //cout << "step length = " << sqrt(pow((x1-x0),2)+pow((y1-y0),2)+pow((z1-z0),2)) << endl;

    // assume centre of step is position of energy deposition.
    // GEANT shower starts at (x,y,z) = (0.0,0.0,zmaxen) metres

    double x_dep = (x0+x1)/2.0;
    double y_dep = (y0+y1)/2.0;                                                 290
    double z_dep = (z0+z1)/2.0 - zmaxen;


    // Calculate distance from measurement point :
    // ----------.
    double r = sqrt(pow((x_sensor - x_dep),2) + pow((y_sensor-y_dep),2) + pow((z_sensor-z_dep),2));


    // Find the pressure pulse :
    // ----------.                                                              300
    dp(&times,&pinc,r,uhe,temp,depth,salinity,beta,cp,omega,c);
    scaledVectorAdd(&pressure,&pinc,de);


//=============================================================================
```

```
//Further analysis, want to know what the pressure pulse is like at each step
//————————————————————————————————————————
                                                                                    310
//Calculating pressure pulse for each slice - where slice represents (z - zmax)
//————————————————————————————————————————

if(z_dep<=−5){
  z_count1 = z_count1 + 1;
  scaledVectorAdd(&pressure1,&pinc,de);
}

if((z_dep>−5)&&(z_dep<=−4)){
  z_count2 = z_count2 + 1;                                                          320
  scaledVectorAdd(&pressure2,&pinc,de);
}

if((z_dep>−4)&&(z_dep<=−3)){
  z_count3 = z_count3 + 1;
  scaledVectorAdd(&pressure3,&pinc,de);
}

if((z_dep>−3)&&(z_dep<=−2)){
  z_count4 = z_count4 + 1;                                                          330
  scaledVectorAdd(&pressure4,&pinc,de);
}

if((z_dep>−2)&&(z_dep<=−1)){
  z_count5 = z_count5 + 1;
  scaledVectorAdd(&pressure5,&pinc,de);
}

if((z_dep>−1)&&(z_dep<=0)){
  z_count6 = z_count6 + 1;                                                          340
  scaledVectorAdd(&pressure6,&pinc,de);
}

if((z_dep>0)&&(z_dep<=1)){
  z_count7 = z_count7 + 1;
  scaledVectorAdd(&pressure7,&pinc,de);
}

if((z_dep>1)&&(z_dep<=2)){
  z_count8 = z_count8 + 1;                                                          350
  scaledVectorAdd(&pressure8,&pinc,de);
}

if((z_dep>2)&&(z_dep<=3)){
  z_count9 = z_count9 + 1;
  scaledVectorAdd(&pressure9,&pinc,de);
}

if((z_dep>3)&&(z_dep<=4)){
  z_count10 = z_count10 + 1;                                                        360
  scaledVectorAdd(&pressure10,&pinc,de);
}

if((z_dep>4)&&(z_dep<=5)){
  z_count11 = z_count11 + 1;
```

```
    scaledVectorAdd(&pressure11,&pinc,de);
  }

  if((z_dep>5)&&(z_dep<=6)){
    z_count12 = z_count12 + 1;
    scaledVectorAdd(&pressure12,&pinc,de);
  }

  if((z_dep>6)&&(z_dep<=7)){
    z_count13 = z_count13 + 1;
    scaledVectorAdd(&pressure13,&pinc,de);
  }

  if((z_dep>7)&&(z_dep<=8)){
    z_count14 = z_count14 + 1;
    scaledVectorAdd(&pressure14,&pinc,de);
  }

  if(z_dep>8){
    z_count15 = z_count15 + 1;
    scaledVectorAdd(&pressure15,&pinc,de);
  }
}


//Displaying the number of steps at in each slice
//————————————————————-

cout << "totE = " << totE << endl;

cout << "total steps in shower " << nstep << endl;
cout <<""<<endl;
cout << "total steps in z < -5 " << z_count1 << endl;
cout << "total steps in -5 < z < -4 " << z_count2 << endl;
cout << "total steps in -4 < z < -3 " << z_count3 << endl;
cout << "total steps in -3 < z < -2 " << z_count4 << endl;
cout << "total steps in -2 < z < -1 " << z_count5 << endl;
cout << "total steps in -1 < z < 0 " << z_count6 << endl;
cout << "total steps in 0 < z < 1 " << z_count7 << endl;
cout << "total steps in 1 < z < 2 " << z_count8 << endl;
cout << "total steps in 2 < z < 3 " << z_count9 << endl;
cout << "total steps in 3 < z < 4 " << z_count10 << endl;
cout << "total steps in 4 < z < 5 " << z_count11 << endl;
cout << "total steps in 5 < z < 6 " << z_count12 << endl;
cout << "total steps in 6 < z < 7 " << z_count13 << endl;
cout << "total steps in 7 < z < 8 " << z_count14 << endl;
cout << "total steps in z > 8  " << z_count15 << endl;
cout<<""<<endl;

//Quick check to see if the number of steps are the same
//————————————————————————

cout << "do steps add up?      " << nstep -(z_count1 + z_count2 +z_count3 +z_count4 +z_count5 +z_count6 +z_count7

cout << "" << endl;

//Creating some titles fot the histogrms
//————————————————
```

```
char title[20];
sprintf(title,"angle %5.1f_total",angle);

char title1[20];
sprintf(title1,"angle %5.1f_1",angle);
char title2[20];
sprintf(title2,"angle %5.1f_2",angle);
char title3[20];
sprintf(title3,"angle %5.1f_3",angle);
char title4[20];
sprintf(title4,"angle %5.1f_4",angle);
char title5[20];
sprintf(title5,"angle %5.1f_5",angle);
char title6[20];
sprintf(title6,"angle %5.1f_6",angle);
char title7[20];
sprintf(title7,"angle %5.1f_7",angle);
char title8[20];
sprintf(title8,"angle %5.1f_8",angle);
char title9[20];
sprintf(title9,"angle %5.1f_9",angle);
char title10[20];
sprintf(title10,"angle %5.1f_10",angle);
char title11[20];
sprintf(title11,"angle %5.1f_11",angle);
char title12[20];
sprintf(title12,"angle %5.1f_12",angle);
char title13[20];
sprintf(title13,"angle %5.1f_13",angle);
char title14[20];
sprintf(title14,"angle %5.1f_14",angle);
char title15[20];
sprintf(title14,"angle %5.1f_15",angle);


// Store pressure pulse in a histogram :
// ====================================

//Signal from all z
//———————

TH1F signal(title,wavename,num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal.SetBinContent(i,pressure[i]);
}

//Signals from separate slice of z
//———————————————

TH1F signal1(title1,"slice1",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal1.SetBinContent(i,pressure1[i]);
}

TH1F signal2(title2,"slice2",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal2.SetBinContent(i,pressure2[i]);
```

```
}

TH1F signal3(title3,"slice3",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal3.SetBinContent(i,pressure3[i]);
}

TH1F signal4(title4,"slice4",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal4.SetBinContent(i,pressure4[i]);
}

TH1F signal5(title5,"slice5",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal5.SetBinContent(i,pressure5[i]);
}

TH1F signal6(title6,"slice6",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal6.SetBinContent(i,pressure6[i]);
}

TH1F signal7(title7,"slice7",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal7.SetBinContent(i,pressure7[i]);
}

TH1F signal8(title8,"slice8",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal8.SetBinContent(i,pressure8[i]);
}

TH1F signal9(title9,"slice9",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal9.SetBinContent(i,pressure9[i]);
}

TH1F signal10(title10,"slice10",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal10.SetBinContent(i,pressure10[i]);
}

TH1F signal11(title11,"slice11",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal11.SetBinContent(i,pressure11[i]);
}

TH1F signal12(title12,"slice12",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal12.SetBinContent(i,pressure12[i]);
}

TH1F signal13(title13,"slice13",num_times,start_time,end_time);
for (int i = 0; i < num_times; ++i) {
  signal13.SetBinContent(i,pressure13[i]);
}

TH1F signal14(title14,"slice14",num_times,start_time,end_time);
```

```
  for (int i = 0; i < num_times; ++i) {
    signal14.SetBinContent(i,pressure14[i]);
  }

  TH1F signal15(title15,"slice15",num_times,start_time,end_time);
  for (int i = 0; i < num_times; ++i) {
    signal15.SetBinContent(i,pressure15[i]);
  }


  //Write signals to file
  //————————

  histoFile->cd();
  signal.Write(); signal1.Write(); signal2.Write(); signal3.Write(); signal4.Write(); signal5.Write(); signal6.Write();signal7.Write();sig

  //Getting the maximum/minimum pressure, and time they occur, of siganls
  //————————————————————————————

  double maxpres = signal.GetBinContent(signal.GetMaximumBin());
  double minp = signal.GetBinContent(signal.GetMinimumBin());
  double maxpt = times[signal.GetMaximumBin()];
  double minpt = times[signal.GetMinimumBin()];

  // filling  angle against max pressure histogram
  //————————————————-
  anglepres.Fill(angle,maxpres);


  //print to screen
  //——————

  printf("\n Maximum pressure  : %f \n",maxpres);
  printf(" Minimum pressure  : %f \n",minp);
  printf("\n Maximum pressure time : %f \n",maxpt);
  printf(" Minimum pressure time : %f \n",minpt);

}
histoFile->cd(); anglepres.Write();
histoFile->Close();

return 0;
    }
```

## B.5 geant_ntuple_anal2.C

```
//=====================================================//
//created by Dave Waters                               //
//modified by Simon Bevan, 17/10/03                    //
//code to display histograms from ntuple created by    //
//geant                                                //
//modified to allow two 2D histograms to be opened     //
//and compared                                         //
//=====================================================//
```

```
#include "newslide.h"

void geant_ntuple_anal2(){

  Float_t M_PI=3.1415927;

  //=========================================================================

  //Root code
  //———:
  // Clean up :
  // ———-
  gROOT->Reset();
  gROOT->SetStyle("Plain");

  // This gives nice purple->red color scale for 2-D histogram plotting
  gStyle->SetPalette(1);

  // No stats :
  // ———-
  // Reasonable looking fonts :
  // ——————————
  gStyle->SetLabelFont(12,"x");
  gStyle->SetLabelFont(12,"y");
  gStyle->SetTitleFont(12);
  gStyle->SetTitleSize(0.06);

  char pad_name[120];
  c1 = NewSlide("plots","plots",1,1);
  c1->SetFillColor(0);

  // c1->SetLogy();

  sprintf(pad_name,"%s_p1","acoustic_plots");
  TPad* p1 = (TPad*) c1->GetPrimitive(pad_name);

  //=========================================================================

  //Create chain for 100TeV files
  //——————————
  TChain chain1("ntuple");

  chain1.Add("100TeV_2/shower_data_00.root");
  chain1.Add("100TeV_2/shower_data_01.root");
  chain1.Add("100TeV_2/shower_data_02.root");
  chain1.Add("100TeV_2/shower_data_03.root");
  chain1.Add("100TeV_2/shower_data_04.root");
  chain1.Add("100TeV_2/shower_data_05.root");

  // Open input file :
  // ———————
  //  TFile neutrino("20TeV_pion_seawater_3.root");
  //   neutrino.cd();
  //   neutrino.ls();
  //   TNtuple* geant = (TNtuple*) gROOT.FindObject("ntuple");

  //if using virtual files coment above and use below
  //——————————————————————
```

```
TTree* geant;
geant = (TTree*)(&chain1);
                                                                                70
// Loop through ntuple, filling arrays :
//-----------------
Int_t n = geant->GetEntries();
cout << "Ntuple has " << n << " entries" << endl;

// Book histograms to fill :
// ----------
TH1F* dedr   = new TH1F("dedr","dedr",1000,0.0,1000.0); dedr->SetDirectory(0);
TH1F* dedz   = new TH1F("dedz","dedz",100,0.0,10.0); dedz->SetDirectory(0);
TH1F* dedphi = new TH1F("dedphi","dedphi",100,-1.0*M_PI,1.0*M_PI); dedphi->SetDirectory(0); 80
TH2F* rz     = new TH2F("rz","rz",200,-2000.0,2000.0,100,0.0,3000.0); rz->SetDirectory(0);
TH1F* StepLength = new TH1F("StepLength","StepLength",10000,0.0,0.05); StepLength->SetDirectory(0);

Float_t totE = 0.0;
Float_t totstepleng = 0.0;

//loop through events
//--------
for (Int_t i = 0; i < n; i++) {
  geant->GetEvent(i);                                                           90

  // x0,y0,z0 : start of GEANT step (mm)
  // x1,y1,z1 : end of GEANT step (mm)
  // de       : energy deposited (MeV)

  Float_t x0     = geant->GetLeaf("x0")->GetValue();
  Float_t y0     = geant->GetLeaf("y0")->GetValue();
  Float_t z0     = geant->GetLeaf("z0")->GetValue();
  Float_t x1     = geant->GetLeaf("x1")->GetValue();
  Float_t y1     = geant->GetLeaf("y1")->GetValue();                            100
  Float_t z1     = geant->GetLeaf("z1")->GetValue();
  Float_t de     = geant->GetLeaf("de")->GetValue();

  // Check of total energy deposition :
  totE += de;

  // Take point of energy deposition to be midway through the step :
  Float_t x_dep = (x0+x1)/2.0;
  Float_t y_dep = (y0+y1)/2.0;
  Float_t z_dep = (z0+z1)/2.0;                                                  110

  Float_t r_dep = sqrt(pow(x_dep,2)+pow(y_dep,2));

  Float_t phi_dep = atan2(x_dep,y_dep);

  dedr->Fill(r_dep,de);
  dedz->Fill(z_dep/1000.0,de);
  dedphi->Fill(phi_dep,de);

  if (phi_dep>0.0) {                                                            120
    rz->Fill(r_dep/10.0,z_dep/10.0,de);
  } else {
    rz->Fill(-1.0*r_dep/10.0,z_dep/10.0,de);
  }
```

```
    Float_t steplength = sqrt(pow((x0-x1),2)+pow((y0-y1),2)+pow((z0-z1),2));
    totstepleng += steplength;
    StepLength->Fill(steplength);
    Float_t avstepleng = totstepleng/n;
  };                                                                          130

  //Show result
  //----
  //Making Histograms look nice
  p1->cd(1);
  gPad->SetGrid();

  //rz graph
  //========
  rz->SetTitle("");                                                           140
  rz->GetXaxis()->SetTitle("r (cm)");
  rz->GetXaxis()->CenterTitle();
  rz->GetXaxis()->SetTitleFont(12);
  rz->GetXaxis()->SetTitleSize(0.04);
  rz->GetYaxis()->SetTitle("z (cm)");
  rz->GetYaxis()->CenterTitle();
  rz->GetYaxis()->SetTitleFont(12);
  rz->GetYaxis()->SetTitleOffset(1.5);
  rz->GetYaxis()->SetTitleSize(0.04);
  //rz->Draw("surf");                                                         150
  rz->Draw("colz");
  TArrow* ta = new TArrow();
  ta->SetLineColor(2);
  ta->SetLineWidth(3);
  ta->DrawArrow(0.0,25.0,0.0,100.0,0.02);

}


                                                                              160
```

## B.6 dedzfitter.C

```
//=============================================================
//Created by Simon Bevan
//03/12/03

//Root macro for creating dedr,dedz, and dedphi graphs
//Also fits dedz, gets parameters and plots them aginst logE
//=============================================================

#include <stdio.h>
#include <iostream>                                                           10
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
```

```
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"

//Code for excluding points below a given x value
//-------------------
Double_t fitf(Double_t* xx, Double_t *par){

  Double_t x = xx[0];

  if(x<1){
    TF1::RejectPoint();
    return 0;
  }

  //the fitting function
  //--------
  Double_t fitval = par[0]*pow(fabs(x-1.0-par[3]),par[1])*exp(par[2]*(fabs(x-1.0-par[3])));
  return fitval;
}

void dedzfitter(){

  // make some virtual files for the 100TeV files
  //-------------------
  TChain chain1("ntuple");
  TChain chain2("ntuple");
  TChain chain3("ntuple");
  TChain chain4("ntuple");
  TChain chain5("ntuple");

  //Add files to chains
  //-------
  chain1.Add("100TeV_1/shower_data_00.root");
  chain1.Add("100TeV_1/shower_data_01.root");
  chain1.Add("100TeV_1/shower_data_02.root");
  chain1.Add("100TeV_1/shower_data_03.root");
  chain1.Add("100TeV_1/shower_data_04.root");
  chain1.Add("100TeV_1/shower_data_05.root");

  chain2.Add("100TeV_2/shower_data_00.root");
  chain2.Add("100TeV_2/shower_data_01.root");
  chain2.Add("100TeV_2/shower_data_02.root");
  chain2.Add("100TeV_2/shower_data_03.root");
  chain2.Add("100TeV_2/shower_data_04.root");
  chain2.Add("100TeV_2/shower_data_05.root");

  chain3.Add("100TeV_3/shower_data_00.root");
  chain3.Add("100TeV_3/shower_data_01.root");
  chain3.Add("100TeV_3/shower_data_02.root");
  chain3.Add("100TeV_3/shower_data_03.root");
  chain3.Add("100TeV_3/shower_data_04.root");
  chain3.Add("100TeV_3/shower_data_05.root");

  chain4.Add("100TeV_4/shower_data_00.root");
  chain4.Add("100TeV_4/shower_data_01.root");
```

```
chain4.Add("100TeV_4/shower_data_02.root");
chain4.Add("100TeV_4/shower_data_03.root");
chain4.Add("100TeV_4/shower_data_04.root");
chain4.Add("100TeV_4/shower_data_05.root");

chain5.Add("100TeV_5/shower_data_00.root");
chain5.Add("100TeV_5/shower_data_01.root");
chain5.Add("100TeV_5/shower_data_02.root");
chain5.Add("100TeV_5/shower_data_03.root");
chain5.Add("100TeV_5/shower_data_04.root");
chain5.Add("100TeV_5/shower_data_05.root");

//Make log file more presentable
//————————————
std::cout << "" <<std::endl;
std::cout << "              LOG FILE FOR GRAPHFITTER() " <<std::endl;
std::cout << "" << std::endl;

// Clean up :
// ———-
gROOT->Reset();
gROOT->SetStyle("Plain");

// No stats :
// ———-
gStyle->SetOptStat(0);

// Reasonable looking fonts :
// ————————————
gStyle->SetLabelFont(12,"x");
gStyle->SetLabelFont(12,"y");
gStyle->SetTitleFont(12);
gStyle->SetTitleSize(0.06);

//Define some variables for later
//————————————————
Double_t p0 = 0;
Double_t p1 = 0;
Double_t p2 = 0;
Double_t p3 = 0;
Double_t p0avtemp = 0;
Double_t p1avtemp = 0;
Double_t p2avtemp = 0;
Double_t p3avtemp = 0;
Double_t zmaxen = 0;
Double_t maxheight = 0;
Double_t halfheightwidth = 0;
Double_t avmaxheighttemp = 0;
Double_t avzmaxentemp = 0;
Double_t avhalfheightwidthtemp = 0;

//Create some histograms for average graphs
//——————————————————-
TH1F energyp0("Energy_vs_Average_p0","Energy_vs_Average_p0", 1500, 10.0, 15.0);
TH1F energyp1("Energy_vs_Average_p1","Energy_vs_Average_p1", 1500, 10.0, 15.0);
TH1F energyp2("Energy_vs_Average_p2","Energy vs Average p2", 1500, 10.0, 15.0);
TH1F energymaxheight("Energy_vs_maxheight","Energy_vs_maxheight", 1500, 10.0, 15.0);
TH1F energyzmaxen("Energy_vs_zmaxen","Energy_vs_zmaxen", 1500, 10.0, 15.0);
```

```
TH1F energyhhw("Energy_vs_hhw","Energy_vs_hhw", 1500, 10.0, 15.0);

// Make outpout file for average graphs
// ----------------
TFile* averagehistoFile = new TFile("average_graphs.root","RECREATE","neutrinosim geant histos");


// Loop through all of the files
// ============================
// Loop though each energy
// ---------

for (Int_t d = 0; d <= 8 ; d++){

  // Make output files :
  // -------

  Int_t energy = 0;
  Int_t jmax = 0;

  if(d == 0){
    energy = 1;
    jmax = 9;
  }
  else if(d == 1){
    energy = 2;
    jmax = 13;
  }
  else if(d == 2){
    energy = 3;
    jmax = 6;
  }
  else if(d == 3){
    energy = 5;
    jmax = 11;
  }
  else if(d == 4){
    energy = 7;
    jmax = 4;
  }
  else if(d == 5){
    energy = 10;
    jmax = 8;
  }
  else if(d == 6){
    energy = 15;
    jmax = 5;
  }
  else if(d == 7){
    energy = 20;
    jmax = 7;
  }
  else if(d == 8){
    energy = 100;
    jmax = 5;
  }

  //Create some Histograms for varition of parameters within each energy
```

```
// ————————————————————————
Char_t filenamep0p1[100];
std::sprintf(filenamep0p1, "%dTeV_pion_seawater_p0p1",energy);

Char_t filenamep1p2[100];
std::sprintf(filenamep1p2, "%dTeV_pion_seawater_p1p2",energy);

Char_t filenamep0p2[100];
std::sprintf(filenamep0p2, "%dTeV_pion_seawater_p0p2",energy);

Char_t filenameparcomp[100];
std::sprintf(filenameparcomp, "%dTeV_pion_seawater_parcomp.root",energy);

TH1F p0p1(filenamep0p1,filenamep0p1,1500,0.0,250000);
TH1F p1p2(filenamep1p2,filenamep1p2,1500,0.0,150.0);
TH1F p0p2(filenamep0p2,filenamep0p2,1500,-25.0,0.0);

//Create file for parameter comparisons
//————————————————
TFile* parcomphistoFile = new TFile(filenameparcomp,"RECREATE","neutrinosim geant histos_1");


//loop hrough files in each energy
//————————————-
p0avtemp = 0;
p1avtemp = 0;
p2avtemp = 0;
p3avtemp = 0;
avmaxheighttemp = 0;
avzmaxentemp = 0;
avhalfheightwidthtemp = 0;

for (Int_t j = 1; j <= jmax; j++) {

  //reset values for loop
  p0 = 0;
  p1 = 0;
  p2 = 0;
  p3 = 0;

  zmaxen = 0;
  maxheight = 0;
  halfheightwidth = 0;

  //make a filenames that are appropiate from the loop
  //————————————————————
  std::cout << "" <<std::endl;

  Char_t filename[100];
  std::sprintf(filename, "%dTeV_pion_seawater_%d.root",energy,j);

  Char_t filenameout[100];
  std::sprintf(filenameout, "%dTeV_pion_seawater_out_%d.root",energy,j);

  Char_t filenamededr[100];
  std::sprintf(filenamededr, "%dTeV_pion_seawater_%d_dedr",energy,j);

  Char_t filenamededz[100];
```

```
std::sprintf(filenamededz, "%dTeV_pion_seawater_%d_dedz",energy,j);

Char_t filenamededphi[100];
std::sprintf(filenamededphi, "%dTeV_pion_seawater_%d_dedphi",energy,j);
                                                                                    250
// Make output files :
// ---------
TFile* histoFile = new TFile(filenameout,"RECREATE","neutrinosim geant histos2");

//Book some hisograms to fill
//----------
TH1F dedr(filenamededr,"dedr",1000,0.0,1000.0);
TH1F dedz(filenamededz,"dedz",250,0.0,20.0);
TH1F dephi(filenamededphi,"dedphi",100,-1.0*M_PI,1.0*M_PI);
                                                                                    260
// Open input file :
// -------

TTree* geant;

TFile* neutrino;

if (energy <=20 ){

  neutrino = new TFile(filename);                                                   270

  neutrino->cd();
  neutrino->ls();

  geant = (TTree*)(neutrino->Get("ntuple"));
}

//for 100TeV need to open virtual files
//----------
                                                                                    280
if (energy > 20){

  if(j==1){
    geant = (TTree*)(&chain1);
  }
  if(j==2){
    geant = (TTree*)(&chain2);
  }
  if(j==3){
    geant = (TTree*)(&chain3);                                                      290
  }
  if(j==4){
    geant = (TTree*)(&chain4);
  }
  if(j==5){
    geant = (TTree*)(&chain5);
  }
}

// Loop through ntuple, filling arrays :                                          300
Int_t n = (Int_t)(geant->GetEntries());

std::cout << "Ntuple has " << n << " entries" << std::endl;
```

```
Float_t totE = 0.0;
Float_t totstepleng = 0.0;

for (Int_t i = 0; i < n; i++) {
  geant->GetEvent(i);
                                                                          310
  // x0,y0,z0 : start of GEANT step (mm)
  // x1,y1,z1 : end of GEANT step (mm)
  // de       : energy deposited (MeV)

  Float_t x0     = geant->GetLeaf("x0")->GetValue();
  Float_t y0     = geant->GetLeaf("y0")->GetValue();
  Float_t z0     = geant->GetLeaf("z0")->GetValue();
  Float_t x1     = geant->GetLeaf("x1")->GetValue();
  Float_t y1     = geant->GetLeaf("y1")->GetValue();
  Float_t z1     = geant->GetLeaf("z1")->GetValue();                      320
  Float_t de     = geant->GetLeaf("de")->GetValue();

  // Check of total energy deposition :
  totE += de;

  // Take point of energy deposition to be midway through the step :
  Float_t x_dep = (x0+x1)/2.0;
  Float_t y_dep = (y0+y1)/2.0;
  Float_t z_dep = (z0+z1)/2.0;
                                                                          330
  Float_t r_dep = sqrt(pow(x_dep,2)+pow(y_dep,2));

  Float_t phi_dep = atan2(x_dep,y_dep);

  // Store results in a histogram :
  // ----------

  Char_t titlededr[100];
  std::sprintf(titlededr,"%s_dedr ",filename);
                                                                          340
  Char_t titlededz[100];
  std::sprintf(titlededz,"%s_dedz ",filename);

  Char_t titledephi[100];
  std::sprintf(titledephi,"%s_dedphi ",filename);

  dedr.Fill(r_dep,de);
  dedz.Fill(z_dep/1000.0,de);
  dephi.Fill(phi_dep,de);
                                                                          350
  Float_t steplength = sqrt(pow((x0-x1),2)+pow((y0-y1),2)+pow((z0-z1),2));
  totstepleng += steplength;
  Float_t avstepleng = totstepleng/n;
}

//add errors to bins
for (Int_t w=0; w<= dedz.GetNbinsX(); ++w) {
  dedz.SetBinError(w,750.0);
}
                                                                          360
//Defining the fit function
```

```
  //————————————-

  TF1 *func = new TF1("swbfit", fitf, 1.0,25.0,4);

  func->SetRange(1.0,25.0);
  func->SetParameters(10000.0,1.0,-1.0);

  dedz.Fit("swbfit");
                                                                                    370
  TF1 *fit = dedz.GetFunction("swbfit");

  histoFile->cd(); dedz.Write(); dedr.Write(); dephi.Write(); fit->Write();

  // Analyse the fit
  // ———————
  zmaxen = fit->GetMaximumX(1.0,17.0);
  p0     = fit->GetParameter(0);
  p1     = fit->GetParameter(1);
  p2     = fit->GetParameter(2);                                                    380
  p3     = fit->GetParameter(3);

  maxheight = p0*pow(fabs(zmaxen-1.0-p3),p1)*exp(p2*(fabs(zmaxen-1.0-p3)));

  halfheightwidth = (fit->GetX((maxheight/2),zmaxen,17.0)) - (fit->GetX((maxheight/2),zmaxen,1.0));

  //Get Averages
  //—————
  avmaxheighttemp = avmaxheighttemp + maxheight;
  avzmaxentemp = avzmaxentemp + zmaxen;                                             390
  avhalfheightwidthtemp = avhalfheightwidthtemp + halfheightwidth;

  p0avtemp = p0avtemp + p0;
  p1avtemp = p1avtemp + p1;
  p2avtemp = p2avtemp + p2;
  p3avtemp = p3avtemp + p3;

  p0p1.Fill(p0,p1);
  p1p2.Fill(p1,p2);
  p0p2.Fill(p0,p2);                                                                 400

  //Print everything to screen
  //————————————-

  std::cout << "" <<std::endl;
  std::cout << "" <<std::endl;

  std::cout << "The total energy in this file is " << totE << "MeV" << std::endl;
  std::cout << "the maximum height is         " << maxheight << std::endl;
  std::cout << "the z of the maximum height is " << zmaxen << std::endl;          410
  std::cout << "the half height width is      " << halfheightwidth << std::endl;

  std::cout << "" <<std::endl;
  std::cout << "-------------------------------------------------------" <<std::endl;
  std::cout << "" <<std::endl;

  histoFile->Close();

}
```

```
  std::cout << "-------------------------------------------------------" << std::endl;
  std::cout <<""<<std::endl;
  std::cout << "             " << energy << "TeV analysis"<<std::endl;
  std::cout <<""<<std::endl;

  //calculate averages
  //———————
  Double_t p0av = p0avtemp/jmax;
  Double_t p1av = p1avtemp/jmax;
  Double_t p2av = p2avtemp/jmax;
  Double_t avmaxheight = avmaxheighttemp/jmax;
  Double_t avzmaxen = avzmaxentemp/jmax;
  Double_t avhalfheightwidth = avhalfheightwidthtemp/jmax;

  std::cout << "The average value for p0 for " << energy << "TeV is " << p0av << std::endl;
  std::cout << "The average value for p1 for " << energy << "TeV is " << p1av << std::endl;
  std::cout << "The average value for p2 for " << energy << "TeV is " << p2av << std::endl;
  std::cout << "The average maxheight for " << energy << "TeV is " << avmaxheight << std::endl;
  std::cout << "The average zmaxen for   " << energy << "TeV is " << avzmaxen << std::endl;
  std::cout << "The average halfheightwidth for " << energy << "TeV is " << avhalfheightwidth << std::endl;

  parcomphistoFile->cd(); p0p1.Write(); p1p2.Write(); p0p2.Write();

  Double_t logenergy = log10(energy*pow(10,12));

  energyp0.Fill(logenergy,p0av);
  energyp1.Fill(logenergy,p1av);
  energyp2.Fill(logenergy,p2av);

  energymaxheight.Fill(logenergy, avmaxheight);
  energyzmaxen.Fill(logenergy, avzmaxen);
  energyhhw.Fill(logenergy, avhalfheightwidth);

  parcomphistoFile->Close();

  std::cout << "" <<std::endl;
  std::cout << "=======================================================" <<std::endl;
  std::cout << "" <<std::endl;

}

averagehistoFile->cd(); energyp0.Write(); energyp1.Write(); energyp2.Write(); energymaxheight.Write(); energyzmaxen.Write(); en
averagehistoFile->Close();

    }
```

## B.7 dedzcomprevised.C

```
//=========================================================//
//Created by Simon Bevan                                   //
//29/11/03                                                 //
//                                                         //
//Revised 12/01/04 to tidy code                            //
//                                                         //
//Root macro for displaying dedz                           //
```

```
//from output of graphfitter()                    //
//=========================================================//
                                                                    10
#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"                                                 20
#include "TCanvas.h"


//fit function for dedz
//————————

Double_t fitf(Double_t* xx, Double_t *par){
  Double_t x = xx[0];

  //excludes ponts for x<2                                          30
  //————————
  if(x<1){
    TF1::RejectPoint();
    return 0;
  }

  //the fit function
  //———————-
  Double_t fitval = par[0]*pow(fabs(x-1.0-par[3]),par[1])*exp(par[2]*(fabs(x-1.0-par[3])));
                                                                    40
  return fitval;
}

  void dedzcomprevised() {

//=========================================================

//Set up for root
//===============
                                                                    50
// Clean up :
// ==========
gROOT->Reset();

// No stats :
// ==========
gStyle->SetOptStat(0);
gROOT->SetStyle("Plain");

c1 = new TCanvas("c1","neutrinosim",200,10,700,500);                60

c1->SetFillColor(0);
c1->SetGridx();
c1->SetGridy();
```

```
//=============================================================

//Loop through all files creating histograms
//-----------------
                                                                    70
TFile *pulse[26];
TH1F *histo[26];
TF1 *funcfit[26];
TF1 *fit[26];

for(int i = 1; i<=20 ; i++){//

  Int_t energy = 0;
  Int_t j = 0;
  Int_t pulsenum = 0;                                               80
  Int_t histonum = 0;

  if((i>=1)&&(i<=5)){
    energy = 1;
    j=i;
  }

  if((i>=6)&&(i<=10)){
    energy = 5;
    j = i-5;                                                        90
  }

  if((i>=11)&&(i<=15)){
    energy = 10;
    j = i-10;
  }

  if((i>=16)&&(i<=20)){
    energy = 20;
        j = i-15;                                                   100
  }

  Char_t filename[100];
  std::sprintf(filename, "%dTeV_pion_seawater_out_%d.root",energy,j);
  Char_t filenamekey[100];
  std::sprintf(filenamekey, "%dTeV_pion_seawater_%d_dedz",energy,j);

  pulse[i] = new TFile(filename);
  histo[i] = (TH1F*) pulse[i]->Get(filenamekey);
                                                                    110
}

//100TeV have different key name, so have to make implicitly
//-------------------------
pulse[21] = new TFile("100TeV_pion_seawater_out_1.root");
histo[21] = (TH1F*) pulse[21]->Get("100TeV_pion_seawater_1_dedz");
pulse[22] = new TFile("100TeV_pion_seawater_out_2.root");
histo[22] = (TH1F*) pulse[22]->Get("100TeV_pion_seawater_2_dedz");
pulse[23] = new TFile("100TeV_pion_seawater_out_3.root");
histo[23] = (TH1F*) pulse[23]->Get("100TeV_pion_seawater_3_dedz");  120
pulse[24] = new TFile("100TeV_pion_seawater_out_4.root");
histo[24] = (TH1F*) pulse[24]->Get("100TeV_pion_seawater_4_dedz");
pulse[25] = new TFile("100TeV_pion_seawater_out_5.root");
```

```
histo[25] = (TH1F*) pulse[25]->Get("100TeV_pion_seawater_5_dedz");

for(int i = 1; i<=25; i++){

                //adding errors to the graphs
                //------------
                for (Int_t w=0; w<= histo[i]->GetNbinsX(); w++) {
                  histo[i]->SetBinError(w,500.0);
               }

                //fitting the function
                //---------
                Char_t filenamefit[100];
                std::sprintf(filenamefit, "pion_seawater_%d_dedz_fit",i);
                funcfit[i] = new TF1(filenamefit, fitf, 1.0, 25.0, 4);
                funcfit[i]->SetParameters(10000.0,1.0,-1.0);

                histo[i].Fit(filenamefit);
                fit[i] = histo[i].GetFunction(filenamefit);

                //set line styles for all pulses
                //-----------
                histo[i]->SetLineWidth(1.0);
                histo[i]->SetLineStyle(1);
                fit[i]->SetLineWidth(1.0);
                fit[i]->SetLineStyle(1);

}

 //=====================================================================

//Want average graph for each energy
//-----------

//Creat some histograms
//---------

 TH1F* TeV1_avsignal = new TH1F("Average_Pulse_1TeV","Average_Pulse_1TeV",250,0.0,20.0);
 TH1F* TeV5_avsignal = new TH1F("Average_Pulse_5TeV","Average_Pulse_5TeV",250,0.0,20.0);
 TH1F* TeV10_avsignal = new TH1F("Average_Pulse_10TeV","Average_Pulse_10TeV",250,0.0,20.0);
 TH1F* TeV20_avsignal = new TH1F("Average_Pulse_20TeV","Average_Pulse_20TeV",250,0.0,20.0);
 TH1F* TeV100_avsignal = new TH1F("Average_Pulse_100TeV","Average_Pulse_100TeV",250,0.0,20.0);

 for(int i = 1; i <=5; i ++){
   TeV1_avsignal->Add(fit[i],0.2);
 }
 for(int i = 6; i <=10; i ++){
   TeV5_avsignal->Add(fit[i],0.2);
 }
 for(int i = 11; i <=15; i ++){
   TeV10_avsignal->Add(fit[i],0.2);
 }
 for(int i = 16; i <=20; i ++){
   TeV20_avsignal->Add(fit[i],0.2);
 }
 for(int i = 21; i <=25; i ++){
   TeV100_avsignal->Add(fit[i],0.2);
```

```
 }

//Set line styles
//------.

TeV1_avsignal->SetLineStyle(1);
TeV1_avsignal->SetLineWidth(2.0);

TeV5_avsignal->SetLineStyle(1);                                                    190
TeV5_avsignal->SetLineWidth(2.0);

TeV10_avsignal->SetLineStyle(1);
TeV10_avsignal->SetLineWidth(2.0);

TeV20_avsignal->SetLineStyle(1);
TeV20_avsignal->SetLineWidth(2.0);

TeV100_avsignal->SetLineStyle(1);
TeV100_avsignal->SetLineWidth(2.0);                                                200

//====================================================================================

//Setting line colours (set individually for obvious reasons)
//----------------------------------

 //1TeV Pulse
 //----.
histo[1]->SetLineColor(1);
histo[2]->SetLineColor(1);                                                         210
histo[3]->SetLineColor(1);
histo[4]->SetLineColor(1);
histo[5]->SetLineColor(1);

 //5TeV Pulses
 //-----
histo[6]->SetLineColor(2);
histo[7]->SetLineColor(2);
histo[8]->SetLineColor(2);
histo[9]->SetLineColor(2);                                                         220
histo[10]->SetLineColor(2);

 //10TeV Pulses
 //-----
histo[11]->SetLineColor(3);
histo[12]->SetLineColor(3);
histo[13]->SetLineColor(3);
histo[14]->SetLineColor(3);
histo[15]->SetLineColor(3);
                                                                                   230
 //20TeV Pulses
 //-----
histo[16]->SetLineColor(4);
histo[17]->SetLineColor(4);
histo[18]->SetLineColor(4);
histo[19]->SetLineColor(4);
histo[20]->SetLineColor(4);

 //100TeV Pulses
```

```
//-----
histo[21]->SetLineColor(6);
histo[22]->SetLineColor(6);
histo[23]->SetLineColor(6);
histo[24]->SetLineColor(6);
histo[25]->SetLineColor(6);

//Average Pulses
//-----
TeV1_avsignal->SetLineColor(1);
TeV5_avsignal->SetLineColor(2);
TeV10_avsignal->SetLineColor(3);
TeV20_avsignal->SetLineColor(4);
TeV100_avsignal->SetLineColor(6);


//set line colours for fits
//----------

//1TeV Pulse
//----
// fit[1]->SetLineColor(1);
// fit[2]->SetLineColor(1);
// fit[3]->SetLineColor(1);
// fit[4]->SetLineColor(1);
// fit[5]->SetLineColor(1);

// //5TeV Pulses
// //-----
// fit[6]->SetLineColor(2);
// fit[7]->SetLineColor(2);
// fit[8]->SetLineColor(2);
// fit[9]->SetLineColor(2);
// fit[10]->SetLineColor(2);

// //10TeV Pulses
// //----
// fit[11]->SetLineColor(3);
// fit[12]->SetLineColor(3);
// fit[13]->SetLineColor(3);
// fit[14]->SetLineColor(3);
// fit[15]->SetLineColor(3);

// //20TeV Pulses
// //-----
// fit[16]->SetLineColor(4);
// fit[17]->SetLineColor(4);
// fit[18]->SetLineColor(4);
// fit[19]->SetLineColor(4);
// fit[20]->SetLineColor(4);

// //100TeV Pulses
// //-----
// fit[21]->SetLineColor(6);
// fit[22]->SetLineColor(6);
// fit[23]->SetLineColor(6);
// fit[24]->SetLineColor(6);
// fit[25]->SetLineColor(6);
```

```
//set axis (set for histo21, as biggest graph, and want axis to be set from this)
//————————————————————————————————————————
                                                                                300

histo[21]->SetTitle("");
histo[21]->GetXaxis()->SetTitle("z / m");
histo[21]->GetXaxis()->CenterTitle();
histo[21]->GetXaxis()->SetTitleOffset(1.0);
histo[21]->GetYaxis()->SetTitle("energy / MeV");
histo[21]->GetYaxis()->SetTitleFont(12);
histo[21]->GetYaxis()->SetTitleSize(0.06);
histo[21]->GetYaxis()->SetTitleOffset(1.0);
histo[21]->GetYaxis()->CenterTitle();                                           310


//=============================================================================

//Drawing the Pulses(again done separately so can turn on or off depending on what you want to look at)
//————————————————————————————————————————

//100TeV pulses (drawn first to set axis correctly)
//————————————————
                                                                                320
 // histo[21]->Draw();
// histo[22]->Draw("Same");
// histo[23]->Draw("Same");
// histo[24]->Draw("Same");
// histo[25]->Draw("Same");

// //1TeV pulses
// //————
// histo[1]->Draw("Same");
// histo[2]->Draw("Same");                                                      330
// histo[3]->Draw("Same");
// histo[4]->Draw("Same");
// histo[5]->Draw("Same");

// //5TeV pulses
// //———-
// histo[6]->Draw("Same");
// histo[7]->Draw("Same");
// histo[8]->Draw("Same");
// histo[9]->Draw("Same");                                                      340
// histo[10]->Draw("Same");

// //10TeV pulses
// //————
// histo[11]->Draw("Same");
// histo[12]->Draw("Same");
// histo[13]->Draw("Same");
// histo[14]->Draw("Same");
// histo[15]->Draw("Same");
                                                                                350
// //20TeV pulses
// //————
// histo[16]->Draw("Same");
// histo[17]->Draw("Same");
// histo[18]->Draw("Same");
```

```
// histo[19]->Draw("Same");
// histo[20]->Draw("Same");


//    TLegend *dedzleg = new TLegend(0.6,0.7,0.89,0.89);
//    dedzleg->AddEntry(histo[1], "1TeV", "l");
//    dedzleg->AddEntry(histo[6], "5TeV", "l");
//    dedzleg->AddEntry(histo[11], "10TeV", "l");
//    dedzleg->AddEntry(histo[16], "20TeV", "l");
//    dedzleg->AddEntry(histo[21], "100TeV", "l");

//   dedzleg->Draw();


 //Average puse for 1TeV
 //---------

 TeV100_avsignal->SetTitle("");
 TeV100_avsignal->GetXaxis()->SetTitle("z / m");
 TeV100_avsignal->GetXaxis()->CenterTitle();
 TeV100_avsignal->GetXaxis()->SetTitleOffset(1.0);
 TeV100_avsignal->GetYaxis()->SetTitle("energy / MeV");
 TeV100_avsignal->GetYaxis()->SetTitleFont(12);
 TeV100_avsignal->GetYaxis()->SetTitleSize(0.06);
 TeV100_avsignal->GetYaxis()->SetTitleOffset(1.0);
 TeV100_avsignal->GetYaxis()->CenterTitle();

 TLegend *dedzavleg = new TLegend(0.6,0.7,0.89,0.89);
 dedzavleg->AddEntry(histo[1], "1TeV", "l");
 dedzavleg->AddEntry(histo[6], "5TeV", "l");
 dedzavleg->AddEntry(histo[11], "10TeV", "l");
 dedzavleg->AddEntry(histo[16], "20TeV", "l");
 dedzavleg->AddEntry(histo[21], "100TeV", "l");

 TeV100_avsignal->Draw();
 TeV1_avsignal->Draw("Same");
 TeV5_avsignal->Draw("Same");
 TeV10_avsignal->Draw("Same");
 TeV20_avsignal->Draw("Same");
 dedzavleg->Draw();

//Code for displaying the fits
//-----------

// fit[21]->SetTitle("");
// fit[21]->GetXaxis()->SetTitle("z / m");
// fit[21]->GetXaxis()->CenterTitle();
// fit[21]->GetXaxis()->SetTitleOffset(1.0);
// fit[21]->GetYaxis()->SetTitle("energy / MeV");
// fit[21]->GetYaxis()->SetTitleFont(12);
// fit[21]->GetYaxis()->SetTitleSize(0.06);
// fit[21]->GetYaxis()->SetTitleOffset(1.0);
// fit[21]->GetYaxis()->CenterTitle();


// //100TeV
// //--
// fit[21]->Draw();
```

```
// fit[22]->Draw("Same");
// fit[23]->Draw("Same");
// fit[24]->Draw("Same");
// fit[25]->Draw("Same");

// //1TeV fits
// //---
// fit[1]->Draw("Same");
// fit[2]->Draw("Same");
// fit[3]->Draw("Same");
// fit[4]->Draw("Same");
// fit[5]->Draw("Same");

// //5TeV fits
// //---
// fit[6]->Draw("Same");
// fit[7]->Draw("Same");
// fit[8]->Draw("Same");
// fit[9]->Draw("Same");
// fit[10]->Draw("Same");

// //10TeV pulses
// //-----
// fit[11]->Draw("Same");
// fit[12]->Draw("Same");
// fit[13]->Draw("Same");
// fit[14]->Draw("Same");
// fit[15]->Draw("Same");

// //20TeV pulses
// //-----
// fit[16]->Draw("Same");
// fit[17]->Draw("Same");
// fit[18]->Draw("Same");
// fit[19]->Draw("Same");
// fit[20]->Draw("Same");

//    TLegend *dedzlegfit = new TLegend(0.6,0.7,0.89,0.89);
//    dedzlegfit->AddEntry(fit[1], "1TeV", "l");
//    dedzlegfit->AddEntry(fit[6], "5TeV", "l");
//    dedzlegfit->AddEntry(fit[11], "10TeV", "l");
//    dedzlegfit->AddEntry(fit[16], "20TeV", "l");
//    dedzlegfit->AddEntry(fit[21], "100TeV", "l");

//   dedzlegfit->Draw();

//=========================================================================

      }
```

## B.8 dedrfitter.C

```
//=========================================================//
//Created by Simon Bevan                                   //
//03/12/03                                                 //
//                                                         //
```

```
//Root macro to get fits for dedr, and get parameters    //
//=============================================================//

#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"

//Code for excluding points below a given x value
//-----------------.
Double_t fitf(Double_t* xx, Double_t *par){

  Double_t x = xx[0];

  if (x<2.0 ){
    TF1::RejectPoint();
    return 0;
  }

  //the fitting function
  //---------
  Double_t fitval = (par[0])*exp(-(x-par[4])/par[1]) + (par[2])*exp(-pow((x-par[5]),2)/pow(par[3],2));
  return fitval;

}

void dedrfitter(){

  // make some virtual files for the 100TeV files
  //------------------
  TChain chain1("ntuple");
  TChain chain2("ntuple");
  TChain chain3("ntuple");
  TChain chain4("ntuple");
  TChain chain5("ntuple");

  //Add files to chains
  //--------
  chain1.Add("100TeV_1/shower_data_00.root");
  chain1.Add("100TeV_1/shower_data_01.root");
  chain1.Add("100TeV_1/shower_data_02.root");
  chain1.Add("100TeV_1/shower_data_03.root");
  chain1.Add("100TeV_1/shower_data_04.root");
  chain1.Add("100TeV_1/shower_data_05.root");

  chain2.Add("100TeV_2/shower_data_00.root");
  chain2.Add("100TeV_2/shower_data_01.root");
  chain2.Add("100TeV_2/shower_data_02.root");
  chain2.Add("100TeV_2/shower_data_03.root");
  chain2.Add("100TeV_2/shower_data_04.root");
  chain2.Add("100TeV_2/shower_data_05.root");
```

```
chain3.Add("100TeV_3/shower_data_00.root");
chain3.Add("100TeV_3/shower_data_01.root");
chain3.Add("100TeV_3/shower_data_02.root");
chain3.Add("100TeV_3/shower_data_03.root");
chain3.Add("100TeV_3/shower_data_04.root");
chain3.Add("100TeV_3/shower_data_05.root");

chain4.Add("100TeV_4/shower_data_00.root");
chain4.Add("100TeV_4/shower_data_01.root");
chain4.Add("100TeV_4/shower_data_02.root");
chain4.Add("100TeV_4/shower_data_03.root");
chain4.Add("100TeV_4/shower_data_04.root");
chain4.Add("100TeV_4/shower_data_05.root");

chain5.Add("100TeV_5/shower_data_00.root");
chain5.Add("100TeV_5/shower_data_01.root");
chain5.Add("100TeV_5/shower_data_02.root");
chain5.Add("100TeV_5/shower_data_03.root");
chain5.Add("100TeV_5/shower_data_04.root");
chain5.Add("100TeV_5/shower_data_05.root");

//Make log file more presentable
//————————————
std::cout << "" <<std::endl;
std::cout << "              LOG FILE FOR GRAPHFITTER() " <<std::endl;
std::cout << "" << std::endl;

//Clean up :
//———-
gROOT->Reset();
gROOT->SetStyle("Plain");

//No stats :
//———-
gStyle->SetOptStat(0);

//Reasonable looking fonts :
//————————————
gStyle->SetLabelFont(12,"x");
gStyle->SetLabelFont(12,"y");
gStyle->SetTitleFont(12);
gStyle->SetTitleSize(0.06);

//Define some variables for later
//————————————————
Double_t p0 = 0;
Double_t p1 = 0;
Double_t p2 = 0;
Double_t p3 = 0;
Double_t p4 = 0;
Double_t p5 = 0;
Double_t p0avtemp = 0;
Double_t p1avtemp = 0;
Double_t p2avtemp = 0;
Double_t p3avtemp = 0;
Double_t p4avtemp = 0;
Double_t p5avtemp = 0;
```

```
//Create some histograms for average graphs
//-----------------------
TH1F energyp0("Energy_vs_Average_p0","Energy_vs_Average_p0", 1500, 10.0, 15.0);
TH1F energyp1("Energy_vs_Average_p1","Energy_vs_Average_p1", 1500, 10.0, 15.0);
TH1F energyp2("Energy_vs_Average_p2","Energy vs Average p2", 1500, 10.0, 15.0);
TH1F energyp3("Energy_vs_Average_p3","Energy vs Average p3", 1500, 10.0, 15.0);
TH1F energyp4("Energy_vs_Average_p4","Energy vs Average p4", 1500, 10.0, 15.0);
TH1F energyp5("Energy_vs_Average_p5","Energy vs Average p5", 1500, 10.0, 15.0);

// Make outpout file for average graphs                                    130
// ------------------
TFile* averagehistoFile = new TFile("average_graphs_dedr.root","RECREATE","neutrinosim geant histos");

// Loop through all of the files
// ============================
// Loop though each energy
// ----------

for (Int_t d = 0; d <= 8 ; d++){
                                                                           140
  // Make output files :
  // --------
  Int_t energy = 0;
  Int_t jmax = 0;

  if(d == 0){
    energy = 1;
    jmax = 9;
  }
  else if(d == 1){                                                         150
    energy = 2;
    jmax = 13;
  }
  else if(d == 2){
    energy = 3;
    jmax = 6;
  }
  else if(d == 3){
    energy = 5;
    jmax = 11;                                                             160
  }
  else if(d == 4){
    energy = 7;
    jmax = 4;
  }
  else if(d == 5){
    energy = 10;
    jmax = 8;
  }
  else if(d == 6){                                                         170
    energy = 15;
    jmax = 5;
  }
  else if(d == 7){
    energy = 20;
    jmax = 7;
  }
  else if(d == 8){
```

```
  energy = 100;
  jmax = 5;
}

//loop hrough files in each energy
//————————————-

//reset parameters for each loop
//————————————
p0avtemp = 0;
p1avtemp = 0;
p2avtemp = 0;
p3avtemp = 0;
p4avtemp = 0;
p5avtemp = 0;

for (Int_t j = 1; j <= jmax; j++) {

  //reset values for loop
  //—————————
  p0 = 0;
  p1 = 0;
  p2 = 0;
  p3 = 0;
  p4 = 0;
  p5 = 0;

  //make a filenames that are appropiate from the loop
  //————————————————————
  Char_t filename[100];
  std::sprintf(filename, "%dTeV_pion_seawater_%d.root",energy,j);

  Char_t filenameout[100];
  std::sprintf(filenameout, "%dTeV_pion_seawater_out_dedr%d.root",energy,j);

  Char_t filenamededr[100];
  std::sprintf(filenamededr, "%dTeV_pion_seawater_%d_dedr",energy,j);

  Char_t filenamededz[100];
  std::sprintf(filenamededz, "%dTeV_pion_seawater_%d_dedz",energy,j);

  Char_t filenamededphi[100];
  std::sprintf(filenamededphi, "%dTeV_pion_seawater_%d_dedphi",energy,j);

  TFile* histoFile = new TFile(filenameout,"RECREATE","neutrinosim geant histos2");

  TH1F dedr(filenamededr,"dedr",1000,0.0,1000.0);
  TH1F dedz(filenamededz,"dedz",250,0.0,20.0);
  TH1F dephi(filenamededphi,"dedphi",100,-1.0*M_PI,1.0*M_PI);

  //open input file
  //——————
  TTree* geant;

  TFile* neutrino;

  if (energy <=20 ){
```

```
  neutrino = new TFile(filename);

  neutrino->cd();
  neutrino->ls();

  geant = (TTree*)(neutrino->Get("ntuple"));
}

//for 100TeV need to open virtual files
//-----------------
if (energy > 20){

  if(j==1){
    geant = (TTree*)(&chain1);
  }
  if(j==2){
    geant = (TTree*)(&chain2);
  }
  if(j==3){
    geant = (TTree*)(&chain3);
  }
  if(j==4){
    geant = (TTree*)(&chain4);
  }
  if(j==5){
    geant = (TTree*)(&chain5);
  }

}

//Store results in a histogram :
//----------
Char_t titlededr[100];
std::sprintf(titlededr,"%s_dedr ",filename);

Char_t titlededz[100];
std::sprintf(titlededz,"%s_dedz ",filename);

Char_t titledephi[100];
std::sprintf(titledephi,"%s_dedphi ",filename);

//Loop through ntuple, filling arrays :
//---------------
Int_t n = (Int_t)(geant->GetEntries());

std::cout << "Ntuple has " << n << " entries" << std::endl;

Float_t totE = 0.0;
Float_t totstepleng = 0.0;

for (Int_t i = 0; i < n; i++) {
  geant->GetEvent(i);

  // x0,y0,z0 : start of GEANT step (mm)
  // x1,y1,z1 : end of GEANT step (mm)
  // de       : energy deposited (MeV)

  Float_t x0      = geant->GetLeaf("x0")->GetValue();
```

```
  Float_t y0     = geant->GetLeaf("y0")->GetValue();
  Float_t z0     = geant->GetLeaf("z0")->GetValue();
  Float_t x1     = geant->GetLeaf("x1")->GetValue();
  Float_t y1     = geant->GetLeaf("y1")->GetValue();
  Float_t z1     = geant->GetLeaf("z1")->GetValue();
  Float_t de     = geant->GetLeaf("de")->GetValue();                    300

  // Check of total energy deposition :
  totE += de;

  // Take point of energy deposition to be midway through the step :
  Float_t x_dep = (x0+x1)/2.0;
  Float_t y_dep = (y0+y1)/2.0;
  Float_t z_dep = (z0+z1)/2.0;

  Float_t r_dep = sqrt(pow(x_dep,2)+pow(y_dep,2));                      310

  Float_t phi_dep = atan2(x_dep,y_dep);

  dedr.Fill(r_dep,de);
  dedz.Fill(z_dep/1000.0,de);
  dephi.Fill(phi_dep,de);

}

for (Int_t w=0; w<= dedr.GetNbinsX(); ++w) {                            320
  dedr.SetBinError(w,500.0);
}

//Defining the fit function
//-----------
TF1 *func = new TF1("swbfit", fitf, 2.0 , 1000.0, 6);
func->SetNpx(1000);

func->SetRange(2.0,1000.0);
func->SetParameters(10000.0,10.0,10.0,10.0,10.0,10.0);                  330

dedr.Fit(func);

TF1 *fit = dedr.GetFunction("swbfit");
fit->SetNpx(1000);

dedr.Draw();

histoFile->cd(); dedz.Write(); dedr.Write(); dephi.Write(); fit->Write();
                                                                        340
// Analyse the fit
// ------
p0     = fit->GetParameter(0);
p1     = fit->GetParameter(1);
p2     = fit->GetParameter(2);
p3     = fit->GetParameter(3);
p4     = fit->GetParameter(4);
p5     = fit->GetParameter(5);

//Get Averages                                                          350
//-----
p0avtemp = p0avtemp + p0;
```

```
    p1avtemp = p1avtemp + p1;
    p2avtemp = p2avtemp + p2;
    p3avtemp = p3avtemp + p3;
    p4avtemp = p4avtemp + p4;
    p5avtemp = p5avtemp + p5;


    //Print everything to screen                                                    360
    //————————————-

    std::cout << "" <<std::endl;
    std::cout << "" <<std::endl;

    std::cout << "" <<std::endl;
    std::cout << "-----------------------------------------------------" <<std::endl;
    std::cout << "" <<std::endl;

    histoFile->Close();                                                           370

    delete func;

  }

  std::cout << "-----------------------------------------------------" << std::endl;
  std::cout <<""<<std::endl;
  std::cout << "             " << energy << "TeV analysis"<<std::endl;
  std::cout <<""<<std::endl;
                                                                                  380
  Double_t p0av = p0avtemp/jmax;
  Double_t p1av = p1avtemp/jmax;
  Double_t p2av = p2avtemp/jmax;
  Double_t p3av = p3avtemp/jmax;
  Double_t p4av = p4avtemp/jmax;
  Double_t p5av = p5avtemp/jmax;

  std::cout << "The average value for p0 for " << energy << "TeV is " << p0av << std::endl;
  std::cout << "The average value for p1 for " << energy << "TeV is " << p1av << std::endl;
  std::cout << "The average value for p2 for " << energy << "TeV is " << p2av << std::endl; 390
  std::cout << "The average value for p3 for " << energy << "TeV is " << p3av << std::endl;
  std::cout << "The average value for p4 for " << energy << "TeV is " << p4av << std::endl;
  std::cout << "The average value for p5 for " << energy << "TeV is " << p5av << std::endl;

  Double_t logenergy = log10(energy*pow(10,12));

  energyp0.Fill(logenergy,p0av);
  energyp1.Fill(logenergy,p1av);
  energyp2.Fill(logenergy,p2av);
  energyp3.Fill(logenergy,p3av);                                                  400
  energyp4.Fill(logenergy,p4av);
  energyp5.Fill(logenergy,p5av);


  std::cout << "" <<std::endl;
  std::cout << "=====================================================" <<std::endl;
  std::cout << "" <<std::endl;

}
                                                                                  410
```

```
  averagehistoFile->cd(); energyp0.Write(); energyp1.Write(); energyp2.Write(); energyp3.Write(); energyp4.Write(); energyp5.Write

  averagehistoFile->Close();

      }
```

## B.9 dedrcomprevised.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//29/11/03                                                     //
//                                                             //
//Revised 12/01/04 to tidy code                                //
//                                                             //
//Root macro for displaying dedr                               //
//from output of graphfitter()                                 //
//=============================================================//
                                                                                10

#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"                                                              20
#include "TChain.h"
#include "TCanvas.h"


//fit function for dedr
//---------
Double_t fitf(Double_t* xx, Double_t *par){
  Double_t x = xx[0];

  //excludes ponts for x<2                                                      30
  //---------
  if(x<2.0){
    TF1::RejectPoint();
    return 0;
  }

  //the fit function
  //------
 Double_t fitval = (par[0])*exp(-(x-par[4])/par[1]) + (par[2])*exp(-pow((x-par[5]),2)/pow(par[3],2));
 return fitval;                                                                 40

}

void dedrcomprevised() {

//=============================================================

//Set up for root
```

```
//==============
                                                                    50
// Clean up :
// ==========
gROOT->Reset();

// No stats :
// ==========
gStyle->SetOptStat(0);
gROOT->SetStyle("Plain");

c1 = new TCanvas("c1","neutrinosim",200,10,700,500);                60

c1->SetFillColor(0);
c1->SetGridx();
c1->SetGridy();

//=============================================================

//Loop through all files creating histograms
//-------------------
                                                                    70
 TFile *pulse[26];
 TH1F *histo[26];
 TF1 *funcfit[26];
 TF1 *fit[26];

 for(int i = 1; i<=20 ; i++){//

   Int_t energy = 0;
   Int_t j = 0;
   Int_t pulsenum = 0;                                              80
   Int_t histonum = 0;

   if((i>=1)&&(i<=5)){
     energy = 1;
     j=i;
   }

   if((i>=6)&&(i<=10)){
     energy = 5;
     j = i-5;                                                       90
   }

   if((i>=11)&&(i<=15)){
     energy = 10;
     j = i-10;
   }

   if((i>=16)&&(i<=20)){
     energy = 20;
          j = i-15;                                                 100
   }

              Char_t filename[100];
              std::sprintf(filename, "%dTeV_pion_seawater_out_%d.root",energy,j);
              Char_t filenamekey[100];
              std::sprintf(filenamekey, "%dTeV_pion_seawater_%d_dedr",energy,j);
```

```
            pulse[i] = new TFile(filename);
            histo[i] = (TH1F*) pulse[i]->Get(filenamekey);
}

//100TeV have different key name, so have to make implicitly
//-------------------------
pulse[21] = new TFile("100TeV_pion_seawater_out_1.root");
histo[21] = (TH1F*) pulse[21]->Get("100TeV_pion_seawater_1_dedr");
pulse[22] = new TFile("100TeV_pion_seawater_out_2.root");
histo[22] = (TH1F*) pulse[22]->Get("100TeV_pion_seawater_2_dedr");
pulse[23] = new TFile("100TeV_pion_seawater_out_3.root");
histo[23] = (TH1F*) pulse[23]->Get("100TeV_pion_seawater_3_dedr");
pulse[24] = new TFile("100TeV_pion_seawater_out_4.root");
histo[24] = (TH1F*) pulse[24]->Get("100TeV_pion_seawater_4_dedr");
pulse[25] = new TFile("100TeV_pion_seawater_out_5.root");
histo[25] = (TH1F*) pulse[25]->Get("100TeV_pion_seawater_5_dedr");

for(int i = 1; i<=25; i++){

              //adding errors to the graphs
              //----------
              for (Int_t w=0; w<= histo[i]->GetNbinsX(); w++) {
                histo[i]->SetBinError(w,500.0);
             }

              //fitting the function
              //--------
              Char_t filenamefit[100];
              std::sprintf(filenamefit, "pion_seawater_%d_dedr_fit",i);
              funcfit[i] = new TF1(filenamefit, fitf, 0.0, 10.0, 6);
              funcfit[i]->SetParameters(10000.0,1.0,1.0,1.0,1.0,1.0);

              histo[i].Fit(filenamefit);
              fit[i] = histo[i].GetFunction(filenamefit);

              //set line styles for all pulses
              //----------
              histo[i]->SetLineWidth(1.0);
              histo[i]->SetLineStyle(1);
              fit[i]->SetLineWidth(1.0);
              fit[i]->SetLineStyle(1);

}

 //=========================================================================

//Want average graph for each energy
//----------

//Creat some histograms
//---------

 TH1F* TeV1_avsignal = new TH1F("Average_Pulse_1TeV","Average_Pulse_1TeV",1000,0.0,1000.0);
 TH1F* TeV5_avsignal = new TH1F("Average_Pulse_5TeV","Average_Pulse_5TeV",1000,0.0,1000.0);
 TH1F* TeV10_avsignal = new TH1F("Average_Pulse_10TeV","Average_Pulse_10TeV",1000,0.0,1000.0);
 TH1F* TeV20_avsignal = new TH1F("Average_Pulse_20TeV","Average_Pulse_20TeV",1000,0.0,1000.0);
  TH1F* TeV100_avsignal = new TH1F("Average_Pulse_100TeV","Average_Pulse_100TeV",1000,0.0,1000.0);
```

```
for(int i = 1; i <=5; i ++){
  TeV1_avsignal->Add(histo[i],0.2);
}
for(int i = 6; i <=10; i ++){
  TeV5_avsignal->Add(histo[i],0.2);                                    170
}
for(int i = 11; i <=15; i ++){
  TeV10_avsignal->Add(histo[i],0.2);
}
for(int i = 16; i <=20; i ++){
  TeV20_avsignal->Add(histo[i],0.2);
}
for(int i = 21; i <=25; i ++){
  TeV100_avsignal->Add(histo[i],0.2);
 }                                                                     180

//Set line styles
//------

TeV1_avsignal->SetLineStyle(1);
TeV1_avsignal->SetLineWidth(2.0);

TeV5_avsignal->SetLineStyle(1);
TeV5_avsignal->SetLineWidth(2.0);
                                                                       190
TeV10_avsignal->SetLineStyle(1);
TeV10_avsignal->SetLineWidth(2.0);

TeV20_avsignal->SetLineStyle(1);
TeV20_avsignal->SetLineWidth(2.0);

TeV100_avsignal->SetLineStyle(1);
TeV100_avsignal->SetLineWidth(2.0);

//=============================================================================== 200

//Setting line colours (set individually for obvious reasons)
//--------------------------

 //1TeV Pulse
 //----
histo[1]->SetLineColor(1);
histo[2]->SetLineColor(1);
histo[3]->SetLineColor(1);
histo[4]->SetLineColor(1);                                             210
histo[5]->SetLineColor(1);

 //5TeV Pulses
 //-----
histo[6]->SetLineColor(2);
histo[7]->SetLineColor(2);
histo[8]->SetLineColor(2);
histo[9]->SetLineColor(2);
histo[10]->SetLineColor(2);
                                                                       220
 //10TeV Pulses
 //-----
```

```
histo[11]->SetLineColor(3);
histo[12]->SetLineColor(3);
histo[13]->SetLineColor(3);
histo[14]->SetLineColor(3);
histo[15]->SetLineColor(3);

//20TeV Pulses
//-----
histo[16]->SetLineColor(4);
histo[17]->SetLineColor(4);
histo[18]->SetLineColor(4);
histo[19]->SetLineColor(4);
histo[20]->SetLineColor(4);

//100TeV Pulses
//-----
histo[21]->SetLineColor(6);
histo[22]->SetLineColor(6);
histo[23]->SetLineColor(6);
histo[24]->SetLineColor(6);
histo[25]->SetLineColor(6);

//Average Pulses
//-----
TeV1_avsignal->SetLineColor(1);
TeV5_avsignal->SetLineColor(2);
TeV10_avsignal->SetLineColor(3);
TeV20_avsignal->SetLineColor(4);
TeV100_avsignal->SetLineColor(5);


//set line colours for fits
//----------

//1TeV Pulse
//----
fit[1]->SetLineColor(1);
fit[2]->SetLineColor(1);
fit[3]->SetLineColor(1);
fit[4]->SetLineColor(1);
fit[5]->SetLineColor(1);

//5TeV Pulses
//-----
fit[6]->SetLineColor(2);
fit[7]->SetLineColor(2);
fit[8]->SetLineColor(2);
fit[9]->SetLineColor(2);
fit[10]->SetLineColor(2);

//10TeV Pulses
//-----
fit[11]->SetLineColor(3);
fit[12]->SetLineColor(3);
fit[13]->SetLineColor(3);
fit[14]->SetLineColor(3);
fit[15]->SetLineColor(3);
```

```
//20TeV Pulses
//-----
fit[16]->SetLineColor(4);
fit[17]->SetLineColor(4);
fit[18]->SetLineColor(4);
fit[19]->SetLineColor(4);
fit[20]->SetLineColor(4);

//100TeV Pulses
//-----                                                                    290
fit[21]->SetLineColor(6);
fit[22]->SetLineColor(6);
fit[23]->SetLineColor(6);
fit[24]->SetLineColor(6);
fit[25]->SetLineColor(6);

//set axis (set for histo21, as biggest graph, and want axis to be set from this)
//-------------------------------------------

histo[21]->SetTitle("");                                                 300
histo[21]->GetXaxis()->SetTitle("r / cm");
histo[21]->GetXaxis()->CenterTitle();
histo[21]->GetXaxis()->SetTitleOffset(1.0);
histo[21]->GetYaxis()->SetTitle("energy / MeV");
histo[21]->GetYaxis()->SetTitleFont(12);
histo[21]->GetYaxis()->SetTitleSize(0.06);
histo[21]->GetYaxis()->SetTitleOffset(1.0);
histo[21]->GetYaxis()->CenterTitle();

                                                                         310
//=============================================================================

//Drawing the Pulses(again done separately so can turn on or off depending on what you want to look at)
//-------------------------------------------

//100TeV pulses (drawn first to set axis correctly)
//-----------------

 // histo[21]->Draw();
//  histo[22]->Draw("Same");                                               320
//  histo[23]->Draw("Same");
//  histo[24]->Draw("Same");
//  histo[25]->Draw("Same");

//  //1TeV pulses
//  //----
//  histo[1]->Draw("Same");
//  histo[2]->Draw("Same");
//  histo[3]->Draw("Same");
//  histo[4]->Draw("Same");                                                330
//  histo[5]->Draw("Same");

//  //5TeV pulses
//  //----
//  histo[6]->Draw("Same");
//  histo[7]->Draw("Same");
//  histo[8]->Draw("Same");
//  histo[9]->Draw("Same");
```

```
// histo[10]->Draw("Same");

// //10TeV pulses
// //-----
// histo[11]->Draw("Same");
// histo[12]->Draw("Same");
// histo[13]->Draw("Same");
// histo[14]->Draw("Same");
// histo[15]->Draw("Same");

// //20TeV pulses
// //-----
// histo[16]->Draw("Same");
// histo[17]->Draw("Same");
// histo[18]->Draw("Same");
// histo[19]->Draw("Same");
// histo[20]->Draw("Same");


//    TLegend *dedrleg = new TLegend(0.6,0.7,0.89,0.89);
//    dedrleg->AddEntry(histo[1], "1TeV", "l");
//    dedrleg->AddEntry(histo[6], "5TeV", "l");
//    dedrleg->AddEntry(histo[11], "10TeV", "l");
//    dedrleg->AddEntry(histo[16], "20TeV", "l");
//    dedrleg->AddEntry(histo[21], "100TeV", "l");

//   dedrleg->Draw();


 //Average puse for 1TeV
 //--------

 TeV100_avsignal->SetTitle("");
 TeV100_avsignal->GetXaxis()->SetTitle("r / cm");
 TeV100_avsignal->GetXaxis()->CenterTitle();
 TeV100_avsignal->GetXaxis()->SetTitleOffset(1.0);
 TeV100_avsignal->GetYaxis()->SetTitle("energy / MeV");
 TeV100_avsignal->GetYaxis()->SetTitleFont(12);
 TeV100_avsignal->GetYaxis()->SetTitleSize(0.06);
 TeV100_avsignal->GetYaxis()->SetTitleOffset(1.0);
 TeV100_avsignal->GetYaxis()->CenterTitle();

 TLegend *dedravleg = new TLegend(0.6,0.7,0.89,0.89);
 dedravleg->AddEntry(histo[1], "1TeV", "l");
 dedravleg->AddEntry(histo[6], "5TeV", "l");
 dedravleg->AddEntry(histo[11], "10TeV", "l");
 dedravleg->AddEntry(histo[16], "20TeV", "l");
 dedravleg->AddEntry(histo[21], "100TeV", "l");

 TeV100_avsignal->Draw();
 TeV1_avsignal->Draw("Same");
 TeV5_avsignal->Draw("Same");
 TeV10_avsignal->Draw("Same");
 TeV20_avsignal->Draw("Same");
 dedravleg->Draw();

//Code for displaying the fits
//----------
```

```
// fit[21]->SetTitle("");
// fit[21]->GetXaxis()->SetTitle("r / cm");
// fit[21]->GetXaxis()->CenterTitle();
// fit[21]->GetXaxis()->SetTitleOffset(1.0);
// fit[21]->GetYaxis()->SetTitle("energy / MeV");
// fit[21]->GetYaxis()->SetTitleFont(12);
// fit[21]->GetYaxis()->SetTitleSize(0.06);
// fit[21]->GetYaxis()->SetTitleOffset(1.0);
// fit[21]->GetYaxis()->CenterTitle();


// //100TeV
// //——
// fit[21]->Draw();
// fit[22]->Draw("Same");
// fit[23]->Draw("Same");
// fit[24]->Draw("Same");
// fit[25]->Draw("Same");

// //1TeV fits
// //———
// fit[1]->Draw("Same");
// fit[2]->Draw("Same");
// fit[3]->Draw("Same");
// fit[4]->Draw("Same");
// fit[5]->Draw("Same");

// //5TeV fits
// //——-
// fit[6]->Draw("Same");
// fit[7]->Draw("Same");
// fit[8]->Draw("Same");
// fit[9]->Draw("Same");
// fit[10]->Draw("Same");

// //10TeV pulses
// //————
// fit[11]->Draw("Same");
// fit[12]->Draw("Same");
// fit[13]->Draw("Same");
// fit[14]->Draw("Same");
// fit[15]->Draw("Same");

// //20TeV pulses
// //————
// fit[16]->Draw("Same");
// fit[17]->Draw("Same");
// fit[18]->Draw("Same");
// fit[19]->Draw("Same");
// fit[20]->Draw("Same");

//    TLegend *dedrlegfit = new TLegend(0.6,0.7,0.89,0.89);
//    dedrlegfit->AddEntry(fit[1], "1TeV", "l");
//    dedrlegfit->AddEntry(fit[6], "5TeV", "l");
//    dedrlegfit->AddEntry(fit[11], "10TeV", "l");
//    dedrlegfit->AddEntry(fit[16], "20TeV", "l");
//    dedrlegfit->AddEntry(fit[21], "100TeV", "l");
```

```
//   dedrlegfit->Draw();

//=============================================================================

    }                                                                          460
```

## B.10 dedr_dz_maker.C

```
//=============================================//
//Created by Simon Bevan                       //
//03/12/03                                     //
//                                             //
//Root macro for creating dedr graphs at       //
//varying Z for 1,10, 20, and 100TeV           //
//=============================================//


#include <stdio.h>                                                             10
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"

void dedr_dz_maker(){                                                          20

  // Clean up :
  // ----------
  gROOT->Reset();
  gROOT->SetStyle("Plain");

  // This gives nice purple->red color scale for 2-D histogram plotting
  gStyle->SetPalette(1);

  // No stats :                                                                30
  // ----------
  gStyle->SetOptStat(0);
  gStyle->SetLabelFont(12,"x");
  gStyle->SetLabelFont(12,"y");
  gStyle->SetTitleFont(12);
  gStyle->SetTitleSize(0.06);

  Char_t pad_name[120];

  //make arrays for filenames                                                40
  //-------------------
  Char_t filenamededr1[30];
  Char_t filenamededr2[30];
  Char_t filenamededr3[30];
  Char_t filenamededr4[30];
  Char_t filenamededr5[30];
  Char_t filenamededr6[30];
```

```
Char_t filenamededr7[30];
Char_t filenamededr8[30];
Char_t filenamededr9[30];
Char_t filenamededr10[30];

// Loop through all of the files
// ============================
// Loop though each energy
// ----------
for (Int_t d = 0; d <= 3 ; d++){

  // Define names of file that is to be read :
  // ----------------
  Int_t energy = 0;
  Int_t jmax = 0;

  if(d == 0){
    energy = 1;
    jmax = 5;
  }
  else if(d == 1){
    energy = 10;
    jmax = 5;
  }
  else if(d == 2){
    energy = 20;
    jmax = 5;
  }
  else if(d == 3){
    energy = 100;
    jmax = 5;
  }

  //loop through files in each energy
  //------------
  for (Int_t j = 1; j <= jmax; j++) {

    //make filename of input file
    //-----------
    Char_t filename[100];
    std::sprintf(filename, "%dTeV_pion_seawater_%d.root",energy,j);

    //make a filenames for output file
    //-------------
    std::sprintf(filenamededr1, "%dTeV_pion_seawater_%d_dedr_1",energy,j);
    std::sprintf(filenamededr2, "%dTeV_pion_seawater_%d_dedr_2",energy,j);
    std::sprintf(filenamededr3, "%dTeV_pion_seawater_%d_dedr_3",energy,j);
    std::sprintf(filenamededr4, "%dTeV_pion_seawater_%d_dedr_4",energy,j);
    std::sprintf(filenamededr5, "%dTeV_pion_seawater_%d_dedr_5",energy,j);
    std::sprintf(filenamededr6, "%dTeV_pion_seawater_%d_dedr_6",energy,j);
    std::sprintf(filenamededr7, "%dTeV_pion_seawater_%d_dedr_7",energy,j);
    std::sprintf(filenamededr8, "%dTeV_pion_seawater_%d_dedr_8",energy,j);
    std::sprintf(filenamededr9, "%dTeV_pion_seawater_%d_dedr_9",energy,j);
    std::sprintf(filenamededr10, "%dTeV_pion_seawater_%d_dedr_10",energy,j);

    //Book some hisograms to fill
    //----------
    TH1F dedr1(filenamededr1,"dedr",1000,0.0,1000.0);
```

```
    TH1F dedr2(filenamededr2,"dedr",1000,0.0,1000.0);
    TH1F dedr3(filenamededr3,"dedr",1000,0.0,1000.0);
    TH1F dedr4(filenamededr4,"dedr",1000,0.0,1000.0);
    TH1F dedr5(filenamededr5,"dedr",1000,0.0,1000.0);
    TH1F dedr6(filenamededr6,"dedr",1000,0.0,1000.0);
    TH1F dedr7(filenamededr7,"dedr",1000,0.0,1000.0);
    TH1F dedr8(filenamededr8,"dedr",1000,0.0,1000.0);
    TH1F dedr9(filenamededr9,"dedr",1000,0.0,1000.0);
    TH1F dedr10(filenamededr10,"dedr",1000,0.0,1000.0);

    Char_t filenameout[100];
    std::sprintf(filenameout, "%dTeV_pion_seawater_%d_dedzdr.root",energy,j);

    // Make output files :
    // ---------
    TFile* histoFile = new TFile(filenameout,"RECREATE","neutrinosim geant histos2");

    // Open input file :
    // -------

// Read in GEANT shower information :
//-------------------

    if(energy <= 20){
      TFile neutrino(filename);
      neutrino.cd(); neutrino.ls();
      TNtuple* geant = (TNtuple*)(neutrino.Get("ntuple"));
    }

    if(energy > 20){
      TChain chain1("ntuple");

      Char_t TeV100_0[100];
      std::sprintf(TeV100_0,"100TeV_%d/shower_data_00.root",j);
      Char_t TeV100_1[100];
      std::sprintf(TeV100_1,"100TeV_%d/shower_data_01.root",j);
      Char_t TeV100_2[100];
      std::sprintf(TeV100_2,"100TeV_%d/shower_data_02.root",j);
      Char_t TeV100_3[100];
      std::sprintf(TeV100_3,"100TeV_%d/shower_data_03.root",j);
      Char_t TeV100_4[100];
      std::sprintf(TeV100_4,"100TeV_%d/shower_data_04.root",j);
      Char_t TeV100_5[100];
      std::sprintf(TeV100_5,"100TeV_%d/shower_data_05.root",j);

      chain1.Add(TeV100_0);
      chain1.Add(TeV100_1);
      chain1.Add(TeV100_2);
      chain1.Add(TeV100_3);
      chain1.Add(TeV100_4);
      chain1.Add(TeV100_5);
      TTree* geant = (TTree*)(&chain1);
    }

int nstep = (int)(geant->GetEntries());

    // Loop through ntuple, filling arrays :
    //-----------------
```

```
Int_t n = (Int_t)(geant->GetEntries());
std::cout << "Ntuple has " << n << " entries" << std::endl;

for (Int_t i = 0; i < n; i++) {
  geant->GetEvent(i);

  // x0,y0,z0 : start of GEANT step (mm)                                    170
  // x1,y1,z1 : end of GEANT step (mm)
  // de       : energy deposited (MeV)

  Float_t x0     = geant->GetLeaf("x0")->GetValue();
  Float_t y0     = geant->GetLeaf("y0")->GetValue();
  Float_t z0     = geant->GetLeaf("z0")->GetValue();
  Float_t x1     = geant->GetLeaf("x1")->GetValue();
  Float_t y1     = geant->GetLeaf("y1")->GetValue();
  Float_t z1     = geant->GetLeaf("z1")->GetValue();
  Float_t de     = geant->GetLeaf("de")->GetValue();                        180

  // Take point of energy deposition to be midway through the step :
  Float_t x_dep = (x0+x1)/2.0;
  Float_t y_dep = (y0+y1)/2.0;
  Float_t z_dep = (z0+z1)/(2.0*1000.0);

  Float_t r_dep = sqrt(pow(x_dep,2)+pow(y_dep,2));

  // Store results in a histogram :
  // ---------                                                              190
  if(z_dep <= 1){
    dedr1.Fill(r_dep,de);
  }
  else if((z_dep <= 2) && (z_dep > 1)){
    dedr2.Fill(r_dep,de);
  }
  else if((z_dep <= 3) && (z_dep > 2)){
    dedr3.Fill(r_dep,de);
  }
  else if((z_dep <= 4) && (z_dep > 3)){                                     200
    dedr4.Fill(r_dep,de);
  }
  else if((z_dep <= 5) && (z_dep > 4)){
    dedr5.Fill(r_dep,de);
  }
  else if((z_dep <= 6) && (z_dep > 5)){
    dedr6.Fill(r_dep,de);
  }
  else if((z_dep <= 7) && (z_dep > 6)){
                                                                            210
    dedr7.Fill(r_dep,de);
  }
  else if((z_dep <= 8) && (z_dep > 7)){
    dedr8.Fill(r_dep,de);
  }
  else if((z_dep <= 9) && (z_dep > 8)){
    dedr9.Fill(r_dep,de);
  }
  else if((z_dep <= 10) && (z_dep > 9)){
    dedr10.Fill(r_dep,de);                                                  220
  }
```

```
    }

    histoFile->cd();
    dedr1.Write();dedr2.Write();dedr3.Write();dedr4.Write();dedr5.Write();dedr6.Write();dedr7.Write();
    dedr8.Write();dedr9.Write();dedr10.Write();

  }
                                                                                    230
  histoFile->Close();

}
    }
```

## B.11 dedr_dz_plotter.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//03/12/03                                                     //
//                                                             //
//revised 14/01/04 to tidy code and add 100TeV results         //
//                                                             //
//Root macro for displaying and dedr                           //
//at varying z from output of dedzdr maker                     //
//                                                             //
//=============================================================// 10

void dedr_dz_plotter(){//start

  //Setup for root
  //==============

  //Clean up :
  //----
  gROOT->Reset();
                                                                                    20
  //No stats :
  //----
  gStyle->SetOptStat(0);
  gROOT->SetStyle("Plain");

  c1 = new TCanvas("c1","neutrinosim",200,10,700,500);

  c1->SetFillColor(0);
  c1->SetGridx();
  c1->SetGridy();                                                                   30

  //=============================================================================

  //Create some histograms for the averages
  //---------------------------------------

  TH1F *avhisto1[15];
  TH1F *avhisto10[15];
  TH1F *avhisto20[15];
  TH1F *avhisto100[15];                                                             40
```

```
//loop creating average histograms
//————————————
 for(int p = 1; p<=10; p++){

  Char_t avgraphtitle100[100];
  std::sprintf(avgraphtitle100, "dedr_at_varying_z_for_100TeV_%p.root",p);
  Char_t avgraphtitle20[100];
  std::sprintf(avgraphtitle20, "dedr_at_varying_z_for_20TeV_%p.root",p);
  Char_t avgraphtitle10[100];
  std::sprintf(avgraphtitle10, "dedr_at_varying_z_for_10TeV_%p.root",p);
  Char_t avgraphtitle1[100];
  std::sprintf(avgraphtitle1, "dedr_at_varying_z_for_1TeV_%p.root",p);

  avhisto1[p] = new TH1F(avgraphtitle1,avgraphtitle1,1000,0.0,1000.0);
  avhisto10[p] = new TH1F(avgraphtitle10,avgraphtitle10,1000,0.0,1000.0);
  avhisto20[p] = new TH1F(avgraphtitle20,avgraphtitle20,1000,0.0,1000.0);
  avhisto100[p] = new TH1F(avgraphtitle100,avgraphtitle100,1000,0.0,1000.0);
}


//Loop through all files creating histograms
//————————————————
TFile *pulse[230];
TH1F *histo[230];

int pulse_num = 0;
int j = 0;

for(int i = 1; i<=18 ; i++){
  Int_t energy = 0;

 if((i>=1)&&(i<=5)){
    energy = 1;
    j = i;
  }
  if((i>=6)&&(i<=10)){
    energy = 10;
    j = i-5;
  }
  if((i>=11)&&(i<=15)){
    energy = 20;
    j = i-10;
  }
  if((i>=16)&&(i<=18)){
    energy = 100;
    j = i-15;
  }

  Char_t filename[50];
  std::sprintf(filename, "%dTeV_pion_seawater_%d_dedzdr.root",energy,j);

  Char_t key[50];
    for(int k = 1; k<=10; k++){

      pulse_num = pulse_num + 1;

      std::sprintf(key, "%dTeV_pion_seawater_%d_dedr_%d",energy,j,k);
```

```
      pulse[pulse_num] = new TFile(filename);
      histo[pulse_num] = (TH1F*) pulse[pulse_num]->Get(key);

    }

    std::sprintf(key, "%dTeV_pion_seawater_%d_dedr_1",energy,j);
    pulse[pulse_num] = new TFile(filename);
    histo[pulse_num] = (TH1F*) pulse[pulse_num]->Get(key);

}

//calculate averages for each slice
//------------
int j = 0;
int n = 0;
for(int k=1; k<=10; k++){
  n=k;
  for( n ; n <= 50; n = n+10){

    avhisto1[k]->Add(histo[n],0.2);
  }
}

j = 0;
n = 0;
for(int k=1; k<=10; k++){
  j=k;
  n=k;
  for( n+50; n <= 100; n = n+10){
    avhisto10[j]->Add(histo[n],0.2);
  }
}

j = 0;
n = 0;
for(int k=1; k<=10; k++){
  j=k;
  n=k;
  for( n+100; n <= 150; n = n+10){
    avhisto20[j]->Add(histo[n],0.2);
  }
}

j = 0;
n = 0;
for(int k=1; k<=10; k++){
  j=k;
  n=k;
  for( n+150; n <= 180; n = n+10){
    avhisto100[j]->Add(histo[n],0.33333333333);
  }
}


//Displaying graphs (only displaying average graphs rather than all individual graphs)
//=================================================================================

//set axis
```

```
//========
avhisto100[4]->SetTitle("");
avhisto100[4]->GetXaxis()->SetTitle("r / cm");
avhisto100[4]->GetXaxis()->CenterTitle();
avhisto100[4]->GetXaxis()->SetTitleOffset(1.0);
avhisto100[4]->GetYaxis()->SetTitle("energy / MeV");
avhisto100[4]->GetYaxis()->SetTitleFont(12);
avhisto100[4]->GetYaxis()->SetTitleSize(0.06);
avhisto100[4]->GetYaxis()->SetTitleOffset(1.0);
avhisto100[4]->GetYaxis()->CenterTitle();

//set line colour (set individually, to allow easy comparrison)
//=============================================================

//1TeV
//--
avhisto1[1]->SetLineColor(10);
avhisto1[2]->SetLineColor(9);
avhisto1[3]->SetLineColor(1);
avhisto1[4]->SetLineColor(5);
avhisto1[5]->SetLineColor(4);
avhisto1[6]->SetLineColor(2);
avhisto1[7]->SetLineColor(3);
avhisto1[8]->SetLineColor(6);
avhisto1[9]->SetLineColor(7);
avhisto1[10]->SetLineColor(8);

//10TeV
//--
avhisto10[1]->SetLineColor(10);
avhisto10[2]->SetLineColor(9);
avhisto10[3]->SetLineColor(1);
avhisto10[4]->SetLineColor(3);
avhisto10[5]->SetLineColor(4);
avhisto10[6]->SetLineColor(2);
avhisto10[7]->SetLineColor(3);
avhisto10[8]->SetLineColor(6);
avhisto10[9]->SetLineColor(7);
avhisto10[10]->SetLineColor(8);

//20TeV
//--
avhisto20[1]->SetLineColor(10);
avhisto20[2]->SetLineColor(9);
avhisto20[3]->SetLineColor(1);
avhisto20[4]->SetLineColor(2);
avhisto20[5]->SetLineColor(4);
avhisto20[6]->SetLineColor(2);
avhisto20[7]->SetLineColor(3);
avhisto20[8]->SetLineColor(6);
avhisto20[9]->SetLineColor(7);
avhisto20[10]->SetLineColor(8);

//100TeV
//--
avhisto100[1]->SetLineColor(10);
avhisto100[2]->SetLineColor(9);
avhisto100[3]->SetLineColor(1);
```

```
  avhisto100[4]->SetLineColor(1);
  avhisto100[5]->SetLineColor(4);
  avhisto100[6]->SetLineColor(2);
  avhisto100[7]->SetLineColor(3);
  avhisto100[8]->SetLineColor(6);
  avhisto100[9]->SetLineColor(7);                                                    220
  avhisto100[10]->SetLineColor(8);


  //Average graphs (pick the ones you want to see)
  //————————————————————

  //100TeV
  //——
//   avhisto100[5]->Draw();
//   avhisto100[1]->Draw("SAME");                                                   230
//   avhisto100[2]->Draw("SAME");
//   avhisto100[3]->Draw("SAME");
//   avhisto100[4]->Draw("SAME");
//   avhisto100[6]->Draw("SAME");
//   avhisto100[7]->Draw("SAME");
//   avhisto100[8]->Draw("SAME");
//   avhisto100[9]->Draw("SAME");
//   avhisto100[10]->Draw("SAME");

// TLegend *TeV100 = new TLegend(0.6,0.7,0.89,0.89);                                240
// TeV100->AddEntry(avhisto100[1], "0m-1m (from shower start)", "l");
// TeV100->AddEntry(avhisto100[2], "1m-2m", "l");
// TeV100->AddEntry(avhisto100[3], "2m-3m", "l");
// TeV100->AddEntry(avhisto100[4], "3m-4m", "l");
// TeV100->AddEntry(avhisto100[5], "4m-5m", "l");
// TeV100->AddEntry(avhisto100[6], "5m-6m (from shower start)", "l");
// TeV100->AddEntry(avhisto100[7], "6m-7m", "l");
// TeV100->AddEntry(avhisto100[8], "7m-8m", "l");
// TeV100->AddEntry(avhisto100[9], "8m-9m", "l");
// TeV100->AddEntry(avhisto100[10], ">9m", "l");                                    250

// TeV100->Draw();



      //20TeV
      //——
//       avhisto20[5]->Draw("SAME");
//       avhisto20[1]->Draw("SAME");
//       avhisto20[2]->Draw("SAME");                                                260
//       avhisto20[3]->Draw("SAME");
//       avhisto20[4]->Draw("SAME");
//       avhisto20[6]->Draw("SAME");
//       avhisto20[7]->Draw("SAME");
//       avhisto20[8]->Draw("SAME");
//       avhisto20[9]->Draw("SAME");
//       avhisto20[10]->Draw("SAME");

// TLegend *TeV20 = new TLegend(0.6,0.7,0.89,0.89);
// TeV20->AddEntry(avhisto20[1], "0m-1m (from shower start)", "l");                 270
// TeV20->AddEntry(avhisto20[2], "1m-2m", "l");
// TeV20->AddEntry(avhisto20[3], "2m-3m", "l");
```

```
//  TeV20->AddEntry(avhisto20[4], "3m-4m", "l");
//  TeV20->AddEntry(avhisto20[5], "4m-5m", "l");
//  TeV20->AddEntry(avhisto20[6], "5m-6m", "l");
//  TeV20->AddEntry(avhisto20[7], "6m-7m", "l");
//  TeV20->AddEntry(avhisto20[8], "7m-8m", "l");
//  TeV20->AddEntry(avhisto20[9], "8m-9m", "l");
//  TeV20->AddEntry(avhisto20[10], ">9m", "l");
                                                                                    280
//  TeV20->Draw();


  //10TeV
  //=====
     avhisto10[4]->Draw("SAME");
//     avhisto10[1]->Draw("SAME");
//     avhisto10[2]->Draw("SAME");
//     avhisto10[3]->Draw("SAME");
//     avhisto10[4]->Draw("SAME");                                                  290
//     avhisto10[6]->Draw("SAME");
//     avhisto10[7]->Draw("SAME");
//     avhisto10[8]->Draw("SAME");
//     avhisto10[9]->Draw("SAME");
//     avhisto10[10]->Draw("SAME");

//  TLegend *TeV10 = new TLegend(0.6,0.7,0.89,0.89);
//  TeV10->AddEntry(avhisto10[1], "0m-1m (from shower start)", "l");
//  TeV10->AddEntry(avhisto10[2], "1m-2m", "l");
//  TeV10->AddEntry(avhisto10[3], "2m-3m", "l");                                    300
//  TeV10->AddEntry(avhisto10[4], "3m-4m", "l");
//  TeV10->AddEntry(avhisto10[5], "4m-5m", "l");
//  TeV10->AddEntry(avhisto10[6], "5m-6m", "l");
//  TeV10->AddEntry(avhisto10[7], "6m-7m", "l");
//  TeV10->AddEntry(avhisto10[8], "7m-8m", "l");
//  TeV10->AddEntry(avhisto10[9], "8m-9m", "l");
//  TeV10->AddEntry(avhisto10[10], ">9m", "l");

//  TeV10->Draw();
                                                                                    310

     //1TeV
     //=====
  avhisto1[4]->Draw("SAME");
//   avhisto1[1]->Draw("SAME");
//   avhisto1[2]->Draw("SAME");
//   avhisto1[3]->Draw("SAME");
//   avhisto1[4]->Draw("SAME");
//   avhisto1[6]->Draw("SAME");
//   avhisto1[7]->Draw("SAME");                                                     320
//   avhisto1[8]->Draw("SAME");
//   avhisto1[9]->Draw("SAME");
//   avhisto1[10]->Draw("SAME");

//  TLegend *TeV1 = new TLegend(0.6,0.7,0.89,0.89);
//  TeV1->AddEntry(avhisto1[1], "0m-1m (from shower start)", "l");
//  TeV1->AddEntry(avhisto1[2], "1m-2m", "l");
//  TeV1->AddEntry(avhisto1[3], "2m-3m", "l");
//  TeV1->AddEntry(avhisto1[4], "3m-4m", "l");
//  TeV1->AddEntry(avhisto1[5], "4m-5m", "l");                                      330
```

```
//  TeV1->AddEntry(avhisto1[6], "5m-6m", "l");
//  TeV1->AddEntry(avhisto1[7], "6m-7m", "l");
//  TeV1->AddEntry(avhisto1[8], "7m-8m", "l");
//  TeV1->AddEntry(avhisto1[9], "8m-9m", "l");
//  TeV1->AddEntry(avhisto1[10], ">9m", "l");

//  TeV1->Draw();


  TLegend *slice5 = new TLegend(0.6,0.7,0.89,0.89);                    340
  slice5->AddEntry(avhisto1[4], "1TeV", "l");
  slice5->AddEntry(avhisto10[4], "10TeV", "l");
  slice5->AddEntry(avhisto20[4], "20TeV", "l");
  slice5->AddEntry(avhisto100[4], "100TeV", "l");

  slice5->Draw();
}


                                                                       350
```

---

## B.12 dedphicomprevised.C

---

```
//=============================================================//
//Created by Simon Bevan                                       //
//29/11/03                                                     //
//                                                             //
//Revised 16/11/03 to tidy code                                //
//                                                             //
//Root macro for displaying dedphi                             //
//from output of graphfitter()                                 //
//=============================================================//
                                                                10
void dedphicomprevised() {

  Double_t M_PI = 3.14159265358979323846;

  //===========================================================

  //Set up for root
  //===============

  // Clean up :                                                  20
  // ===========
  gROOT->Reset();

  // No stats :
  // ===========
  gStyle->SetOptStat(0);
  gROOT->SetStyle("Plain");

  c1 = new TCanvas("c1","neutrinosim",200,10,700,500);
                                                                30
  c1->SetFillColor(0);
  c1->SetGridx();
```

```
c1->SetGridy();

//=============================================================================

//Loop through all files creating histograms
//-----------------------

TFile *pulse[26];
TH1F *histo[26];

Int_t energy = 0;
Int_t j = 0;
Int_t pulsenum = 0;
Int_t histonum = 0;

for(int i = 1; i<=20 ; i++){//

  if((i>=1)&&(i<=5)){
    energy = 1;
    j=i;
  }
  if((i>=6)&&(i<=10)){
    energy = 5;
    j = i-5;
  }
  if((i>=11)&&(i<=15)){
    energy = 10;
    j = i-10;
  }
  if((i>=16)&&(i<=20)){
    energy = 20;
    j = i-15;
  }

  Char_t filename[100];
  std::sprintf(filename, "%dTeV_pion_seawater_out_%d.root",energy,j);
  Char_t filenamekey[100];
  std::sprintf(filenamekey, "%dTeV_pion_seawater_%d_dedphi",energy,j);

  pulse[i] = new TFile(filename);
  histo[i] = (TH1F*) pulse[i]->Get(filenamekey);
}

//100TeV have different key name, so have to make implicitly
//----------------------
pulse[21] = new TFile("100TeV_pion_seawater_out_1.root");
histo[21] = (TH1F*) pulse[21]->Get("100TeV_pion_seawater_1_dedphi");
pulse[22] = new TFile("100TeV_pion_seawater_out_2.root");
histo[22] = (TH1F*) pulse[22]->Get("100TeV_pion_seawater_2_dedphi");
pulse[23] = new TFile("100TeV_pion_seawater_out_3.root");
histo[23] = (TH1F*) pulse[23]->Get("100TeV_pion_seawater_3_dedphi");
pulse[24] = new TFile("100TeV_pion_seawater_out_4.root");
histo[24] = (TH1F*) pulse[24]->Get("100TeV_pion_seawater_4_dedphi");
pulse[25] = new TFile("100TeV_pion_seawater_out_5.root");
histo[25] = (TH1F*) pulse[25]->Get("100TeV_pion_seawater_5_dedphi");

for(int i = 1; i<=25; i++){
```

```
  //set line styles for all pulses
  //————————————
  histo[i]->SetLineWidth(1.0);
  histo[i]->SetLineStyle(1);
}

//=========================================================================

//Want average graph for each energy
//————————————                                                          100

//Creat some histograms
//—————————

TH1F* TeV1_avsignal = new TH1F("Average_Pulse_1TeV","Average_Pulse_1TeV",100,-1.0*M_PI,1.0*M_PI);
TH1F* TeV5_avsignal = new TH1F("Average_Pulse_5TeV","Average_Pulse_5TeV",100,-1.0*M_PI,1.0*M_PI);
TH1F* TeV10_avsignal = new TH1F("Average_Pulse_10TeV","Average_Pulse_10TeV",100,-1.0*M_PI,1.0*M_PI);
TH1F* TeV20_avsignal = new TH1F("Average_Pulse_20TeV","Average_Pulse_20TeV",100,-1.0*M_PI,1.0*M_PI);
TH1F* TeV100_avsignal = new TH1F("Average_Pulse_100TeV","Average_Pulse_100TeV",100,-1.0*M_PI,1.0*M_PI);
                                                                        110
for(int i = 1; i <=5; i ++){
  TeV1_avsignal->Add(histo[i],0.2);
}
for(int i = 6; i <=10; i ++){
  TeV5_avsignal->Add(histo[i],0.2);
}
for(int i = 11; i <=15; i ++){
  TeV10_avsignal->Add(histo[i],0.2);
}
for(int i = 16; i <=20; i ++){                                          120
  TeV20_avsignal->Add(histo[i],0.2);
}
for(int i = 21; i <=25; i ++){
  TeV100_avsignal->Add(histo[i],0.2);
}

//Set line styles
//———————-
TeV1_avsignal->SetLineStyle(1);
TeV1_avsignal->SetLineWidth(2.0);                                       130

TeV5_avsignal->SetLineStyle(1);
TeV5_avsignal->SetLineWidth(2.0);

TeV10_avsignal->SetLineStyle(1);
TeV10_avsignal->SetLineWidth(2.0);

TeV20_avsignal->SetLineStyle(1);
TeV20_avsignal->SetLineWidth(2.0);
                                                                        140
TeV100_avsignal->SetLineStyle(1);
TeV100_avsignal->SetLineWidth(2.0);

//=========================================================================

//Setting line colours (set individually for obvious reasons)
//————————————————————————-
```

```
//1TeV Pulse
//----
histo[1]->SetLineColor(1);
histo[2]->SetLineColor(1);
histo[3]->SetLineColor(1);
histo[4]->SetLineColor(1);
histo[5]->SetLineColor(1);

//5TeV Pulses
//-----
histo[6]->SetLineColor(2);
histo[7]->SetLineColor(2);
histo[8]->SetLineColor(2);
histo[9]->SetLineColor(2);
histo[10]->SetLineColor(2);

//10TeV Pulses
//-----
histo[11]->SetLineColor(3);
histo[12]->SetLineColor(3);
histo[13]->SetLineColor(3);
histo[14]->SetLineColor(3);
histo[15]->SetLineColor(3);

//20TeV Pulses
//-----
histo[16]->SetLineColor(4);
histo[17]->SetLineColor(4);
histo[18]->SetLineColor(4);
histo[19]->SetLineColor(4);
histo[20]->SetLineColor(4);

//100TeV Pulses
//-----
histo[21]->SetLineColor(6);
histo[22]->SetLineColor(6);
histo[23]->SetLineColor(6);
histo[24]->SetLineColor(6);
histo[25]->SetLineColor(6);

//Average Pulses
//-----
TeV1_avsignal->SetLineColor(1);
TeV5_avsignal->SetLineColor(2);
TeV10_avsignal->SetLineColor(3);
TeV20_avsignal->SetLineColor(4);
TeV100_avsignal->SetLineColor(5);

//=============================================================================

//set axis (set for hist016, as biggest graph, and want axis to be set from this)
//-----------------------------------------

histo[21]->SetTitle("");
histo[21]->GetXaxis()->SetTitle("phi / radians");
histo[21]->GetXaxis()->CenterTitle();
histo[21]->GetXaxis()->SetTitleOffset(1.0);
histo[21]->GetYaxis()->SetTitle("energy / MeV");
```

```
histo[21]->GetYaxis()->SetTitleFont(12);
histo[21]->GetYaxis()->SetTitleSize(0.06);
histo[21]->GetYaxis()->SetTitleOffset(1.0);
histo[21]->GetYaxis()->CenterTitle();                                                    210

//=========================================================================================

//Drawing the Pulses(again done separately for ease of comparrison
//-------------------------

//100TeV pulses (drawn first to set axis correctly)
//-------------------
//  histo[21]->Draw();
//  histo[22]->Draw("Same");                                                             220
//  histo[23]->Draw("Same");
//  histo[24]->Draw("Same");
//  histo[25]->Draw("Same");

//  //1TeV pulses
//  //-----
//  histo[1]->Draw("Same");
//  histo[2]->Draw("Same");
//  histo[3]->Draw("Same");
//  histo[4]->Draw("Same");                                                              230
//  histo[5]->Draw("Same");

//  //5TeV pulses
//  //----
//  histo[6]->Draw("Same");
//  histo[7]->Draw("Same");
//  histo[8]->Draw("Same");
//  histo[9]->Draw("Same");
//  histo[10]->Draw("Same");
                                                                                       240
//  //10TeV pulses
//  //-----
//  histo[11]->Draw("Same");
//  histo[12]->Draw("Same");
//  histo[13]->Draw("Same");
//  histo[14]->Draw("Same");
//  histo[15]->Draw("Same");

//  //20TeV pulses
//  //-----                                                                            250
//  histo[16]->Draw("Same");
//  histo[17]->Draw("Same");
//  histo[18]->Draw("Same");
//  histo[19]->Draw("Same");
//  histo[20]->Draw("Same");

//  TLegend *dedrleg = new TLegend(0.6,0.7,0.89,0.89);
//  dedrleg->AddEntry(histo[1], "1TeV", "l");
//  dedrleg->AddEntry(histo[6], "5TeV", "l");
//  dedrleg->AddEntry(histo[11], "10TeV", "l");                                          260
//  dedrleg->AddEntry(histo[16], "20TeV", "l");
//  dedrleg->AddEntry(histo[21], "100TeV", "l");

//  dedrleg->Draw();
```

```
//set axis (set for hist016, as biggest graph, and want axis to be set from this)
//————————————————————————————

TeV100_avsignal->SetTitle("");
TeV100_avsignal->GetXaxis()->SetTitle("Phi / radians");
TeV100_avsignal->GetXaxis()->CenterTitle();
TeV100_avsignal->GetXaxis()->SetTitleOffset(1.0);
TeV100_avsignal->GetYaxis()->SetTitle("energy / MeV");
TeV100_avsignal->GetYaxis()->SetTitleFont(12);
TeV100_avsignal->GetYaxis()->SetTitleSize(0.06);
TeV100_avsignal->GetYaxis()->SetTitleOffset(1.0);
TeV100_avsignal->GetYaxis()->CenterTitle();

//Average puse for 1TeV
//————————
TeV100_avsignal->Draw();
TeV1_avsignal->Draw("Same");
TeV5_avsignal->Draw("Same");
TeV10_avsignal->Draw("Same");
TeV20_avsignal->Draw("Same");

TLegend *dedrlegfit = new TLegend(0.6,0.7,0.89,0.89);
dedrlegfit->AddEntry(TeV1_avsignal, "1TeV", "l");
dedrlegfit->AddEntry(TeV5_avsignal, "5TeV", "l");
dedrlegfit->AddEntry(TeV10_avsignal, "10TeV", "l");
dedrlegfit->AddEntry(TeV20_avsignal, "20TeV", "l");
dedrlegfit->AddEntry(TeV100_avsignal, "100TeV", "l");

dedrlegfit->Draw();

//=============================================================================

    }
```

## B.13 pulsecomprevised.C

```
//=========================================================//
//Created by Simon Bevan                                   //
//30/10/03                                                 //
//                                                         //
//Root macro for displaying and pressure pulses            //
//from output of neutrinosim_geant                         //
//                                                         //
//Revised 19/02/04                                         //
//Revised code to make easier to use                       //
//                                                         //
//Histos 1-5 - 1TeV                                        //
//Histos 5-10 - 5TeV                                       //
//Histos 11-15 - 10TeV                                     //
//Histos 16-20 - 20TeV                                     //
//Histos 21-25 - 100TeV                                    //
//Histos 26-28 - 1TeV electrons                            //
//Histos 29-32 - 1TeV freshwater                           //
//Histos 33-47 - cores                                     //
//=========================================================//
```

```
#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"
#include "TCanvas.h"

void pulsecomprevised() {

  // Clean up :
  // ==========
  gROOT->Reset();

  // No stats :
  // ==========
  gStyle->SetOptStat(0);
  gROOT->SetStyle("Plain");

  c1 = new TCanvas("c1","neutrinosim",200,10,700,500);

  c1->SetFillColor(0);
  c1->SetGridx();
  c1->SetGridy();

  //=========================================================================

  //Loop through all hadronic files extracting histograms
  //----------------------

  TFile *pulse[49];
  TH1F *histo[49];

  for(int i = 1; i<=20 ; i++){

    Int_t energy = 0;
    Int_t j = 0;

    if((i>=1)&&(i<=5)){
      energy = 1;
      j= i;
    }
    if((i>=6)&&(i<=10)){
      energy = 5;
      j = i-5;
    }
    if((i>=11)&&(i<=15)){
      energy = 10;
      j = i-10;
    }
    if((i>=16)&&(i<=20)){
      energy = 20;
      j = i - 15;
```

```
  }

  Char_t filename[100];
  std::sprintf(filename, "%dTeV_pion_seawater_%d_out.root",energy,j);
  Char_t filenamekey[100];
  std::sprintf(filenamekey, "angle 0.0");

  pulse[i] = new TFile(filename);
  histo[i] = (TH1F*) pulse[i]->Get(filenamekey);

  pulse[4] = new TFile("1TeV_pion_seawater_4_out.root");
  histo[4] = (TH1F*) pulse[4]->Get("angle -0.0");

}

//100TeV have different key name, so have to make implicitly
//--------------------------------
pulse[21] = new TFile("100TeV_pion_seawater_1_out.root");
histo[21] = (TH1F*) pulse[21]->Get("angle 0.0_total");
pulse[22] = new TFile("100TeV_pion_seawater_2_out.root");
histo[22] = (TH1F*) pulse[22]->Get("angle 0.0_total");
pulse[23] = new TFile("100TeV_pion_seawater_3_out.root");
histo[23] = (TH1F*) pulse[23]->Get("angle 0.0_total");
pulse[24] = new TFile("100TeV_pion_seawater_4_out.root");
histo[24] = (TH1F*) pulse[24]->Get("angle 0.0_total");
pulse[25] = new TFile("100TeV_pion_seawater_5_out.root");
histo[25] = (TH1F*) pulse[25]->Get("angle 0.0_total");

//electromagnetic pulse
//--------
pulse[26] = new TFile("1TeV_electron_seawater_1_out.root");
histo[26] = (TH1F*) pulse[26]->Get("angle -0.0_total");
pulse[27] = new TFile("1TeV_electron_seawater_2_out.root");
histo[27] = (TH1F*) pulse[27]->Get("angle -0.0_total");
pulse[28] = new TFile("1TeV_electron_seawater_3_out.root");
histo[28] = (TH1F*) pulse[28]->Get("angle -0.0_total");

//freshwater pulses
//------
pulse[29] = new TFile("1TeV_pion_freshwater_1_out.root");
histo[29] = (TH1F*) pulse[29]->Get("angle 0.0");
pulse[30] = new TFile("1TeV_pion_freshwater_2_out.root");
histo[30] = (TH1F*) pulse[30]->Get("angle 0.0");
pulse[31] = new TFile("1TeV_pion_freshwater_3_out.root");
histo[31] = (TH1F*) pulse[31]->Get("angle 0.0");
pulse[32] = new TFile("1TeV_pion_freshwater_4_out.root");
histo[32] = (TH1F*) pulse[32]->Get("angle 0.0");


//loop through all core files extracting histos
//----------------------

for(int g = 33; g<=42 ; g++){

  Int_t energy = 0;
  Int_t j = 0;

  if((g>=33)&&(g<=37)){
```

```
    energy = 1;
    j= g-32;
  }
  if((g>=38)&&(g<=42)){
    energy = 10;
    j = g-37;
  }
  if((g>=43)&&(g<=47)){
    energy = 20;
    j = g-42;
  }

  Char_t filename[100];
  std::sprintf(filename, "%dTeV_pion_seawater_core_%d_out.root",energy,j);
  Char_t filenamekey[100];
  std::sprintf(filenamekey, "angle 0.0");

  pulse[g] = new TFile(filename);
  histo[g] = (TH1F*) pulse[g]->Get(filenamekey);

}

histo[43] = histo[16];
histo[44] = histo[17];
histo[45] = histo[18];
histo[46] = histo[19];
histo[47] = histo[20];

//=========================================================================


//set line styles for all pulses
//-------------
for(int k =1; k<=47; k++){
  histo[k]->SetLineWidth(1.0);
  histo[k]->SetLineStyle(1);
}



//Want average graph for each energy
//--------------

//Find time window
//-------
double time_window_max = (1000/1505.19)+0.0002;
double time_window_min = (1000/1505.19)-0.0002;

//Create histograms
//---------
TH1F* TeV1_avsignal = new TH1F("Average_Pulse_1TeV","Average_Pulse_1TeV",512,time_window_min,time_window_max);
TH1F* TeV5_avsignal = new TH1F("Average_Pulse_5TeV","Average_Pulse_5TeV",512,time_window_min,time_window_max);
TH1F* TeV10_avsignal = new TH1F("Average_Pulse_10TeV","Average_Pulse_10TeV",512,time_window_min,time_window_max);
TH1F* TeV20_avsignal = new TH1F("Average_Pulse_20TeV","Average_Pulse_20TeV",512,time_window_min,time_window_max);
TH1F* TeV100_avsignal = new TH1F("Average_Pulse_100TeV","Average_Pulse_100TeV",512,time_window_min,time_window_m
TH1F* TeV1_elec_avsignal = new TH1F("Average_Pulse_elec_1TeV","Average_Pulse_elec_1TeV",512,time_window_min,time_w
TH1F* TeV1_fresh_avsignal = new TH1F("Average_Pulse_fresh_1TeV","Average_Pulse_fresh_1TeV",512,time_window_min,tim
TH1F* TeV1_core_avsignal = new TH1F("Average_Pulse_core_1TeV","Average_Pulse_core_1TeV",512,time_window_min,time_w
```

```
TH1F* TeV10_core_avsignal = new TH1F("Average_Pulse_core_10TeV","Average_Pulse_core_10TeV",512,time_window_min,tim
TH1F* TeV20_core_avsignal = new TH1F("Average_Pulse_core_20TeV","Average_Pulse_core_20TeV",512,time_window_min,tim

//average histograms for particuar energy
//-----------------
for(int i = 1; i <=5; i ++){
  TeV1_avsignal->Add(histo[i],0.2);
}
for(int i = 6; i <=10; i ++){
  TeV5_avsignal->Add(histo[i],0.2);
}
for(int i = 11; i <=15; i ++){
  TeV10_avsignal->Add(histo[i],0.2);
}
for(int i = 16; i <=20; i ++){
  TeV20_avsignal->Add(histo[i],0.2);
}
for(int i = 21; i <=25; i ++){
  TeV100_avsignal->Add(histo[i],0.2);
}
for(int i = 26; i <=28; i ++){
  TeV1_elec_avsignal->Add(histo[i],0.333);
}
for(int i = 29; i <=32; i ++){
  TeV1_fresh_avsignal->Add(histo[i],0.25);
}
for(int i = 33; i <=37; i ++){
  TeV1_core_avsignal->Add(histo[i],0.2);
}
for(int i = 38; i <=42; i ++){
  TeV10_core_avsignal->Add(histo[i],0.2);
}
for(int i = 43; i <=47; i ++){
  TeV20_core_avsignal->Add(histo[i],0.2);
}

//Set line styles
//------
TeV1_avsignal->SetLineStyle(1);
TeV1_avsignal->SetLineWidth(1.0);

TeV5_avsignal->SetLineStyle(1);
TeV5_avsignal->SetLineWidth(1.0);

TeV10_avsignal->SetLineStyle(1);
TeV10_avsignal->SetLineWidth(1.0);

TeV20_avsignal->SetLineStyle(1);
TeV20_avsignal->SetLineWidth(1.0);

TeV100_avsignal->SetLineStyle(1);
TeV100_avsignal->SetLineWidth(1.0);

TeV1_elec_avsignal->SetLineStyle(1);
TeV1_elec_avsignal->SetLineWidth(1.0);

TeV1_fresh_avsignal->SetLineStyle(1);
TeV1_fresh_avsignal->SetLineWidth(1.0);
```

```
TeV1_core_avsignal->SetLineStyle(2);
TeV1_core_avsignal->SetLineWidth(1.0);

TeV10_core_avsignal->SetLineStyle(2);
TeV10_core_avsignal->SetLineWidth(1.0);

TeV20_core_avsignal->SetLineStyle(2);
TeV20_core_avsignal->SetLineWidth(1.0);                                    260

//=============================================================================

//Setting line colours (set individually for obvious reasons)
//--------------------

//1TeV Pulse
//----
histo[1]->SetLineColor(1);
histo[2]->SetLineColor(2);                                                 270
histo[3]->SetLineColor(3);
histo[4]->SetLineColor(4);
histo[5]->SetLineColor(5);

//5TeV Pulses
//-----
histo[6]->SetLineColor(1);
histo[7]->SetLineColor(2);
histo[8]->SetLineColor(3);
histo[9]->SetLineColor(4);                                                 280
histo[10]->SetLineColor(5);

//10TeV Pulses
//-----
histo[11]->SetLineColor(1);
histo[12]->SetLineColor(2);
histo[13]->SetLineColor(3);
histo[14]->SetLineColor(4);
histo[15]->SetLineColor(5);
                                                                           290
//20TeV Pulses
//-----
histo[16]->SetLineColor(1);
histo[17]->SetLineColor(2);
histo[18]->SetLineColor(3);
histo[19]->SetLineColor(4);
histo[20]->SetLineColor(5);

//100TeV Pulses
//-----                                                                    300
histo[21]->SetLineColor(5);
histo[22]->SetLineColor(2);
histo[23]->SetLineColor(3);
histo[24]->SetLineColor(4);
histo[25]->SetLineColor(1);

//1TeV electro Pulses
//------
histo[26]->SetLineColor(1);
```

```
histo[27]->SetLineColor(2);
histo[28]->SetLineColor(3);

//1TeV fresh water Pulses
//----------

histo[29]->SetLineColor(5);
histo[30]->SetLineColor(2);
histo[31]->SetLineColor(3);
histo[32]->SetLineColor(4);

//1TeV core Pulse
//------
histo[33]->SetLineColor(1);
histo[34]->SetLineColor(2);
histo[35]->SetLineColor(3);
histo[36]->SetLineColor(4);
histo[37]->SetLineColor(5);

//10TeV core Pulses
//-------
histo[38]->SetLineColor(1);
histo[39]->SetLineColor(2);
histo[40]->SetLineColor(3);
histo[41]->SetLineColor(4);
histo[42]->SetLineColor(5);

//20TeV core Pulses
//-------
histo[43]->SetLineColor(1);
histo[44]->SetLineColor(2);
histo[45]->SetLineColor(3);
histo[46]->SetLineColor(4);
histo[47]->SetLineColor(5);

//Average Pulses
//-----
TeV1_avsignal->SetLineColor(1);
TeV5_avsignal->SetLineColor(5);
TeV10_avsignal->SetLineColor(2);
TeV20_avsignal->SetLineColor(3);
TeV100_avsignal->SetLineColor(4);
TeV1_elec_avsignal->SetLineColor(2);
TeV1_fresh_avsignal->SetLineColor(3);
TeV1_core_avsignal->SetLineColor(4);
TeV10_core_avsignal->SetLineColor(5);
TeV20_core_avsignal->SetLineColor(6);

//=========================================================================

//Drawing the Pulses(again done separately so can turn on or off depending on what you want to look at)
//-----------------------------------------------------

//1TeV pulses
//----
histo[1]->SetTitle("");
histo[1]->GetXaxis()->SetTitle("Time/s");
histo[1]->GetXaxis()->CenterTitle();
```

```
  histo[1]->GetXaxis()->SetTitleOffset(1.2);
  histo[1]->GetYaxis()->SetTitle("Pressure/Pa");
  histo[1]->GetYaxis()->SetTitleFont(12);                                    370
  histo[1]->GetYaxis()->SetTitleSize(0.06);
  histo[1]->GetYaxis()->SetTitleOffset(0.8);
  histo[1]->GetYaxis()->CenterTitle();

 // histo[1]->Draw();
//  histo[2]->Draw("Same");
//  histo[3]->Draw("Same");
//  histo[4]->Draw("Same");
//  histo[5]->Draw("Same");
                                                                             380
  TLegend *leg1TeV = new TLegend(0.6,0.7,0.89,0.89);

  leg1TeV->AddEntry(histo[1], "1TeV_1", "l");
  leg1TeV->AddEntry(histo[2], "1TeV_2", "l");
  leg1TeV->AddEntry(histo[3], "1TeV_3", "l");
  leg1TeV->AddEntry(histo[4], "1TeV_4", "l");
  leg1TeV->AddEntry(histo[5], "1TeV_5", "l");

  // leg1TeV->Draw();
                                                                             390
  //5TeV pulses
  //----
  histo[6]->SetTitle("");
  histo[6]->GetXaxis()->SetTitle("Time/s");
  histo[6]->GetXaxis()->CenterTitle();
  histo[6]->GetXaxis()->SetTitleOffset(1.2);
  histo[6]->GetYaxis()->SetTitle("Pressure/Pa");
  histo[6]->GetYaxis()->SetTitleFont(12);
  histo[6]->GetYaxis()->SetTitleSize(0.06);
  histo[6]->GetYaxis()->SetTitleOffset(0.8);                                 400
  histo[6]->GetYaxis()->CenterTitle();


//  histo[6]->Draw();
//  histo[7]->Draw("Same");
//  histo[8]->Draw("Same");
//  histo[9]->Draw("Same");
//  histo[10]->Draw("Same");

  TLegend *leg5TeV = new TLegend(0.6,0.7,0.89,0.89);                         410
  leg5TeV->AddEntry(histo[6], "5TeV_1", "l");
  leg5TeV->AddEntry(histo[7], "5TeV_2", "l");
  leg5TeV->AddEntry(histo[8], "5TeV_3", "l");
  leg5TeV->AddEntry(histo[9], "5TeV_4", "l");
  leg5TeV->AddEntry(histo[10], "5TeV_5", "l");

  // leg5TeV->Draw();

  //10TeV pulses
  //-----                                                                    420

  histo[11]->SetTitle("");
  histo[11]->GetXaxis()->SetTitle("Time/s");
  histo[11]->GetXaxis()->CenterTitle();
  histo[11]->GetXaxis()->SetTitleOffset(1.2);
```

```
  histo[11]->GetYaxis()->SetTitle("Pressure/Pa");
  histo[11]->GetYaxis()->SetTitleFont(12);
  histo[11]->GetYaxis()->SetTitleSize(0.06);
  histo[11]->GetYaxis()->SetTitleOffset(0.8);
  histo[11]->GetYaxis()->CenterTitle();                                    430

//   histo[11]->Draw();
//   histo[12]->Draw("Same");
//   histo[13]->Draw("Same");
//   histo[14]->Draw("Same");
//   histo[15]->Draw("Same");


  TLegend *leg10TeV = new TLegend(0.6,0.7,0.89,0.89);
                                                                           440
  leg10TeV->AddEntry(histo[6], "10TeV_1", "l");
  leg10TeV->AddEntry(histo[7], "10TeV_2", "l");
  leg10TeV->AddEntry(histo[8], "10TeV_3", "l");
  leg10TeV->AddEntry(histo[9], "10TeV_4", "l");
  leg10TeV->AddEntry(histo[10], "10TeV_5", "l");

  // leg10TeV->Draw();

  //20TeV
  //--                                                                     450
  histo[16]->SetTitle("");
  histo[16]->GetXaxis()->SetTitle("Time/s");
  histo[16]->GetXaxis()->CenterTitle();
  histo[16]->GetXaxis()->SetTitleOffset(1.2);
  histo[16]->GetYaxis()->SetTitle("Pressure/Pa");
  histo[16]->GetYaxis()->SetTitleFont(12);
  histo[16]->GetYaxis()->SetTitleSize(0.06);
  histo[16]->GetYaxis()->SetTitleOffset(0.8);
  histo[16]->GetYaxis()->CenterTitle();
                                                                           460
 //  histo[16]->Draw();
//   histo[17]->Draw("Same");
//   histo[18]->Draw("Same");
//   histo[19]->Draw("Same");
//   histo[20]->Draw("Same");

  TLegend *leg20TeV = new TLegend(0.6,0.7,0.89,0.89);

  leg20TeV->AddEntry(histo[16], "20TeV_1", "l");
  leg20TeV->AddEntry(histo[17], "20TeV_2", "l");                           470
  leg20TeV->AddEntry(histo[18], "20TeV_3", "l");
  leg20TeV->AddEntry(histo[19], "20TeV_4", "l");
  leg20TeV->AddEntry(histo[20], "20TeV_5", "l");

  //leg20TeV->Draw();

  //100TeV
  //---
  histo[21]->SetTitle("");
  histo[21]->GetXaxis()->SetTitle("Time/s");                               480
  histo[21]->GetXaxis()->CenterTitle();
  histo[21]->GetXaxis()->SetTitleOffset(1.2);
  histo[21]->GetYaxis()->SetTitle("Pressure/Pa");
```

```
histo[21]->GetYaxis()->SetTitleFont(12);
histo[21]->GetYaxis()->SetTitleSize(0.06);
histo[21]->GetYaxis()->SetTitleOffset(0.8);
histo[21]->GetYaxis()->CenterTitle();

histo[21]->SetLineStyle(1);
histo[22]->SetLineStyle(1);                                                    490
histo[23]->SetLineStyle(1);
histo[24]->SetLineStyle(1);
histo[25]->SetLineStyle(1);

histo[21]->Draw();
histo[22]->Draw("Same");
histo[23]->Draw("Same");
histo[24]->Draw("Same");
histo[25]->Draw("Same");
                                                                               500
TLegend *leg100TeV = new TLegend(0.6,0.7,0.89,0.89);

leg100TeV->AddEntry(histo[21], "100TeV_1", "l");
leg100TeV->AddEntry(histo[22], "100TeV_2", "l");
leg100TeV->AddEntry(histo[23], "100TeV_3", "l");
leg100TeV->AddEntry(histo[24], "100TeV_4", "l");
leg100TeV->AddEntry(histo[25], "100TeV_5", "l");

leg100TeV->Draw();
                                                                               510
//1TeV electro
//-----
histo[26]->SetTitle("");
histo[26]->GetXaxis()->SetTitle("Time/s");
histo[26]->GetXaxis()->CenterTitle();
histo[26]->GetXaxis()->SetTitleOffset(1.2);
histo[26]->GetYaxis()->SetTitle("Pressure/Pa");
histo[26]->GetYaxis()->SetTitleFont(12);
histo[26]->GetYaxis()->SetTitleSize(0.06);
histo[26]->GetYaxis()->SetTitleOffset(0.8);                                    520
histo[26]->GetYaxis()->CenterTitle();

histo[26]->SetLineStyle(2);
histo[27]->SetLineStyle(3);
histo[28]->SetLineStyle(4);

// histo[26]->Draw();
//   histo[27]->Draw("Same");
//   histo[28]->Draw("Same");
                                                                               530
TLegend *leg1TeVelec = new TLegend(0.6,0.7,0.89,0.89);

leg1TeVelec->AddEntry(histo[26], "1TeV_electromag_1", "l");
leg1TeVelec->AddEntry(histo[27], "1TeV_electromag_2", "l");
leg1TeVelec->AddEntry(histo[28], "1TeV_electromag_3", "l");

// leg1TeVelec->Draw();

//1TeV freshwater
//------                                                                       540
histo[29]->SetTitle("");
```

```
  histo[29]->GetXaxis()->SetTitle("Time/s");
  histo[29]->GetXaxis()->CenterTitle();
  histo[29]->GetXaxis()->SetTitleOffset(1.2);
  histo[29]->GetYaxis()->SetTitle("Pressure/Pa");
  histo[29]->GetYaxis()->SetTitleFont(12);
  histo[29]->GetYaxis()->SetTitleSize(0.06);
  histo[29]->GetYaxis()->SetTitleOffset(0.8);
  histo[29]->GetYaxis()->CenterTitle();

  histo[29]->SetLineStyle(1);
  histo[30]->SetLineStyle(1);
  histo[31]->SetLineStyle(1);
  histo[32]->SetLineStyle(1);

 // histo[29]->Draw();
//  histo[30]->Draw("Same");
//  histo[31]->Draw("Same");
//  histo[32]->Draw("Same");

  TLegend *leg1TeVfresh = new TLegend(0.6,0.7,0.89,0.89);

  leg1TeVfresh->AddEntry(histo[29], "1TeV_fresh_water_1", "l");
  leg1TeVfresh->AddEntry(histo[30], "1TeV_fresh_water_2", "l");
  leg1TeVfresh->AddEntry(histo[31], "1TeV_fresh_water_3", "l");
  leg1TeVfresh->AddEntry(histo[32], "1TeV_fresh_water_4", "l");

  // leg1TeVfresh->Draw();

  //Average pulses
  //--------
  TeV100_avsignal->SetTitle("");
  TeV100_avsignal->GetXaxis()->SetTitle("Time/s");
  TeV100_avsignal->GetXaxis()->CenterTitle();
  TeV100_avsignal->GetXaxis()->SetTitleOffset(1.2);
  TeV100_avsignal->GetYaxis()->SetTitle("Pressure/Pa");
  TeV100_avsignal->GetYaxis()->SetTitleFont(12);
  TeV100_avsignal->GetYaxis()->SetTitleSize(0.06);
  TeV100_avsignal->GetYaxis()->SetTitleOffset(0.8);
  TeV100_avsignal->GetYaxis()->CenterTitle();

 // TeV100_avsignal->Draw();
//  TeV1_avsignal->Draw("Same");
//  TeV5_avsignal->Draw("Same");
//  TeV10_avsignal->Draw("Same");
//  TeV20_avsignal->Draw("Same");

  TLegend *av = new TLegend(0.6,0.7,0.89,0.89);

  av->AddEntry(TeV1_avsignal, "1TeV", "l");
  av->AddEntry(TeV5_avsignal, "5TeV", "l");
  av->AddEntry(TeV10_avsignal, "10TeV", "l");
  av->AddEntry(TeV20_avsignal, "20TeV", "l");
  av->AddEntry(TeV100_avsignal, "100TeV", "l");

  // av->Draw();

  //Comparrison of hadronic (fresh and salt water), and electomagnetic
  //----------------------
```

```
  TeV1_elec_avsignal->SetTitle("");
  TeV1_elec_avsignal->GetXaxis()->SetTitle("Time/s");
  TeV1_elec_avsignal->GetXaxis()->CenterTitle();
  TeV1_elec_avsignal->GetXaxis()->SetTitleOffset(1.2);
  TeV1_elec_avsignal->GetYaxis()->SetTitle("Pressure/Pa");
  TeV1_elec_avsignal->GetYaxis()->SetTitleFont(12);
  TeV1_elec_avsignal->GetYaxis()->SetTitleSize(0.06);
  TeV1_elec_avsignal->GetYaxis()->SetTitleOffset(0.8);
  TeV1_elec_avsignal->GetYaxis()->CenterTitle();

  TeV1_elec_avsignal->SetLineStyle(1);
  TeV1_fresh_avsignal->SetLineStyle(1);
  TeV1_avsignal->SetLineStyle(1);

 //  TeV1_elec_avsignal->Draw();
//   TeV1_fresh_avsignal->Draw("Same");
//   TeV1_avsignal->Draw("Same");

  TLegend *types = new TLegend(0.6,0.7,0.89,0.89);

  types->AddEntry(TeV1_elec_avsignal, "1TeV electromagentic pulse", "l");
  types->AddEntry(TeV1_fresh_avsignal, "1TeV hadronic (fresh water) pulse", "l");
  types->AddEntry(TeV1_avsignal, "1TeV hadronic (salt water) pulse", "l");

  //types->Draw();

  //Comparrison of cores vs whole shower
  //-------------------

  TeV20_core_avsignal->SetTitle("");
  TeV20_core_avsignal->GetXaxis()->SetTitle("Time/s");
  TeV20_core_avsignal->GetXaxis()->CenterTitle();
  TeV20_core_avsignal->GetXaxis()->SetTitleOffset(1.2);
  TeV20_core_avsignal->GetYaxis()->SetTitle("Pressure/Pa");
  TeV20_core_avsignal->GetYaxis()->SetTitleFont(12);
  TeV20_core_avsignal->GetYaxis()->SetTitleSize(0.06);
  TeV20_core_avsignal->GetYaxis()->SetTitleOffset(0.8);
  TeV20_core_avsignal->GetYaxis()->CenterTitle();

  TeV20_core_avsignal->SetLineStyle(2);
  TeV10_core_avsignal->SetLineStyle(2);
  TeV10_core_avsignal->SetLineStyle(2);
  TeV1_avsignal->SetLineStyle(1);
  TeV10_avsignal->SetLineStyle(1);
  TeV20_avsignal->SetLineStyle(1);

 //  TeV20_core_avsignal->Draw();
//   TeV10_core_avsignal->Draw("Same");
//   TeV1_core_avsignal->Draw("Same");
//   TeV20_avsignal->Draw("Same");
//   TeV10_avsignal->Draw("Same");
//   TeV1_avsignal->Draw("Same");

  TLegend *core = new TLegend(0.6,0.7,0.89,0.89);

  core->AddEntry(TeV1_avsignal, "1TeV pulse generated from whole shower", "l");
  core->AddEntry(TeV10_avsignal, "10TeV pulse generated from whole shower", "l");
```

```
  core->AddEntry(TeV20_avsignal, "20TeV pulse generated from whole shower", "l");
  core->AddEntry(TeV1_core_avsignal, "1TeV pulse generated from core of shower", "l");
  core->AddEntry(TeV10_core_avsignal, "10TeV pulse generated from core of shower", "l");
  core->AddEntry(TeV20_core_avsignal, "20TeV pulse generated from core of shower", "l");

  // core->Draw();

  //=============================================================================


  pulse[48] = new TFile("my_pulse.root");
  histo[48] = (TH1F*) pulse[21]->Get("pulse");

   TeV100_avsignal->Draw();
   histo[48]->Draw("SAME");
}
```

---

## B.14 pulse_dedr_dz_revised.C

---

```
//=============================================================//
//Created by Simon Bevan                                       //
//09/12/03                                                     //
//                                                             //
//Revised 01/03/04                                             //
//                                                             //
//Root macro for displaying the pressure pulses                //
//at varying z and comparing them                              //
//                                                             //
//=============================================================//

#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"
#include "TCanvas.h"

void pulse_dedr_dz_revised() {

  // Clean up :
  // ==========
  gROOT->Reset();

  // No stats :
  // ==========
  gStyle->SetOptStat(0);
  gROOT->SetStyle("Plain");

  c1 = new TCanvas("c1","neutrinosim",200,10,700,500);
```

```
c1->SetFillColor(0);
c1->SetGridx();
c1->SetGridy();
                                                                                40
//=============================================================================

//Loop through all hadronic files extracting histograms for slices
//--------------------------------------------------------------------------

//slices 1-15 20TeV 1
//slices 16-35 20TeV 2 etc until 20TeV 5
//-------------------

TFile *pulse[90];                                                               50
TH1F *histo[90];

int pulsenum = -2;
int extra = 0;
int extra2 = 0;

for(int i = 1; i<=5; i++){
  pulsenum = pulsenum +2;
    for(int j = 1; j<=13; j++){
                                                                                60
      pulsenum = pulsenum + 1;

      Char_t filename[100];
      std::sprintf(filename, "20Tev_pion_seawater_slices_%d_out.root",i);
      Char_t filenamekey[100];
      std::sprintf(filenamekey, "angle 0.0_%d",j);

      pulse[pulsenum] = new TFile(filename);
      histo[pulsenum] = (TH1F*) pulse[pulsenum]->Get(filenamekey);
                                                                                70
      extra = i*14+(i-1);
      extra2 = i*15;

      pulse[extra] = new TFile(filename);
      histo[extra] = (TH1F*) pulse[extra]->Get("angle 0.0_15");
      pulse[extra2] = new TFile(filename);
      histo[extra2] = (TH1F*) pulse[extra2]->Get("");
    }
}
                                                                                80
//total pulses
//-----
int j = 0;
for(int i = 76; i<=80; i++){
  j = i - 75;

  Char_t total[100];
  std::sprintf(total, "20Tev_pion_seawater_slices_%d_out.root",j);

  pulse[i] = new TFile(total);                                                  90
  histo[i] = (TH1F*) pulse[i]->Get("angle 0.0_total");
}

//set line styles for all pulses
```

```
//————————————
for(int k =1; k<=80; k++){

  histo[k]->SetLineWidth(1.0);
  histo[k]->SetLineStyle(1);
}

//Comparrison for total of slices and total graph
//————————————————————-

//Find time window
//———————-
double time_window_max = (1000/1505.19)+0.0002;
double time_window_min = (1000/1505.19)-0.0002;

//total of slices
//———————
TH1F* total_slices_20TeV_1 = new TH1F("Average_Pulseslic1","Average_Pulseslic1",512,time_window_min,time_window_ma
TH1F* total_slices_20TeV_2 = new TH1F("Average_Pulseslic2","Average_Pulseslic2",512,time_window_min,time_window_ma
TH1F* total_slices_20TeV_3 = new TH1F("Average_Pulseslic3","Average_Pulseslic3",512,time_window_min,time_window_ma
TH1F* total_slices_20TeV_4 = new TH1F("Average_Pulseslic4","Average_Pulseslic4",512,time_window_min,time_window_ma
TH1F* total_slices_20TeV_5 = new TH1F("Average_Pulseslic5","Average_Pulseslic5",512,time_window_min,time_window_ma

for(int i = 1; i<=15; i++){
  total_slices_20TeV_1->Add(histo[i],1);
}
for(int i = 16; i<=30; i++){
  total_slices_20TeV_2->Add(histo[i],1);
}
for(int i = 31; i<=45; i++){
  total_slices_20TeV_3->Add(histo[i],1);
}
for(int i = 46; i<=60; i++){
  total_slices_20TeV_4->Add(histo[i],1);
}
for(int i = 61; i<=75; i++){
  total_slices_20TeV_5->Add(histo[i],1);
}

total_slices_20TeV_1->SetLineWidth(1.0);
total_slices_20TeV_1->SetLineStyle(4);

total_slices_20TeV_2->SetLineWidth(1.0);
total_slices_20TeV_2->SetLineStyle(4);

total_slices_20TeV_3->SetLineWidth(1.0);
total_slices_20TeV_3->SetLineStyle(4);

total_slices_20TeV_4->SetLineWidth(1.0);
total_slices_20TeV_4->SetLineStyle(4);

total_slices_20TeV_5->SetLineWidth(1.0);
total_slices_20TeV_5->SetLineStyle(4);

//Average graphs
//———————

//Create histograms
```

```
//--------
TH1F* average_slices_20TeV_total = new TH1F("Average_pulsetot","Average_Pulsetot",512,time_window_min,time_window_n
TH1F* average_slices_20TeV_1 = new TH1F("Average_Pulse1","Average_Pulse1",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_2 = new TH1F("Average_Pulse2","Average_Pulse2",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_3 = new TH1F("Average_Pulse3","Average_Pulse3",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_4 = new TH1F("Average_Pulse4","Average_Pulse4",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_5 = new TH1F("Average_Pulse5","Average_Pulse5",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_6 = new TH1F("Average_Pulse6","Average_Pulse6",512,time_window_min,time_window_max); 160
TH1F* average_slices_20TeV_7 = new TH1F("Average_Pulse7","Average_Pulse7",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_8 = new TH1F("Average_Pulse8","Average_Pulse8",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_9 = new TH1F("Average_Pulse9","Average_Pulse9",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_10 = new TH1F("Average_Pulse10","Average_Pulse10",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_11 = new TH1F("Average_Pulse11","Average_Pulse11",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_12 = new TH1F("Average_Pulse12","Average_Pulse12",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_13 = new TH1F("Average_Pulse13","Average_Pulse13",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_14 = new TH1F("Average_Pulse14","Average_Pulse14",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_15 = new TH1F("Average_Pulse15","Average_Pulse15",512,time_window_min,time_window_max);
TH1F* average_slices_20TeV_slictotal = new TH1F("Average_Pulse_slictotal","Average_Pulseslic_total",512,time_window_

//average histograms for particuar energy
//----------------

for(int i = 1; i <=5; i++){
  average_slices_20TeV_slictotal->Add(total_slices_20TeV_1,0.2);
  average_slices_20TeV_slictotal->Add(total_slices_20TeV_2,0.2);
  average_slices_20TeV_slictotal->Add(total_slices_20TeV_3,0.2);
  average_slices_20TeV_slictotal->Add(total_slices_20TeV_4,0.2);
  average_slices_20TeV_slictotal->Add(total_slices_20TeV_5,0.2);                    180
}
for(int i = 76; i <=80; i++){
  average_slices_20TeV_total->Add(histo[i],0.2);
}
for(int i = 1; i <=75; i = i+15){
  average_slices_20TeV_1->Add(histo[i],0.2);
}
for(int i = 2; i <=75; i = i+15){
  average_slices_20TeV_2->Add(histo[i],0.2);
}                                                                                   190
for(int i = 3; i <=75; i = i+15){
  average_slices_20TeV_3->Add(histo[i],0.2);
}
for(int i = 4; i <=75; i = i+15){
  average_slices_20TeV_4->Add(histo[i],0.2);
}
for(int i = 5; i <=75; i = i+15){
  average_slices_20TeV_5->Add(histo[i],0.2);
}
for(int i = 6; i <=75; i = i+15){                                                   200
  average_slices_20TeV_6->Add(histo[i],0.2);
}
for(int i = 7; i <=75; i = i+15){
  average_slices_20TeV_7->Add(histo[i],0.2);
}
for(int i = 8; i <=75; i = i+15){
  average_slices_20TeV_8->Add(histo[i],0.2);
}
for(int i = 9; i <=75; i = i+15){
  average_slices_20TeV_9->Add(histo[i],0.2);                                        210
```

```
}
for(int i = 10; i <=75; i = i+15){
  average_slices_20TeV_10->Add(histo[i],0.2);
}
for(int i = 11; i <=75; i = i+15){
  average_slices_20TeV_11->Add(histo[i],0.2);
}
for(int i = 12; i <=75; i = i+15){
  average_slices_20TeV_12->Add(histo[i],0.2);
}
for(int i = 13; i <=75; i = i+15){
  average_slices_20TeV_13->Add(histo[i],0.2);
}
for(int i = 14; i <=75; i = i+15){
  average_slices_20TeV_14->Add(histo[i],0.2);
}
for(int i = 15; i <=75; i = i+15){
  average_slices_20TeV_15->Add(histo[i],0.2);
}

//Set line styles
//------
average_slices_20TeV_1->SetLineStyle(1);
average_slices_20TeV_1->SetLineWidth(1.0);

average_slices_20TeV_2->SetLineStyle(1);
average_slices_20TeV_2->SetLineWidth(1.0);

average_slices_20TeV_3->SetLineStyle(1);
average_slices_20TeV_3->SetLineWidth(1.0);

average_slices_20TeV_4->SetLineStyle(1);
average_slices_20TeV_4->SetLineWidth(1.0);

average_slices_20TeV_5->SetLineStyle(1);
average_slices_20TeV_5->SetLineWidth(1.0);

average_slices_20TeV_6->SetLineStyle(1);
average_slices_20TeV_6->SetLineWidth(1.0);

average_slices_20TeV_7->SetLineStyle(1);
average_slices_20TeV_7->SetLineWidth(1.0);

average_slices_20TeV_8->SetLineStyle(1);
average_slices_20TeV_8->SetLineWidth(1.0);

average_slices_20TeV_9->SetLineStyle(1);
average_slices_20TeV_9->SetLineWidth(1.0);

average_slices_20TeV_10->SetLineStyle(1);
average_slices_20TeV_10->SetLineWidth(1.0);

average_slices_20TeV_11->SetLineStyle(1);
average_slices_20TeV_11->SetLineWidth(1.0);

average_slices_20TeV_12->SetLineStyle(1);
average_slices_20TeV_12->SetLineWidth(1.0);
```

```
average_slices_20TeV_13->SetLineStyle(1);
average_slices_20TeV_13->SetLineWidth(1.0);

average_slices_20TeV_14->SetLineStyle(1);
average_slices_20TeV_14->SetLineWidth(1.0);

average_slices_20TeV_15->SetLineStyle(1);
average_slices_20TeV_15->SetLineWidth(1.0);

average_slices_20TeV_slictotal->SetLineStyle(4);
average_slices_20TeV_slictotal->SetLineWidth(1.0);

average_slices_20TeV_total->SetLineStyle(2);
average_slices_20TeV_total->SetLineWidth(1.0);

//============================================================================

average_slices_20TeV_total->SetLineColor(1);
average_slices_20TeV_1->SetLineColor(3);
average_slices_20TeV_2->SetLineColor(3);
average_slices_20TeV_3->SetLineColor(3);
average_slices_20TeV_4->SetLineColor(2);
average_slices_20TeV_5->SetLineColor(4);
average_slices_20TeV_6->SetLineColor(1);
average_slices_20TeV_7->SetLineColor(6);
average_slices_20TeV_8->SetLineColor(7);
average_slices_20TeV_9->SetLineColor(5);
average_slices_20TeV_10->SetLineColor(9);
average_slices_20TeV_11->SetLineColor(3);
average_slices_20TeV_12->SetLineColor(3);
average_slices_20TeV_13->SetLineColor(3);
average_slices_20TeV_14->SetLineColor(3);
average_slices_20TeV_15->SetLineColor(3);
average_slices_20TeV_slictotal->SetLineColor(4);


average_slices_20TeV_total->SetTitle("");
average_slices_20TeV_total->GetXaxis()->SetTitle("Time / s");
average_slices_20TeV_total->GetXaxis()->CenterTitle();
average_slices_20TeV_total->GetXaxis()->SetTitleOffset(1.0);
average_slices_20TeV_total->GetYaxis()->SetTitle("Pressure / Pa");
average_slices_20TeV_total->GetYaxis()->SetTitleFont(12);
average_slices_20TeV_total->GetYaxis()->SetTitleSize(0.06);
average_slices_20TeV_total->GetYaxis()->SetTitleOffset(1.0);
average_slices_20TeV_total->GetYaxis()->CenterTitle();

//Draw Slices
//==========

average_slices_20TeV_total->Draw();
average_slices_20TeV_1->Draw("SAME");
average_slices_20TeV_2->Draw("SAME");
average_slices_20TeV_3->Draw("SAME");
average_slices_20TeV_4->Draw("SAME");
average_slices_20TeV_5->Draw("SAME");
average_slices_20TeV_6->Draw("SAME");
average_slices_20TeV_7->Draw("SAME");
average_slices_20TeV_8->Draw("SAME");
```

```
  average_slices_20TeV_9->Draw("SAME");
  average_slices_20TeV_10->Draw("SAME");
  average_slices_20TeV_11->Draw("SAME");
  average_slices_20TeV_12->Draw("SAME");
  average_slices_20TeV_13->Draw("SAME");
  average_slices_20TeV_14->Draw("SAME");
  average_slices_20TeV_15->Draw("SAME");
  average_slices_20TeV_slictotal->Draw("SAME");

  TLegend *leg = new TLegend(0.6,0.7,0.89,0.89);

  leg->AddEntry(average_slices_20TeV_total, "total", "l");
  leg->AddEntry(average_slices_20TeV_slictotal, "slices total", "l");
  leg->AddEntry(average_slices_20TeV_4, "0m < z < 1m", "l");
  leg->AddEntry(average_slices_20TeV_5, "1m < z < 2m", "l");
  leg->AddEntry(average_slices_20TeV_6, "2m < z < 3m", "l");
  leg->AddEntry(average_slices_20TeV_7, "3m < z < 4m", "l");
  leg->AddEntry(average_slices_20TeV_8, "4m < z < 5m", "l");
  leg->AddEntry(average_slices_20TeV_9, "5m < z < 6m", "l");
  leg->AddEntry(average_slices_20TeV_10,"6m < z < 7m", "l");
  leg->AddEntry(average_slices_20TeV_11," z > 7m ", "l");

  leg->Draw();

}
```

## B.15 pulse_reproducer.C

```
// ------------------------------------------//
// Created by Simon Bevan                    //
// To create a paramatized pulse             //
// ------------------------------------------ //

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TH1F.h"
#include "TH2F.h"
#include "TRandom3.h"
#include "TVector3.h"

//relic of old code, but needed for dp.hh, doesn't slow code down, so left alone so old code still works
```

```
enum {
  uhe = 1,
  dump = 2
};

#include "dp.hh"
#include "randomEnergy.hh"                                                    30
#include "randomPosition.hh"
#include "scaledVectorAdd.hh"
#include "scaledPulse.hh"

#define M_PI 3.14159265358979323846

using std::cout;
using std::vector;

int main(int argc, char *argv[]) {                                            40

  //create a reconstructed pulse
  //------------

  //input variables
  //-------
  int     num_events = 1;
  int     seed       = 3;
  int     nint       = 10000;
  double temp     = 13;                                                       50
  double depth    = 300;
  double salinity  = 35 ;
  int no_detectors = 0;
  int no_detectors_temp = 0;
  double maxpt   = 0;
  double maxpres = 0;

  char output[100];
  sprintf(output,"simulated_pulse");
                                                                              60
  // Make output file :
  // --------
  TFile* my_Pulse = new TFile("my_pulse.root","RECREATE","my_pulse");
  TH1F energyCheck("energy","energy",100, 17, 21);

  // Temp, depth, salinity from input of neutrinosim
  //------------------------
  // coefficient of thermal expansion (bulk/volume expansivity) :
  double beta = alpha(temp, depth, salinity);
  // specific heat :                                                          70
  double cp = cpf(temp, depth, salinity);
  // attenuation coefficient :
  double omega = 2.5E10;
  // Speed of sound in water :
  double c   = vel(temp, depth, salinity);

  //=============================================================================

  // Type of event to generate :
  // ----------                                                               80
  int atype;
```

```
//UHE neutrino cascades in water :
atype = uhe;

// Use the same random number generator consistently throughout :
// ----------------------------
TRandom3 ranGen;
ranGen.SetSeed(seed);

//define maximum and minium energy for showers                                  90
//-------------------

double eMin = 1E17;
double eMax = 1E21;

//define z of maximum energy (from evaluating dedz eqn)
//----------------------.
float zmaxenergy = 5.22;

//loop over number of events                                                    100
//-----------
double eventcount = 0;

for (int event_no = 1; event_no <= num_events; ++event_no){

  //Define random enaergy
  //---------
  double neutrino_energy = 0;

  neutrino_energy = randomEnergy(eMin, eMax, &ranGen);                          110

  // Define hydrophone location:
  // ----------.
  double angle       = 0;
  double hy_radius   = 0;
  double hy_dir      = 0;
  double x_sensor    = 0;
  double y_sensor    = 0;
  double z_sensor    = 0;
  double x_cascade   = 0;                                                       120
  double y_cascade   = 0;
  double z_cascade   = 0;
  double theta_cascade = 0;
  double phi_cascade = 0;
  double x_dir       = 0;
  double y_dir       = 0;
  double z_dir       = 0;

  //-------------------------------------------------------------------
                                                                                130
  //Position and energy values
  //----------
  hy_radius = 1000;
  neutrino_energy = 20*1E12;
  angle = 0.0;

  //-------------------------------------------------------------------

  //exclude the no hopers
```

```
//————————                                                                    140
if ((angle < 6.0 && angle > −6.0) || (hy_radius < 1000.0) ){

  eventcount = eventcount +1;

  //Transform to shower-centric co-ordinates
  //————————————————
  x_sensor = hy_radius*cos(M_PI*angle/180.0);
  y_sensor = 0.0;
  z_sensor = hy_radius*sin(M_PI*angle/180.0);
                                                                              150
  // Time-window for pressure pulse :
  // ———————————————
  double start_time = (hy_radius/c) − 0.0002;
  double end_time = (hy_radius/c) + 0.0002;
  int num_times    = 512;
  vector<double> times(num_times);

  for (int i = 0; i < num_times; ++i) {
    times[i] = start_time + (i*(end_time−start_time)/num_times);
  }                                                                           160

  // Define integrated pressure pulse arrays:
  // ————————————
  vector<double> pressure(num_times);
  vector<double> pinc(num_times);

  // Setup Monte Carlo integration parameters (in SI units) :
  // ————————————————————————

  double energy_density_sum = 0.0;                                            170
  double zmax = 10.0;
  double rmax = 0.3;

  for(int j=0; j <= nint; j++){

    double z0 = 0;
    double x0 = 0;
    double y0 = 0;
    double phi0 = 0;
    double r_ran = 0;                                                         180
    double z_ran = 0;

    // Generating shower :
    // ————————
    z_ran = (ranGen.Rndm())*zmax − 5.22;
    r_ran = (ranGen.Rndm())*rmax;
    phi0 = (ranGen.Rndm())*(2*M_PI);

    //change to cartesian co-ords
    //————————————                                                            190
    x0 = r_ran*cos(phi0);
    y0 = r_ran*sin(phi0);
    z0 = z_ran;

    // Calculate distance from measurement point :
    // ——————————————————-
    double r = pow((pow((x_sensor − x0),2) + pow((y_sensor − y0),2) + pow((z_sensor − z0),2)),0.5);
```

```
    //dedz calculator (from geant analysis)
    //————————————————
    double par1z = 9;
    double par2z = −1.75;
    double dedz1 = pow(fabs(z_ran+5.22),par1z)*exp(par2z*(fabs(z_ran+5.22)));

    //dedr calculator (from geant analysis)
    //————————————————
    double par1r =24.6511;
    double par4r = 9.56813;
    double r_ran_mm = r_ran*1000;
    double dedr1 = exp(−(r_ran_mm−par4r)/par1r);

    //energy density (in eV)
    //—————————
    double energy_density = dedz1*dedr1*1.0E12;
    energy_density_sum = energy_density + energy_density_sum;

    // Find the pressure pulse :
    // ————————————-
    dp(&times,&pinc,r,atype,temp,depth,salinity,beta,cp,omega,c);

    // Superpose it :
    // ——————
    scaledVectorAdd(&pressure,&pinc,energy_density);
  }

  //define a scaling number in J
  //————————————-
  double scaling = neutrino_energy*1.602E−19/energy_density_sum;

  // Store pressure pulse in a histogram :
  // ——————————————————-
  TH1F signal("pulse","acoustic pulse",num_times,start_time,end_time);
  for (int i = 0; i < num_times; ++i) {
    signal.SetBinContent(i,pressure[i]*scaling*3.3);

  }
  maxpres = 0;
  maxpres = signal.GetBinContent(signal.GetMaximumBin());
  maxpt = 0;
  maxpt = times[signal.GetMaximumBin()];

  my_Pulse−>cd();
  signal.Write();

 }
}

my_Pulse−>Close();
return 0;
    }
```

## B.16 neutrinosim.cc

```
// ---------------------------------------- //
// Dave Waters, 7/6/2003                    //
// Edited by Simon Bevan, 4/11/2003         //
// Edited to incorporate water_properties.hh and geant simulation data  //
//                                          //
// Edited by Simon Bevan, 4/12/2003         //
// Edited to incorporate test for shower shapes at varying z  //
//                                          //
// Edited 15/12/03 by Simon Bevan           //
// Edited to incorporate pulse reconstructor code  //
//                                          //
// Edited 16/12/03 by Simon Bevan           //
// All old code removed and placed in old_netrinosim_code folder  //
//                                          //
// Edited 28/01/04 by Simon Bevan           //
// All test code removed to leave only necessary code. Test code placed in  //
// old_netrinosim_code folder.              //
//                                          //
// C++ implementation of acoustic neutrino simulation  //
// Originally impelemented in Octave.       //
//                                          //
// Need to set ROOTSYS and LD_LIBRARY_PATH *** gcc versions required *** //
// Compile :                                //
// > g++ -I$ROOTSYS/include -c neutrinosim.cc -o neutrinosim.o  //
// Link :                                   //
// > g++ -o neutrinosim neutrinosim.o -Llib -lCore -lCint -lHist -lGraf  //
//     -lGraf3d -lTree -lMatrix -lRint -lm -ldl -L$ROOTSYS/lib  //
//     -lpthread -rdynamic                  //
// ---------------------------------------- //

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TH1F.h"
#include "TH2F.h"
#include "TRandom3.h"
#include "TVector3.h"

//relic of old code, but needed for dp.hh, doesn't slow code down, so left alone so old code still works
enum {
  uhe = 1,
  dump = 2
};

#include "dp.hh"
#include "randomEnergy.hh"
#include "randomPosition.hh"
```

```
#include "scaledVectorAdd.hh"
#include "scaledPulse.hh"


#define M_PI 3.14159265358979323846                                                    60

using std::cout;
using std::vector;

int main(int argc, char *argv[]) {


  int narg = argc-1;
  if (narg <8 || narg >8) {
    cout << "usage : neutrinosim <num-events> <seed> <nint> <temp> <depth> <salinity> <array input file> <out file>" <
    return 1;
  }


  //input variables
  //-------
  int     num_events = atoi(argv[1]);
  int     seed       = atoi(argv[2]);
  int     nint       = atoi(argv[3]);
  double temp     = atof(argv[4]);                                                     80
  double depth    = atof(argv[5]);
  double salinity  = atof(argv[6]);
  int no_detectors = 0;
  int no_detectors_temp = 0;
  double maxpt = 0;
  double maxpres = 0;

  char array_data[100];
  sprintf(array_data,argv[7]);
                                                                                       90
  char output[100];
  sprintf(output,argv[8]);

  // Make output file :
  // --------
  TFile* my_Pulse = new TFile("my_pulse.root","RECREATE","my_pulse");
  TH1F energyCheck("energy","energy",100, 17, 21);


  // WATER PROPERTIES - from water_properties.hh :                                     100
  // ==========================================

  // Temp, depth, salinity from input of neutrinosim
  //-------------------
  // coefficient of thermal expansion (bulk/volume expansivity) :
  double beta = alpha(temp, depth, salinity);
  // specific heat :
  double cp  = cpf(temp, depth, salinity);
  // attenuation coefficient :
  double omega = 2.5E10;                                                               110
  // Speed of sound in water :
  double c    = vel(temp, depth, salinity);
```

```
//=============================================================================

// printf("\n neutrinosim : \n");
// printf(" ============= \n");

// Type of event to generate :
// ---------------------------
int atype;
//UHE neutrino cascades in water :
atype = uhe;

// Use the same random number generator consistently throughout :
// ---------------------------------------------
// DO NOT USE TRandom2 : BAD.
TRandom3 ranGen;
ranGen.SetSeed(seed);

//check if the water properties seem realistic
//-------------------

ofstream outfile(output);

//outfile << "the temp of the water is    " << temp << endl;
//outfile << "the depth of the pulse is    " << depth << endl;
//outfile << "the salinity of the water is " << salinity << endl;
//outfile << "the speed of the pulse is    " << c << endl;
//outfile << "the coeff of theraml expan is " << beta << endl;
//outfile << "the specific heat cap is     " << cp << endl;
//outfile << "                             " << endl;

outfile << temp  << endl;
outfile << depth << endl;
outfile << salinity << endl;
outfile << c       << endl;
outfile << beta    << endl;
outfile << cp      << endl;
outfile << ""      << endl;


ifstream infile1(array_data);
if (infile1.fail()) {
     cout << "Array data file " << array_data << " NOT FOUND" << endl;
     return 1;
   }

while (!infile1.eof()) {

  double hy_locationx = 0;
  double hy_locationy = 0;
  double hy_locationz = 0;

     infile1 >> hy_locationx;
     infile1 >> hy_locationy;
     infile1 >> hy_locationz;

if (infile1.eof()) break;
 no_detectors ++;
}
```

```
outfile << no_detectors << endl;
outfile << endl;

ifstream infile2(array_data);
if (infile2.fail()) {
     cout << "Array data file " << array_data << " NOT FOUND" << endl;
     return 1;
   }                                                                          180

while (!infile2.eof()) {


     double hy_locationx = 0;
     double hy_locationy = 0;
     double hy_locationz = 0;

     infile2 >> hy_locationx;
     infile2 >> hy_locationy;                                                 190
     infile2 >> hy_locationz;

     if (infile2.eof()) break;

  outfile << hy_locationx << " " << hy_locationy << " " << hy_locationz << " " << endl;

}

outfile << endl;
                                                                              200
 //define maximum and minium energy for showers
 //-----------------

 double eMin = 1E17;
 double eMax = 1E21;


 //define z of maximum energy (from evaluating dedz eqn)
 //-------------------
 float zmaxenergy = 5.22;                                                     210


 //loop over number of events
 //----------
 double eventcount = 0;

 for (int event_no = 1; event_no <= num_events; ++event_no){


   //if (event_no%10 == 0){                                                  220

   //cout << event_no << endl;
   //}

   //Define random enaergy
   //--------
   double neutrino_energy = 0;

//    //Test to check random energy
```

```
//  for (int g; g< 100000; g++){

    neutrino_energy = randomEnergy(eMin, eMax, &ranGen);
//    energyCheck.Fill(log10(neutrino_energy));
//  }

    // Define hydrophone location:
    // ------------

    double angle       = 0;
    double hy_radius   = 0;
    double hy_dir      = 0;
    double x_sensor    = 0;
    double y_sensor    = 0;
    double z_sensor    = 0;
    double x_cascade   = 0;
    double y_cascade   = 0;
    double z_cascade   = 0;
    double theta_cascade = 0;
    double phi_cascade = 0;
    double x_dir       = 0;
    double y_dir       = 0;
    double z_dir       = 0;

    //Random position
    TVector3 ranPos = randomPosition(&ranGen);
    x_cascade = ranPos.X();
    y_cascade = ranPos.Y();
    z_cascade = ranPos.Z();

    //Random direction
    TVector3 ranDir = randomPosition(&ranGen);

    x_dir = ranDir.X();
    y_dir = ranDir.Y();
    z_dir = ranDir.Z();

    theta_cascade = ranDir.Theta();
    phi_cascade = ranDir.Phi();

    outfile << event_no << " " << neutrino_energy << " " << x_cascade << " " << y_cascade << " " << z_cascade << " "

    //Define an array
    //==============

    //Read in file
    //-----
    ifstream infile(array_data);
    if (infile.fail()) {
      cout << "Array data file " << array_data << " NOT FOUND" << endl;
      return 1;
    }

    //counter to give detector number

    //loop through all detectors
    //----------
    while (!infile.eof()) {
```

```
double hy_locationx = 0;
double hy_locationy = 0;                                                    290
double hy_locationz = 0;

infile >> hy_locationx;
infile >> hy_locationy;
infile >> hy_locationz;

if (infile.eof()) break;

//test to see if reading correctly
//————————————-                                                            300
//      std::cout << "New hydrophone x,y,z = " << hy_locationx << " , " << hy_locationy << " , " << hy_locationz << st



//calculate distance from hydrophone


//x_cascade = 2000;
//y_cascade = 0;
//z_cascade = 300;                                                          310

//x_dir = 0;
//y_dir = 0;
//z_dir = 1;

hy_radius = pow((pow((x_cascade−hy_locationx),2) + pow((y_cascade−hy_locationy),2) + pow((z_cascade−hy_locationz),2)),

// cout << hy_locationx << " " << hy_locationy << " " << hy_locationz << " " << endl;

                                                                            320
//find angle that the acoustic pulse make with the detector
//      TVector3 dir_vector = ((-x_cascade + hy_locationx),(-y_cascade + hy_locationy),(-z_cascade + hy_locationz));

double dir_vectorx = 0;
double dir_vectory = 0;
double dir_vectorz = 0;
double dir_vector_mag = 0;

dir_vectorx = (−x_cascade + hy_locationx);
dir_vectory = (−y_cascade + hy_locationy);                                  330
dir_vectorz = (−z_cascade + hy_locationz);
dir_vector_mag = pow(pow(dir_vectorx,2)+pow(dir_vectory,2)+pow(dir_vectorz,2),0.5);

//cout << dir_vectorx << " " << dir_vectory << " " << dir_vectorz << " " <<endl;

angle = (180.0/M_PI)*((M_PI/2.0)−acos(((x_dir*dir_vectorx)+(y_dir*dir_vectory)+(z_dir*dir_vectorz))/(dir_vector_mag*ranD

// cout<< angle <<endl;

outfile << hy_radius << " " << angle << " ";                                340

//————————————————————————————————————————

//Test to compare acoutic pulses generated with new code with old code
//————————————————————————
```

```
// hy_radius = 1000;
// neutrino_energy = 20*1E12;
// angle = 0.0;
   //ranPosX = 1000;
   // ranPosY = 0;
   //ranPosZ = 0;

//-------------------------------------------------------------------


//exclude the no hopers
//---------

if ((angle < 6.0 && angle > -6.0) || (hy_radius < 1000.0) ){

  eventcount = eventcount +1;


  //Transform to shower-centric co-ordinates
  //-----------------
  x_sensor = hy_radius*cos(M_PI*angle/180.0);
  y_sensor = 0.0;
  z_sensor = hy_radius*sin(M_PI*angle/180.0);

  // Time-window for pressure pulse :
  // -------------
  double start_time = (hy_radius/c) - 0.0002;
  double end_time = (hy_radius/c) + 0.0002;
  int num_times    = 512;
  vector<double> times(num_times);

  for (int i = 0; i < num_times; ++i) {
    times[i] = start_time + (i*(end_time-start_time)/num_times);
  }


  // Define integrated pressure pulse arrays:
  // ----------
  vector<double> pressure(num_times);
  vector<double> pinc(num_times);

  // Setup Monte Carlo integration parameters (in SI units) :
  // --------------------
  double energy_density_sum = 0.0;
  double zmax = 10.0;
  double rmax = 0.3;

  for(int j=0; j <= nint; j++){

    double z0 = 0;
    double x0 = 0;
    double y0 = 0;
    double phi0 = 0;
    double r_ran = 0;
    double z_ran = 0;
```

```
// Generating shower :
// ————————

z_ran = (ranGen.Rndm())*zmax − 5.22;
r_ran = (ranGen.Rndm())*rmax;
// double theta = atan(z_ran/r_ran);
phi0 = (ranGen.Rndm())*(2*M_PI);                                        410

//change to cartesian co-ords
//————————————

x0 = r_ran*cos(phi0);
y0 = r_ran*sin(phi0);
z0 = z_ran;


// Calculate distance from measurement point :                          420
// ————————————————-

double r = pow((pow((x_sensor − x0),2) + pow((y_sensor − y0),2) + pow((z_sensor − z0),2)),0.5);


//Find the energy at that point
//————————————-

//dedz calculator (from geant analysis)
//————————————————                                                      430

double par1z = 9;
double par2z = −1.75;

double dedz1 = pow(fabs(z_ran+5.22),par1z)*exp(par2z*(fabs(z_ran+5.22)));

//dedr calculator (from geant analysis)
//————————————————

double par1r =24.6511;                                                  440
double par4r = 9.56813;

double r_ran_mm = r_ran*1000;

double dedr1 = exp(−(r_ran_mm−par4r)/par1r);


//energy density (in eV)
//—————————
double energy_density = dedz1*dedr1*1.0E12;                             450
energy_density_sum = energy_density + energy_density_sum;

// Find the pressure pulse :
// ————————————-
dp(&times,&pinc,r,atype,temp,depth,salinity,beta,cp,omega,c);

// Superpose it :
// ———————
scaledVectorAdd(&pressure,&pinc,energy_density);
                                                                        460
}
```

```
        //define a scaling number in J
        //————————————-
        double scaling = neutrino_energy*1.602E−19/energy_density_sum;

        // Store pressure pulse in a histogram :
        // ——————————————-
        TH1F signal("pulse","acoustic pulse",num_times,start_time,end_time);
        for (int i = 0; i < num_times; ++i) {                                    470
          signal.SetBinContent(i,pressure[i]*scaling*0.72);

        }

        //calculate the pulse height and time
        maxpres = 0;
        maxpres = signal.GetBinContent(signal.GetMaximumBin());
         maxpt = 0;
        maxpt = times[signal.GetMaximumBin()];
                                                                             480

        //my_Pulse->cd();
        //signal.Write();

       outfile <<maxpres << " " << maxpt << " ";

      }

else{
                                                                             490
        maxpt = 0;
        maxpres = 0;

        outfile <<maxpres << " " << maxpt << " ";

      }



    }                                                                        500
    outfile <<endl;
  }

  cout << maxpres <<endl;

  //cout<<“This many particles made the cut ”<<((eventcount/num_events)/no_detectors)*100<<“%”<<endl;

  my_Pulse−>cd();
  energyCheck.Write();
  my_Pulse−>Close();                                                         510
  outfile.close();
  return 0;
       }
```

## B.17 array_log_anal.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//30/01/04                                                     //
//                                                             //
//Root macro for filling a histo of all results ready          //
//for analysis                                                 //
//                                                             //
//=============================================================//

#include "TROOT.h"                                                    10
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TH1F.h"
#include "TFile.h"                                                    20

//=============================================================

//define the variables
//-----------
double temp = 0;
double depth = 0;
double salinity = 0;
double velocity = 0;
double coef = 0;                                                      30
double heatcap = 0;

int no_hydro = 0;
int hydro_counter = 0;
float pres_cut = 0; float pres_cut_max = 0.4;

//Define array of histograms
//----------
TH1F *histo[21];
                                                                      40
int histonum=1;

int main(int argc, char *argv[]){

  //number of events per file
  int no_event = 1000;

  //loop over arrays
  //------
  for(int k=1; k <= 5; k++){                                          50

    cout << "analysing array number " << k << endl;

    histonum=1;
```

```
// Make output file :
// ————————
Char_t filename[20];
sprintf(filename,"array_%d_anal_comp.root",k);
TFile* array_anal = new TFile(filename,"RECREATE","neutrinosim histos");

//loop through hydrophone cuts
//————————————-
for(pres_cut = 0; pres_cut <= pres_cut_max; pres_cut = pres_cut + 0.025){

  cout << "analysing pressure " << pres_cut << endl;

  //create histos for analysis
  //————————————-
  histonum = histonum +1;

  Char_t histo_title[20];
  sprintf(histo_title,"pres_cut_%f",pres_cut);

  Char_t histo_name[20];
  sprintf(histo_name,"pres_cut_%f",pres_cut);

  histo[histonum] = new TH1F(histo_title,histo_title,100,0.0,100.0);

  const int no_events = no_event + 1;

  //define the arrays for energy, position, and direction
  //————————————————————————-
  int event_no[no_events];
  double energy[no_events];

  double positionx[no_events];
  double positiony[no_events];
  double positionz[no_events];

  double directionx[no_events];
  double directiony[no_events];
  double directionz[no_events];

  //loop over files per array
  //————————————
  for(int i = 1; i<=50; i++){

    //open log file and check that it exists
    //————————————————————-
    Char_t logname[20];
    sprintf(logname, "array%d_%d.log",k,i);

    ifstream arrayfile(logname);
    if (arrayfile.fail()) {
      cout << "Array data file " << logname << " NOT FOUND" << endl;
      return 1;
    }

    //read in log file
    //———————
    arrayfile >> temp;
    arrayfile >> depth;
```

```
arrayfile >> salinity;
arrayfile >> velocity;
arrayfile >> coef;
arrayfile >> heatcap;

arrayfile >> no_hydro;
                                                                        120
const int no_hydros = no_hydro;

double hy_locationx[no_hydros+1];
double hy_locationy[no_hydros+1];
double hy_locationz[no_hydros+1];

double radius[no_hydros+1];
double angle[no_hydros+1];
double peak_pres[no_hydros+1];
double peak_time[no_hydros+1];                                          130

for(int hydro_num = 1; hydro_num <= no_hydros; hydro_num++){
  arrayfile >> hy_locationx[hydro_num];
  arrayfile >> hy_locationy[hydro_num];
  arrayfile >> hy_locationz[hydro_num];
}

int event_num=0;

while (!arrayfile.eof() && event_num<no_event) {                        140

  int thiseventnumber;
  arrayfile >> thiseventnumber;

  // If we've read past the end of the file, bail out now :
  if(arrayfile.eof()) break;

  // Otherwise, this is a successful event read :
  event_num++;
                                                                        150
  event_no[event_num]=thiseventnumber;
  hydro_counter = 0;

  arrayfile >> energy[event_num];

  arrayfile >> positionx[event_num];
  arrayfile >> positiony[event_num];
  arrayfile >> positionz[event_num];

  arrayfile >> directionx[event_num];                                   160
  arrayfile >> directiony[event_num];
  arrayfile >> directionz[event_num];

  //for each event, loop through all hydrophones
  //--------------------
  for(int hydro_num = 1; hydro_num<= no_hydros ; hydro_num++){

    arrayfile >> radius[hydro_num];
    arrayfile >> angle[hydro_num];
    arrayfile >> peak_pres[hydro_num];                                  170
    arrayfile >> peak_time[hydro_num];
```

```
        if(peak_pres[hydro_num] > pres_cut){

          //make a counter of hydro number
          hydro_counter = hydro_counter +1;
        }

      }                                                                    180

      //=============================================================================

      //fill histos
      //————
      histo[histonum]->Fill(hydro_counter);
    }
  }

  array_anal->cd();                                                        190
  histo[histonum]->Write();

 }

 array_anal->Close();
}
   }
```

## B.18 array_log_further_anal.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//30/01/04                                                     //
//                                                             //
//Root macro for reading making histos of radius and energy    //
//for each array                                               //
//                                                             //
//=============================================================//

#include "TROOT.h"                                                         10
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TH1F.h"
#include "TFile.h"                                                         20

//=================================================================

//define the variables
//——————————

double temp = 0;
```

```
double depth = 0;
double salinity = 0;
double velocity = 0;
double coef = 0;
double heatcap = 0;

int no_hydro = 0;
int hydro_counter = 0;
float pres_cut = 0; float pres_cut_max = 0.025;

int energy_E20 = 0;

//Define array of histograms
//————————————
TH1F *histo1[100];
TH1F *histo2[100];
TH1F *histo3[100];
TH1F *histo4[100];
TH1F *histo5[100];
TH1F *histo6[100];
TH1F *histo7[100];
TH1F *histo8[100];
TH1F *histo9[100];
TH1F *histo10[100];

int histonum=1;

int main(int argc, char *argv[]){

  //number of events per file
  int no_event = 1000;

  for(int k=1; k <= 5; k++){

    cout << "analysing array number " << k << endl;

    histonum=1;
    energy_E20 = 0;

    // Make output file :
    // ————————
    Char_t filename[20];
    sprintf(filename,"array_%d_anal_50000_comp_count.root",k);
    TFile* array_anal = new TFile(filename,"RECREATE","neutrinosim histos");

    //loop through hydrophone cuts
    //————————————-
    for(pres_cut = 0.025; pres_cut <= pres_cut_max; pres_cut = pres_cut + 0.025){

      cout << "analysing pressure " << pres_cut << endl;

      //create histos for analysis
      //————————————
      histonum = histonum +1;

      Char_t histo_title1[40];
      sprintf(histo_title1,"array_%d_pres_cut_%f",k,pres_cut);
```

```
Char_t histo_title2[40];
sprintf(histo_title2,"array_%d_pres_cut_%f_dist_1",k,pres_cut);

Char_t histo_title3[40];
sprintf(histo_title3,"array_%d_pres_cut_%f_dist_3",k,pres_cut);

Char_t histo_title4[40];
sprintf(histo_title4,"array_%d_pres_cut_%f_dist_4",k,pres_cut);

Char_t histo_title5[40];
sprintf(histo_title5,"array_%d_pres_cut_%f_dist_9",k,pres_cut);

Char_t histo_title6[40];
sprintf(histo_title6,"array_%d_pres_cut_%f_energy_1",k,pres_cut);

Char_t histo_title7[40];
sprintf(histo_title7,"array_%d_pres_cut_%f_energy_3",k,pres_cut);

Char_t histo_title8[40];
sprintf(histo_title8,"array_%d_pres_cut_%f_energy_4",k,pres_cut);

Char_t histo_title9[40];
sprintf(histo_title9,"array_%d_pres_cut_%f_energy_9",k,pres_cut);

Char_t histo_name[40];
sprintf(histo_name,"pres_cut_%f",pres_cut);

histo1[histonum] = new TH1F(histo_title1,histo_title1,100,0.0,100.0);
histo2[histonum] = new TH1F(histo_title2,histo_title2,100,0.0,10000.0);
histo3[histonum] = new TH1F(histo_title3,histo_title3,100,0.0,10000.0);
histo4[histonum] = new TH1F(histo_title4,histo_title4,100,0.0,10000.0);
histo5[histonum] = new TH1F(histo_title5,histo_title5,100,0.0,10000.0);

//need to change axis limits
histo6[histonum] = new TH1F(histo_title6,histo_title6,80,17,21);
histo7[histonum] = new TH1F(histo_title7,histo_title7,80,17,21);
histo8[histonum] = new TH1F(histo_title8,histo_title8,80,17,21);
histo9[histonum] = new TH1F(histo_title9,histo_title9,80,17,21);

const int no_events = no_event + 1;

//define the arrays for energy, positio, and direction
//--------------------.
int event_no[no_events];
double energy[no_events];

double positionx[no_events];
double positiony[no_events];
double positionz[no_events];

double directionx[no_events];
double directiony[no_events];
double directionz[no_events];

for(int i = 1; i<=50; i++){//file loop

  //open log file and check that it exists
  //----------------.
```

```
Char_t logname[20];
sprintf(logname, "array%d/array%d_%d.log",k,k,i);

ifstream arrayfile(logname);
if (arrayfile.fail()) {
  cout << "Array data file " << logname << " NOT FOUND" << endl;
  return 1;
}

//read in log file
//------
arrayfile >> temp;
arrayfile >> depth;
arrayfile >> salinity;
arrayfile >> velocity;
arrayfile >> coef;
arrayfile >> heatcap;

arrayfile >> no_hydro;

const int no_hydros = no_hydro;

double hy_locationx[no_hydros+1];
double hy_locationy[no_hydros+1];
double hy_locationz[no_hydros+1];

double radius[no_hydros+1];
double angle[no_hydros+1];
double peak_pres[no_hydros+1];
double peak_time[no_hydros+1];
double radius_temp[no_hydros+1];

for(int hydro_num = 1; hydro_num <= no_hydros; hydro_num++){
  arrayfile >> hy_locationx[hydro_num];
  arrayfile >> hy_locationy[hydro_num];
  arrayfile >> hy_locationz[hydro_num];
}

int event_num=0;

while (!arrayfile.eof() && event_num<no_event) {//event_loop

  int thiseventnumber;
  arrayfile >> thiseventnumber;

  // If we've read past the end of the file, bail out now :
  if(arrayfile.eof()) break;

  // Otherwise, this is a successful event read :
  event_num++;

  event_no[event_num]=thiseventnumber;
  hydro_counter = 0;

  arrayfile >> energy[event_num];

  if(energy[event_num] >= 1E20){
```

```
  energy_E20 = energy_E20 + 1;

}

arrayfile >> positionx[event_num];
arrayfile >> positiony[event_num];
arrayfile >> positionz[event_num];

arrayfile >> directionx[event_num];                                    210
arrayfile >> directiony[event_num];
arrayfile >> directionz[event_num];

//for each event, loop through all hydrophones
//————————————————————
for(int hydro_num = 1; hydro_num<= no_hydros ; hydro_num++){

  arrayfile >> radius[hydro_num];
  arrayfile >> angle[hydro_num];
  arrayfile >> peak_pres[hydro_num];                                   220
  arrayfile >> peak_time[hydro_num];


  if(peak_pres[hydro_num] > pres_cut){

    //make a counter of hydro number
    hydro_counter = hydro_counter +1;
    radius_temp[hydro_counter]=radius[hydro_num];

  }                                                                    230

}

//=============================================================================

//fill histos
//————
histo1[histonum]->Fill(hydro_counter);

if(hydro_counter >= 1){                                                240
  for(int i = 1; i<= hydro_counter; i++){
    histo2[histonum]->Fill(radius_temp[i]);
  }
}

if(hydro_counter >= 3){
  for(int i = 1; i<= hydro_counter; i++){
    histo3[histonum]->Fill(radius_temp[i]);
  }
}                                                                      250

if(hydro_counter >= 4){
  for(int i = 1; i<= hydro_counter; i++){
    histo4[histonum]->Fill(radius_temp[i]);
  }
}

if(hydro_counter >= 9){
  for(int i = 1; i<= hydro_counter; i++){
```

```
            histo5[histonum]->Fill(radius_temp[i]);
          }
        }

        if(hydro_counter >= 1){
          histo6[histonum]->Fill(log10(energy[event_num]));
        }

        if(hydro_counter >= 3){
          histo7[histonum]->Fill(log10(energy[event_num]));
        }

        if(hydro_counter >= 4){
          histo8[histonum]->Fill(log10(energy[event_num]));
        }

        if(hydro_counter >= 9){
          histo9[histonum]->Fill(log10(energy[event_num]));
        }
      }
    }

    cout << "array " << k << " " << energy_E20 << endl;

    array_anal->cd();
    histo1[histonum]->Write(); histo2[histonum]->Write(); histo3[histonum]->Write(); histo4[histonum]->Write();
    histo5[histonum]->Write(); histo6[histonum]->Write(); histo7[histonum]->Write(); histo8[histonum]->Write();
    histo9[histonum]->Write();
  }

  array_anal->Close();
 }
}
```

---

## B.19 array_further_anal_plotter.C

---

```
//=========================================================//
//Created by Simon Bevan                                   //
// 10/02/04 to tidy code                                   //
//                                                         //
//Root macro for displaying array analysis results         //
//=========================================================//

#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"
```

```
#include "TCanvas.h"

void all_array_further_anal_plotter(){//

  //=============================================================

  //Set up for root
  //===============

  // Clean up :
  // ==========
  gROOT->Reset();

  // No stats :
  // ==========
  gStyle->SetOptStat(0);
  gROOT->SetStyle("Plain");

  c1 = new TCanvas("c1","neutrinosim",200,10,700,500);

  c1->SetFillColor(0);
  c1->SetGridx();
  c1->SetGridy();

  //=============================================================

  TH1F *histo[10];
  TFile *pulse[10];
  TH1F *historad[400];
  TFile *pulserad[400];
  TH1F *histoenergy[400];
  TFile *pulseenergy[400];
  int dist = 1;

  //Comparing all arrays
  //--------

  //Loop through all files creating histograms
  //----------------
  TH1F* one_histo         = new TH1F("all","all",5,0.0,5.0);
  TH1F* above_three_histo = new TH1F("array_comp_three","array_comp_three",5,0.0,5.0);
  TH1F* above_four_histo  = new TH1F("array_comp_four","array_comp_four",5,0.0,5.0);
  TH1F* above_nine_histo  = new TH1F("array_comp_nine","array_comp_nine",5,0.0,5.0);

  for(int i =1; i<=5; i++){//array_loop

    Char_t filename[30];

    std::sprintf(filename, "array_%d_anal_50000_comp.root",i);

    cout << "opening file " << filename << endl;

    //tell which cut you would like
    //----------
    Char_t histoname[40];
    sprintf(histoname,"array_%d_pres_cut_0.350000",i);

    //grab the histo
```

```
  //------
  pulse[i] = new TFile(filename);
  histo[i] = (TH1F*) pulse[i]->Get(histoname);

  //make it look nice                                                    80
  //-------
  histo[i]->SetLineWidth(1.0);
  histo[i]->SetLineStyle(1);
  histo[i]->SetLineColor(i);

  //get varying coincidences and plot onto the same graph
  //--------------------
  Double_t one = 0;
  for (Int_t w=2; w<= histo[i].GetNbinsX(); ++w) {
    one = one + histo[i].GetBinContent(w);                             90
  }
  one_histo->SetBinContent(i, one);

  Double_t above_three = 0;
  for (Int_t w=4; w<= histo[i].GetNbinsX(); ++w) {
    above_three = above_three + histo[i].GetBinContent(w);
  }
  above_three_histo->SetBinContent(i, above_three);

  Double_t above_four = 0;                                             100
  for (Int_t w=5; w<= histo[i].GetNbinsX(); ++w) {
    above_four = above_four + histo[i].GetBinContent(w);
  }
  above_four_histo->SetBinContent(i, above_four);

  Double_t above_nine = 0;
  for (Int_t w=10; w<= histo[i].GetNbinsX(); ++w) {
    above_nine = above_nine + histo[i].GetBinContent(w);
  }
  above_nine_histo->SetBinContent(i, above_nine);                      110

}

//make them differentiable
//----------
one_histo->SetLineColor(1);
one_histo->SetLineWidth(2.0);
one_histo->SetLineStyle(1);

above_three_histo->SetLineColor(2);                                    120
above_three_histo->SetLineWidth(2.0);
above_three_histo->SetLineStyle(1);

above_four_histo->SetLineColor(3);
above_four_histo->SetLineWidth(2.0);
above_four_histo->SetLineStyle(1);

above_nine_histo->SetLineColor(4);
above_nine_histo->SetLineWidth(2.0);
above_nine_histo->SetLineStyle(1);                                     130

//set axis
//--
```

```
one_histo->SetTitle("Hydrophone Cut of 0.350Pa");
one_histo->GetXaxis()->SetTitle("Array spacing / m");
one_histo->GetXaxis()->CenterTitle();
one_histo->GetXaxis()->SetTitleOffset(1.0);
one_histo->GetYaxis()->SetTitle("Number of events");
one_histo->GetYaxis()->SetTitleFont(12);
one_histo->GetYaxis()->SetTitleSize(0.06);
one_histo->GetYaxis()->SetTitleOffset(1.0);
one_histo->GetYaxis()->CenterTitle();

//draw the histos
//------
one_histo->Draw();
above_three_histo->Draw("Same");
above_four_histo->Draw("Same");
above_nine_histo->Draw("Same");

TLegend *leg = new TLegend(0.6,0.7,0.89,0.89);

leg->AddEntry(one_histo, "One or more hydrophones fired", "l");
leg->AddEntry(above_three_histo, "Three or more hydrophones in coincidence", "l");
leg->AddEntry(above_four_histo, "Four or more hydrophones in coincidence", "l");
leg->AddEntry(above_nine_histo, "Nine or more hydrophones in coincidence", "l");

leg->Draw();

//-------------------------------------

//The radius graphs per array
//----------

int histonumber = 1;
int pulsenumber = 1;
int coinc = 1;
int colour_count = 1;

for (int j = 4; j<=4; j++) {//array_loop
  for (int k = 3; k <= 3; k++){//coinc_loop
    if (k ==1){
      coinc = 1;
    }
    if (k ==2){
      coinc = 3;
    }
    if (k ==3){
      coinc = 4;
    }
    if (k ==4){
      coinc = 9;
    }

    Char_t filename[40];
    sprintf(filename,"array_%d_anal_50000_comp.root",j);

    for(int cut = 2; cut <= 2; cut++){//for cut definition see below

      if(cut ==1){
        Char_t histoname[40];
```

```
        sprintf(histoname,"array_%d_pres_cut_0.000000_dist_%d",j,coinc);
      }
      if(cut ==2){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.025000_dist_%d",j,coinc);
      }
      if(cut ==3){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.050000_dist_%d",j,coinc);
      }
      if(cut ==4){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.100000_dist_%d",j,coinc);
      }
      if(cut ==5){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.150000_dist_%d",j,coinc);
      }
      if(cut ==6){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.200000_dist_%d",j,coinc);
      }
      if(cut ==7){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.250000_dist_%d",j,coinc);
      }
      if(cut ==8){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.300000_dist_%d",j,coinc);
      }
      if(cut ==9){
        Char_t histoname[40];
        sprintf(histoname,"array_%d_pres_cut_0.350000_dist_%d",j,coinc);
      }

      pulserad[pulsenumber] = new TFile(filename);
      historad[histonumber] = (TH1F*) pulserad[pulsenumber]->Get(histoname);

      historad[histonumber]->SetLineWidth(1.0);
      historad[histonumber]->SetLineStyle(1);

      histonumber = histonumber +1;
      pulsenumber = pulsenumber +1;
    }
  }
}

int color_count = 0;

historad[1]->SetLineColor(1);
historad[2]->SetLineColor(2);
historad[3]->SetLineColor(3);
historad[4]->SetLineColor(4);
historad[5]->SetLineColor(5);
historad[6]->SetLineColor(5);
historad[7]->SetLineColor(6);
historad[8]->SetLineColor(7);
historad[9]->SetLineColor(8);
```

```
  //set axis
  //---
  historad[1]->SetTitle("Coincidence of 4, cut of 0.025Pa");
  historad[1]->GetXaxis()->SetTitle("radius / m");
  historad[1]->GetXaxis()->CenterTitle();
  historad[1]->GetXaxis()->SetTitleOffset(1.0);
  historad[1]->GetYaxis()->SetTitle("Number of events");
  historad[1]->GetYaxis()->SetTitleFont(12);
  historad[1]->GetYaxis()->SetTitleSize(0.06);
  historad[1]->GetYaxis()->SetTitleOffset(1.0);
  historad[1]->GetYaxis()->CenterTitle();

  //draw the histos
  //------

  historad[1]->Draw();
  historad[2]->Draw("SAME");
  historad[3]->Draw("SAME");
  historad[4]->Draw("SAME");
  historad[5]->Draw("SAME");
  historad[6]->Draw("SAME");
  historad[7]->Draw("SAME");
  historad[8]->Draw("SAME");
  historad[9]->Draw("SAME");

  TLegend *legrad = new TLegend(0.6,0.7,0.89,0.89);

  legrad->AddEntry(historad[1], "array 1", "l");
  legrad->AddEntry(historad[2], "array 2", "l");
  legrad->AddEntry(historad[3], "array 3", "l");
  legrad->AddEntry(historad[4], "array 4", "l");
  legrad->AddEntry(historad[5], "array 5", "l");
//   legrad->AddEntry(histo[6], "cut of 0.20", "l");
//   legrad->AddEntry(histo[7], "cut of 0.25", "l");
//   legrad->AddEntry(histo[8], "cut of 0.30", "l");
//   legrad->AddEntry(histo[9], "cut of 0.35", "l");

  legrad->Draw();

  //-------------------------------------------------

  //The energy graphs
  //-------

//   int pulsenumber = 1;
//   int coinc = 1;
//   int colour_count = 1;

//   for (int j = 1; j<=5; j++) {//array_loop
//     for (int k = 2; k <= 2; k++){
//       if (k ==1){
//      coinc = 1;
//       }
//       if (k ==2){
//      coinc = 3;
//       }
//       if (k ==3){
```

```
//      coinc = 4;
//       }
//       if (k ==4){
//      coinc = 9;
//       }

//       Char_t filename[40];
//       sprintf(filename,"array_%d_anal_50000_comp.root",j);

//       for(int cut = 2; cut <= 2; cut++){

//      if(cut ==1){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.000000_energy_%d",j,coinc);
//      }
//      if(cut ==2){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.025000_energy_%d",j,coinc);
//      }
//      if(cut ==3){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.050000_energy_%d",j,coinc);
//      }
//      if(cut ==4){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.100000_energy_%d",j,coinc);
//      }
//      if(cut ==5){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.150000_energy_%d",j,coinc);
//      }
//      if(cut ==6){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.200000_energy_%d",j,coinc);
//      }
//      if(cut ==7){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.250000_energy_%d",j,coinc);
//      }
//      if(cut ==8){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.300000_energy_%d",j,coinc);
//      }
//      if(cut ==9){
//        Char_t histoname[40];
//        sprintf(histoname,"array_%d_pres_cut_0.350000_energy_%d",j,coinc);
//      }

//      pulseenergy[pulsenumber] = new TFile(filename);
//      histoenergy[histonumber] = (TH1F*) pulseenergy[pulsenumber]->Get(histoname);

//      histoenergy[histonumber]->SetLineWidth(1.0);
//      histoenergy[histonumber]->SetLineStyle(1);

//      histonumber = histonumber +1;
//      pulsenumber = pulsenumber +1;
//       }
//     }
```

```
//  }

//  histoenergy[1]->SetLineColor(1);
//  histoenergy[2]->SetLineColor(2);
//  histoenergy[3]->SetLineColor(3);                                          370
//  histoenergy[4]->SetLineColor(4);
//  histoenergy[5]->SetLineColor(5);
//  histoenergy[6]->SetLineColor(5);
//  histoenergy[7]->SetLineColor(6);
//  histoenergy[8]->SetLineColor(7);
//  histoenergy[9]->SetLineColor(8);

//  //set axis
//  //——-
//  histoenergy[1]->SetTitle("Coincidence of 3 cut of 0.025");                380
//  histoenergy[1]->GetXaxis()->SetTitle("log(energy) / eV");
//  histoenergy[1]->GetXaxis()->CenterTitle();
//  histoenergy[1]->GetXaxis()->SetTitleOffset(1.0);
//  histoenergy[1]->GetYaxis()->SetTitle("Number of events");
//  histoenergy[1]->GetYaxis()->SetTitleFont(12);
//  histoenergy[1]->GetYaxis()->SetTitleSize(0.06);
//  histoenergy[1]->GetYaxis()->SetTitleOffset(1.0);
//  histoenergy[1]->GetYaxis()->CenterTitle();

//  //draw the histos                                                         390
//  //——————

//  histoenergy[1]->Draw();
//  histoenergy[2]->Draw("SAME");
//  histoenergy[3]->Draw("SAME");
//  histoenergy[4]->Draw("SAME");
//  histoenergy[5]->Draw("SAME");
//  histoenergy[6]->Draw("SAME");
//  histoenergy[7]->Draw("SAME");
//  histoenergy[8]->Draw("SAME");                                             400
//  histoenergy[9]->Draw("SAME");

//  TLegend *legenergy = new TLegend(0.6,0.7,0.89,0.89);

//  legenergy->AddEntry(histo[1], "array 1 ", "l");
//  legenergy->AddEntry(histo[2], "array 2", "l");
//  legenergy->AddEntry(histo[3], "array 3", "l");
//  legenergy->AddEntry(histo[4], "array 4", "l");
//  legenergy->AddEntry(histo[5], "array 5", "l");
//  legenergy->AddEntry(histo[6], "cut of 0.20", "l");                        410
//  legenergy->AddEntry(histo[7], "cut of 0.25", "l");
//  legenergy->AddEntry(histo[8], "cut of 0.30", "l");
//  legenergy->AddEntry(histo[9], "cut of 0.35", "l");

//  legenergy->Draw();

//  //=============================================================================

//  //Showing the effect of the cuts for array 3 500m spacing
//  //————————————————————                                                    420

//  TH1F *histo[17];
//  TFile *pulse[17];
```

```
//   TH1F* array_3          = new TH1F("array_3","array_3",16,0.0,0.5);

//   for(Int_t i = 1; i<=16; i++){
//     pulse[i] = new TFile("array_3_anal_comp_500.root");
//   }
                                                                                  430
//   histo[1] = (TH1F*) pulse[1]->Get("pres_cut_0.000000");
//   histo[2] = (TH1F*) pulse[2]->Get("pres_cut_0.025000");
//   histo[3] = (TH1F*) pulse[3]->Get("pres_cut_0.050000");
//   histo[4] = (TH1F*) pulse[4]->Get("pres_cut_0.075000");
//   histo[5] = (TH1F*) pulse[5]->Get("pres_cut_0.100000");
//   histo[6] = (TH1F*) pulse[6]->Get("pres_cut_0.125000");
//   histo[7] = (TH1F*) pulse[7]->Get("pres_cut_0.150000");
//   histo[8] = (TH1F*) pulse[8]->Get("pres_cut_0.175000");
//   histo[9] = (TH1F*) pulse[9]->Get("pres_cut_0.200000");
//   histo[10] = (TH1F*) pulse[10]->Get("pres_cut_0.225000");                       440
//   histo[11] = (TH1F*) pulse[11]->Get("pres_cut_0.250000");
//   histo[12] = (TH1F*) pulse[12]->Get("pres_cut_0.275000");
//   histo[13] = (TH1F*) pulse[13]->Get("pres_cut_0.300000");
//   histo[14] = (TH1F*) pulse[14]->Get("pres_cut_0.325000");
//   histo[15] = (TH1F*) pulse[15]->Get("pres_cut_0.350000");
//   histo[16] = (TH1F*) pulse[16]->Get("pres_cut_0.375000");

//   Double_t j = 0;

//   for(Int_t i = 1; i<=16; i++){                                                 450

//     Double_t total = 0;
//     j= j+0.025;

//     for (Int_t w=5; w<= histo[i]->GetNbinsX(); ++w) {
//       total = total + histo[i]->GetBinContent(w);
//     }
//     array_3->SetBinContent(i, total);
//   }
                                                                                  460
      }
```

## B.20 recon.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//30/01/04                                                     //
//                                                             //
//Root macro for testing reconstructed dist against actual     //
//                                                             //
//=============================================================

#include "TROOT.h"
#include <stdio.h>                                                                10
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
```

```
#include <vector>

#include "TH1F.h"
#include "TFile.h"
#include "TH2F.h"                                                                    20

//=============================================================================

TH2F *histo[10];
TH2F *angle1[10];
TH2F *angle2[10];

int main(int argc, char *argv[]){

  //define some of the variables                                                    30
  //------------
  double temp = 0;
  double depth = 0;
  double salinity = 0;
  double velocity = 0;
  double coef = 0;
  double heatcap = 0;

  int no_hydro = 0;
                                                                                   40
  int counter = 0;

  double hy_locationx[100110]; double hy_locationy[100110]; double hy_locationz[100110];

  double radius[100110];
  double angle[100110];
  double peak_pres[100110];
  double peak_time[100110];
  int histo_num = 0;
                                                                                   50
  int event_num=0;
  int array_num=0;

  double theta1 = 0;
  double theta2 = 0;

  double cos1 = cos(theta1);
  double cos2 = cos(theta2);
  double sin1 = sin(theta1);
  double sin2 = sin(theta2);                                                       60

  double simonr_temp = 0;

  double hydro_sep1 = 0; double hydro_sep2 = 0;
  double max_peak1 = 0;
  int max_hydro1 = 0; int max_hydro2 = 0; int max_hydro3 = 0;
  double hydro_x1 = 0;double hydro_y1 = 0;
  int hydro1 = 0; int hydro2 = 0; int hydro3 = 0;
  int coinc_counter = 1;
  double act_radius = 0;                                                           70
  double time_hydro1 = 0; double time_hydro2 = 0; double time_hydro3 = 0;
  double time1 = 0; double time2 = 0; double time3 = 0;
```

```
double act_angle1= 0;double act_angle2= 0;
double counter1 = 0; double counter2 = 0; double counter3 = 0;

//number of events per file
int no_event = 1000;

// Make output file :                                                    80
// ————————
Char_t filename[20];
sprintf(filename,"array_dist_recon_anal.root");
TFile* recon_anal = new TFile(filename,"RECREATE","neutrinosim histos");

for(int k=1; k <= 1; k++){//array loop

  //create histos for analysis
  //————————————
  const int no_events = no_event + 1;                                   90

  Char_t histo_title[20];
  sprintf(histo_title,"recon_array_%d",k);
  histo[k] = new TH2F(histo_title, histo_title, 1000, 0.0,10000.0, 1000, 0.0,10000.0);

  Char_t histo_title1[20];
  sprintf(histo_title1,"recon_angle1_array_%d",k);
  angle1[k] = new TH2F(histo_title1, histo_title1, 100,−3.14159/2 ,3.14159/2, 100, −3.14159/2, 3.14159/2);

  Char_t histo_title2[20];                                              100
  sprintf(histo_title2,"recon_angle2_array_%d",k);
  angle2[k] = new TH2F(histo_title2, histo_title2, 100, −3.14159/2,3.14159/2, 100, −3.14159/2,3.14159/2);

  histo_num = histo_num + 1;

  //define the arrays for energy, position, and direction
  //————————————————————————-
  int event_no[no_events];
  double energy[no_events];
                                                                        110
  double positionx[no_events];
  double positiony[no_events];
  double positionz[no_events];

  double directionx[no_events];
  double directiony[no_events];
  double directionz[no_events];

  for(int i = 1; i<=10; i++){//file loop
                                                                        120
    array_num = 0;

    //open log file and check that it exists
    //————————————————-
    Char_t logname[20];
    sprintf(logname, "array%d_%d.log",k,i);

    ifstream arrayfile(logname);
    if (arrayfile.fail()) {
      cout << "Array data file " << logname << " NOT FOUND" << endl;     130
      return 1;
```

```
}

cout << logname << endl;

//read in log file
//------
arrayfile >> temp;
arrayfile >> depth;
arrayfile >> salinity;
arrayfile >> velocity;
arrayfile >> coef;
arrayfile >> heatcap;

arrayfile >> no_hydro;

const int no_hydros = no_hydro;

for(int hydro_num = 1; hydro_num <= no_hydros; hydro_num++){
  arrayfile >> hy_locationx[hydro_num];
  arrayfile >> hy_locationy[hydro_num];
  arrayfile >> hy_locationz[hydro_num];
}

event_num = 0;

//read in event information
//----------
while (!arrayfile.eof() && event_num<no_event) {

  int thiseventnumber;
  arrayfile >> thiseventnumber;

  // If we've read past the end of the file, bail out now :
  if(arrayfile.eof()) break;

  // Otherwise, this is a successful event read :
  event_num++;

  event_no[event_num]=thiseventnumber;

  arrayfile >> energy[event_num];

  arrayfile >> positionx[event_num];
  arrayfile >> positiony[event_num];
  arrayfile >> positionz[event_num];

  arrayfile >> directionx[event_num];
  arrayfile >> directiony[event_num];
  arrayfile >> directionz[event_num];

  //for each event, loop through all hydrophones
  //----------
  for(int hydro_num = 1; hydro_num<= no_hydros ; hydro_num++){

    array_num = array_num+1;

    arrayfile >> radius[array_num];
    arrayfile >> angle[array_num];
```

```
    arrayfile >> peak_pres[array_num];
    arrayfile >> peak_time[array_num];
  }
}

//=============================================================================

double theta1 = 0;
double theta2 = 0;
double sin_theta1 = 0;
double sin_theta2 = 0;

//reset values
//-----
counter=0;
cos1 = 0;
cos2 = 0;
sin1 = 0;
sin2 = 0;
simonr_temp = 0;
hydro_sep1 = 0;
hydro_sep2 = 0;

//loop over number of events
for(int j = 1; j <= 1000; j++){//event loop

  //reset all values per loop
  //---------
  max_peak1 = 0;
  max_hydro1 = 0;
  max_hydro2 = 0;
  max_hydro3 = 0;
  time_hydro1 = 0;
  time_hydro2 = 0;
  time_hydro3 = 0;
  hydro_x1 = 0;
  hydro_y1 = 0;
  act_radius = 0;
  coinc_counter = 1;
  hydro1 = 0;
  hydro2 = 0;
  hydro3 = 0;
  time1 = 0;
  time2 = 0;
  time3 = 0;
  act_angle1 = 0;
  act_angle2 = 0;
  counter1 = 0;
  counter2 = 0;
  counter3 = 0;

  //loop over number of hydrophones per event
  for(int i = 1; i <= 100; i++){

    //make a counter for array of times
    counter = counter +1 ;

    //find max peak and use this as a reference hydrophone
```

```
  if(peak_pres[counter] > max_peak1){
    max_peak1 = peak_pres[counter];
    time_hydro1 = peak_time[counter];                                    250
    max_hydro1 = i;
    counter1 = counter;
  }
}

//subtract 100 from counter because of the above loop
//————————————————————
counter = counter − 100;

//code to find three maximum pressures                                   260
//————————————————
//      for(int i = 1; i <= 100; i++){

//        //make a counter for array of times
//        counter = counter +1 ;

//        //find the next highest peak and use this as hydrophone 2
//        if((peak_pres[counter] > max_peak2) && (peak_pres[counter] < max_peak1)){
//          max_peak2 = peak_pres[counter];
//          max_hydro2 = i;                                              270
//        }
//      }

//      for(int i = 1; i <= 100; i++){

//        //make a counter for array of times
//        counter = counter +1 ;

//        //find the next highest peak and use this as hydrophone 3
//        if((peak_pres[counter] > max_peak3) && (peak_pres[counter] < max_peak2)){   280
//          max_peak3 = peak_pres[counter];
//          max_hydro3 = i;
//        }
//      }


//code to find three hydrophones on the same string as the max pressure
//————————————————————————————
hydro_x1 = hy_locationx[max_hydro1];
hydro_y1 = hy_locationy[max_hydro1];                                     290

for (int i = 1; i <= 100; i++){

  counter = counter + 1;

  //find second hydro
  //————————
  if((coinc_counter == 1) && ((hy_locationx[i]==hydro_x1) && (hy_locationy[i]==hydro_y1))&&(peak_pres[counter]!= 0.0
    max_hydro2 = i;
    time_hydro2 = peak_time[counter];                                    300
    act_radius = radius[counter];
    coinc_counter = coinc_counter + 1;
    counter2 = counter;
  }
```

```
  //find third hydro
  //-------
  if((coinc_counter == 2) && ((hy_locationx[i]==hydro_x1) && (hy_locationy[i]==hydro_y1))&&(peak_pres[counter]!= 0.0
    max_hydro3 = i;
    time_hydro3 = peak_time[counter];                                    310
    coinc_counter = coinc_counter +1;
    counter3 = counter;
    //let loop run to end to make sure correct counter value
  }
}

//make sure that there are three hydrophones in coincidence
//--------------------------
if(coinc_counter == 3){//radius_if
  hydro_sep1 = 0;                                                        320
  hydro_sep2 = 0;

  //arrange hydrophones in ascending order
  //---------------
  if(hy_locationz[max_hydro2] > hy_locationz[max_hydro1]){
    hydro1 = max_hydro1;
    hydro2 = max_hydro2;
    hydro3 = max_hydro3;
    time1 = time_hydro1;
    time2 = time_hydro2;                                                 330
    time3 = time_hydro3;
    counter1 = counter1;
    counter2 = counter2;
    counter3 = counter3;
  }

  if(hy_locationz[max_hydro1] >hy_locationz[max_hydro2]){
    hydro1 = max_hydro2;
    hydro2 = max_hydro3;
    hydro3 = max_hydro1;                                                 340
    time1 = time_hydro2;
    time2 = time_hydro3;
    time3 = time_hydro1;
    counter1 = counter2;
    counter2 = counter3;
    counter3 = counter1;

    if(hy_locationz[max_hydro3] > hy_locationz[max_hydro1]){
      hydro2 = max_hydro1;
      hydro3 = max_hydro3;                                               350
      time2 = time_hydro1;
      time3 = time_hydro3;
      counter2 = counter1;
      counter3 = counter3;
    }

  }

  //reconstruct radius
  //--------                                                             360

  //hydro separations
  //-------
```

```
hydro_sep1 = (pow(pow(hy_locationx[hydro1]-hy_locationx[hydro2],2)+
                  pow(hy_locationy[hydro1]-hy_locationy[hydro2],2)+
                  pow(hy_locationz[hydro1]-hy_locationz[hydro2],2),0.5));

hydro_sep2 = (pow(pow(hy_locationx[hydro2]-hy_locationx[hydro3],2)+
                  pow(hy_locationy[hydro2]-hy_locationy[hydro3],2)+
                  pow(hy_locationz[hydro2]-hy_locationz[hydro3],2),0.5));

//reset values
theta1 = 0;
theta2 = 0;

//calculate theta1 and theta2
//------------
sin_theta1 = (velocity*(time1-time2)/hydro_sep1);
sin_theta2 = (velocity*(time2-time3)/hydro_sep2);

//to avoid nans, make sure sin theta never goes > or < 1
if((sin_theta1)>1.0){
  sin_theta1 = 1.0;
}
if((sin_theta2)>1.0){
  sin_theta2 = 1.0;
}
if((sin_theta1)<-1.0){
  sin_theta1 = -1.0;
}
if((sin_theta2)<-1.0){
  sin_theta2 = -1.0;
}

//calculate theta
//------
theta1 = asin((sin_theta1));
theta2 = asin((sin_theta2));

//calculate the actual angle from log file
//----------------
act_angle1 = atan( (positionz[int(counter1/100)+1]-hy_locationz[hydro1]) /
                   pow(pow(positionx[int(counter1/100)+1]-hy_locationx[hydro1],2)+
                       pow(positiony[int(counter1/100)+1]-hy_locationy[hydro1],2),0.5));

act_angle2 = atan((positionz[int(counter2/100)+1]-hy_locationz[hydro2]) /
                  pow(pow(positionx[int(counter2/100)+1]-hy_locationx[hydro2],2)+
                      pow(positiony[int(counter2/100)+1]-hy_locationy[hydro2],2),0.5));

cos1 = cos(theta1);
cos2 = cos(theta2);
sin1 = sin(theta1);
sin2 = sin(theta2);

//calculate radius
//------
simonr_temp = -(hydro_sep1/sin2)/(cos1/cos2 - sin1/sin2);

if(hydro_sep1 == hydro_sep2){
  histo[k]->Fill( act_radius,simonr_temp);
  angle1[k]->Fill(act_angle1 ,theta1);
```

```
        }

        angle2[k]->Fill(act_angle1 ,theta1);

      }
    }
  }

  recon_anal->cd();
  histo[k]->Write();
  angle1[k]->Write();
  angle2[k]->Write();

}

recon_anal->Close();

    }
```

## B.21 recon_plotter.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//03/03/03                                                     //
//                                                             //
//Plot the results of recon code                               //
//=============================================================//

#include <stdio.h>
#include <iostream>
#include "TROOT.h"
#include "TStyle.h"
#include "TH1.h"
#include "TF1.h"
#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TChain.h"
#include "TCanvas.h"

void recon_plotter() {

  // Clean up :
  // ==========
  gROOT->Reset();

  // No stats :
  // ==========
  gStyle->SetOptStat(0);
  gROOT->SetStyle("Plain");

  c1 = new TCanvas("c1","neutrinosim",200,10,700,500);

  c1->SetFillColor(0);
  c1->SetGridx();
  c1->SetGridy();
```

```
//=============================================================================

//Loop through all hadronic files extracting histograms
//--------------------                                                      40

TFile *pulse[8];
TH1F *histo[8];

//extract pulses from the files
//-----------
 pulse[1] = new TFile("array_dist_recon_anal.root");
 histo[1] = (TH1F*) pulse[1]->Get("recon_array_time_1_0");
 pulse[2] = new TFile("array_dist_recon_anal.root");
 histo[2] = (TH1F*) pulse[2]->Get("recon_array_time_1_1");                 50
 pulse[3] = new TFile("array_dist_recon_anal.root");
 histo[3] = (TH1F*) pulse[3]->Get("recon_array_time_1_2");
 pulse[4] = new TFile("array_dist_recon_anal.root");
 histo[4] = (TH1F*) pulse[4]->Get("recon_array_time_1_3");
 pulse[5] = new TFile("array_dist_recon_anal.root");
 histo[5] = (TH1F*) pulse[5]->Get("recon_array_time_1_4");
 pulse[6] = new TFile("array_dist_recon_anal.root");
 histo[6] = (TH1F*) pulse[6]->Get("recon_array_time_1_5");
 pulse[7] = new TFile("array_dist_recon_anal.root");
 histo[7] = (TH1F*) pulse[7]->Get("recon_array_time_1_7");                 60

 //set graph properties
 //---------
 histo[1]->SetLineWidth(1.0);
 histo[1]->SetLineStyle(1);

 histo[2]->SetLineWidth(1.0);
 histo[2]->SetLineStyle(1);

 histo[3]->SetLineWidth(1.0);                                             70
 histo[3]->SetLineStyle(1);

 histo[4]->SetLineWidth(1.0);
 histo[4]->SetLineStyle(1);

 histo[5]->SetLineWidth(1.0);
 histo[5]->SetLineStyle(1);

 histo[6]->SetLineWidth(1.0);
 histo[6]->SetLineStyle(1);                                               80

 histo[7]->SetLineWidth(1.0);
 histo[7]->SetLineStyle(1);

 histo[1]->SetLineColor(1);
 histo[2]->SetLineColor(1);
 histo[3]->SetLineColor(1);
 histo[4]->SetLineColor(1);
 histo[5]->SetLineColor(1);
 histo[6]->SetLineColor(1);                                               90
 histo[7]->SetLineColor(1);

 histo[7]->SetTitle("");
```

```
  histo[7]->GetXaxis()->SetTitle("Actual dist / m");
  histo[7]->GetXaxis()->CenterTitle();
  histo[7]->GetXaxis()->SetTitleOffset(1.2);
  histo[7]->GetYaxis()->SetTitle("Reconstructed dist / m");
  histo[7]->GetYaxis()->SetTitleFont(12);
  histo[7]->GetYaxis()->SetTitleSize(0.06);
  histo[7]->GetYaxis()->SetTitleOffset(0.8);                                    100
  histo[7]->GetYaxis()->CenterTitle();

  //draw
  //—-
 histo[7]->Draw();

      }
```

## B.22 recon_time_scatter.C

```
//=============================================================//
//Created by Simon Bevan                                       //
//30/01/04                                                     //
//                                                             //
//Root macro for testing reconstructed dist against actual     //
//                                                             //
//=============================================================

#include "TROOT.h"
#include <stdio.h>                                                          10
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TH1F.h"
#include "TFile.h"
#include "TH2F.h"                                                           20
#include "rangauss.hh"
#include "TRandom3.h"

//=============================================================================

TH2F *histo[10];
TH2F *time_scattered[10];

int main(int argc, char *argv[]){
                                                                            30
  //define some of the variables
  //——————————
  double temp = 0;
  double depth = 0;
  double salinity = 0;
  double velocity = 0;
  double coef = 0;
  double heatcap = 0;
```

```
int no_hydro = 0;

int counter = 0;

double hy_locationx[100110]; double hy_locationy[100110]; double hy_locationz[100110];

double radius[100110];
double angle[100110];
double peak_pres[100110];
double peak_time[100110];
int histo_num = 0;

int event_num=0;
int array_num=0;

double theta1 = 0;
double theta2 = 0;

double cos1 = cos(theta1);
double cos2 = cos(theta2);
double sin1 = sin(theta1);
double sin2 = sin(theta2);

double simonr_temp = 0;

double hydro_sep1 = 0; double hydro_sep2 = 0;
double max_peak1 = 0;
int max_hydro1 = 0; int max_hydro2 = 0; int max_hydro3 = 0;
double hydro_x1 = 0;double hydro_y1 = 0;
int hydro1 = 0; int hydro2 = 0; int hydro3 = 0;
int coinc_counter = 1;
double act_radius = 0;
double time_hydro1 = 0; double time_hydro2 = 0; double time_hydro3 = 0;
double time1 = 0; double time2 = 0; double time3 = 0;

double act_angle1= 0;double act_angle2= 0;
double counter1 = 0; double counter2 = 0; double counter3 = 0;
double width = 0;

//number of events per file
int no_event = 1000;

TRandom3 ranGen;
ranGen.SetSeed(4);

// Make output file :
// ———————
Char_t filename[20];
sprintf(filename,"array_dist_recon_anal.root");
TFile* recon_anal = new TFile(filename,"RECREATE","neutrinosim histos");

for(int k=1; k <= 1; k++){//array loop

  for(int t = 0; t<=7; t++){//scatter loop

    width = 0;

    //create histos for analysis
```

```
//————————————
const int no_events = no_event + 1;
                                                                    100
Char_t histo_title[20];
sprintf(histo_title,"recon_array_%d",k);
histo[k] = new TH2F(histo_title, histo_title, 1000, 0.0,10000.0, 1000, 0.0,10000.0);

Char_t histo_title1[20];
sprintf(histo_title1,"recon_array_time_%d_%u",k,t);
time_scattered[t] = new TH2F(histo_title1, histo_title1, 1000, 0.0,10000.0, 1000, 0.0,10000.0);

//make values for the time scatter
//————————————-                                                    110
if(t==0){
  width = 0.0;
}
if(t==1){
  width = 5E-6;
}
if(t==2){
  width = 5E-5;
}
if(t==3){                                                           120
  width = 5E-4;
}
if(t==4){
  width = 5E-3;
}
if(t==5){
  width = 5E-2;
}
if(t==6){
  width = 1E-6;                                                     130
}
if(t==7){
  width = 3E-5;
}

cout << width << endl;

histo_num = histo_num + 1;

//define the arrays for energy, positio, and direction              140
//——————————————————————-
int event_no[no_events];
double energy[no_events];

double positionx[no_events];
double positiony[no_events];
double positionz[no_events];

double directionx[no_events];
double directiony[no_events];                                       150
double directionz[no_events];

for(int i = 1; i<=50; i++){//file loop

  array_num = 0;
```

```
//open log file and check that it exists
//————————————————-

Char_t logname[20];                                                    160
sprintf(logname, "array%d_%d.log",k,i);

ifstream arrayfile(logname);
if (arrayfile.fail()) {
  cout << "Array data file " << logname << " NOT FOUND" << endl;
  return 1;
}

cout << logname << endl;
                                                                       170
//read in log file
//———————
arrayfile >> temp;
arrayfile >> depth;
arrayfile >> salinity;
arrayfile >> velocity;
arrayfile >> coef;
arrayfile >> heatcap;

arrayfile >> no_hydro;                                                 180

const int no_hydros = no_hydro;

for(int hydro_num = 1; hydro_num <= no_hydros; hydro_num++){
  arrayfile >> hy_locationx[hydro_num];
  arrayfile >> hy_locationy[hydro_num];
  arrayfile >> hy_locationz[hydro_num];
}

event_num = 0;                                                         190

//read in event information
//——————————
while (!arrayfile.eof() && event_num<no_event) {

  int thiseventnumber;
  arrayfile >> thiseventnumber;

  // If we've read past the end of the file, bail out now :
  if(arrayfile.eof()) break;                                           200

  // Otherwise, this is a successful event read :
  event_num++;

  event_no[event_num]=thiseventnumber;

  arrayfile >> energy[event_num];

  arrayfile >> positionx[event_num];
  arrayfile >> positiony[event_num];                                   210
  arrayfile >> positionz[event_num];

  arrayfile >> directionx[event_num];
```

```
  arrayfile >> directiony[event_num];
  arrayfile >> directionz[event_num];

  //for each event, loop through all hydrophones
  //--------------------
  for(int hydro_num = 1; hydro_num<= no_hydros ; hydro_num++){
                                                                                220
    array_num = array_num+1;

    arrayfile >> radius[array_num];
    arrayfile >> angle[array_num];
    arrayfile >> peak_pres[array_num];
    arrayfile >> peak_time[array_num];

  }
}
                                                                                230
//=============================================================================

double theta1 = 0;
double theta2 = 0;
double sin_theta1 = 0;
double sin_theta2 = 0;

//reset values
//-----
counter=0;                                                                      240
cos1 = 0;
cos2 = 0;
sin1 = 0;
sin2 = 0;
simonr_temp = 0;
hydro_sep1 = 0;
hydro_sep2 = 0;

//loop over number of events
for(int j = 1; j <= 1000; j++){//event loop                                     250

  //reset all values per loop
  //----------
  max_peak1 = 0;
  max_hydro1 = 0;
  max_hydro2 = 0;
  max_hydro3 = 0;
  time_hydro1 = 0;
  time_hydro2 = 0;
  time_hydro3 = 0;                                                              260
  hydro_x1 = 0;
  hydro_y1 = 0;
  act_radius = 0;
  coinc_counter = 1;
  hydro1 = 0;
  hydro2 = 0;
  hydro3 = 0;
  time1 = 0;
  time2 = 0;
  time3 = 0;                                                                    270
  act_angle1 = 0;
```

```
act_angle2 = 0;
counter1 = 0;
counter2 = 0;
counter3 = 0;

//loop over number of hydrophones per evet
for(int i = 1; i <= 100; i++){

  //make a counter for array of times                                    280
  counter = counter +1 ;

  //find max peak and use this as a reference hydrophone
  if(peak_pres[counter] > max_peak1){
    max_peak1 = peak_pres[counter];
    time_hydro1 = peak_time[counter];
    max_hydro1 = i;
    counter1 = counter;

  }                                                                    290
}

//subtract 100 from counter because of the above loop
//————————————————————
counter = counter - 100;


//code to find three hydrophones on the same string as the max pressure
//————————————————————————————
hydro_x1 = hy_locationx[max_hydro1];                                   300
hydro_y1 = hy_locationy[max_hydro1];

for (int i = 1; i <= 100; i++){

  counter = counter + 1;

  //find second hydro
  //———————
  if((coinc_counter == 1) && ((hy_locationx[i]==hydro_x1) && (hy_locationy[i]==hydro_y1))&&(peak_pres[counter]!= 0
    max_hydro2 = i;                                                    310
    time_hydro2 = peak_time[counter];
    act_radius = radius[counter];
    coinc_counter = coinc_counter + 1;
    counter2 = counter;
  }

  //find third hydro
  //———————-
  if((coinc_counter == 2) && ((hy_locationx[i]==hydro_x1) && (hy_locationy[i]==hydro_y1))&&(peak_pres[counter]!= 0
    max_hydro3 = i;                                                    320
    time_hydro3 = peak_time[counter];
    coinc_counter = coinc_counter +1;
    counter3 = counter;
    //let loop run to end to make sure correct counter value
  }
}

//make sure that there are three hydrophones in coincidence
//————————————————————————
```

```
if(coinc_counter == 3){//radius_if                                    330
  hydro_sep1 = 0;
  hydro_sep2 = 0;

  //arrange hydrophones in ascending order
  //-----------------------
  if(hy_locationz[max_hydro2] > hy_locationz[max_hydro1]){
    hydro1 = max_hydro1;
    hydro2 = max_hydro2;
    hydro3 = max_hydro3;
    time1 = time_hydro1;                                             340
    time2 = time_hydro2;
    time3 = time_hydro3;
    counter1 = counter1;
    counter2 = counter2;
    counter3 = counter3;
  }

  if(hy_locationz[max_hydro1] >hy_locationz[max_hydro2]){
    hydro1 = max_hydro2;
    hydro2 = max_hydro3;                                             350
    hydro3 = max_hydro1;
    time1 = time_hydro2;
    time2 = time_hydro3;
    time3 = time_hydro1;
    counter1 = counter2;
    counter2 = counter3;
    counter3 = counter1;

    if(hy_locationz[max_hydro3] > hy_locationz[max_hydro1]){
      hydro2 = max_hydro1;                                           360
      hydro3 = max_hydro3;
      time2 = time_hydro1;
      time3 = time_hydro3;
      counter2 = counter1;
      counter3 = counter3;
    }

  }

  //reconstruct radius                                               370
  //-------

  //scatter the time using rangauss header
  //----------------------
  time1 = rangauss(time1,width,&ranGen);
  time2 = rangauss(time2,width,&ranGen);
  time3 = rangauss(time3,width,&ranGen);

  //the hydrophone separations
  //----------                                                       380
  hydro_sep1 = (pow(pow(hy_locationx[hydro1]-hy_locationx[hydro2],2)+
                  pow(hy_locationy[hydro1]-hy_locationy[hydro2],2)+
                  pow(hy_locationz[hydro1]-hy_locationz[hydro2],2),0.5));

  hydro_sep2 = (pow(pow(hy_locationx[hydro2]-hy_locationx[hydro3],2)+
                  pow(hy_locationy[hydro2]-hy_locationy[hydro3],2)+
                  pow(hy_locationz[hydro2]-hy_locationz[hydro3],2),0.5));
```

```
        //reset values
        theta1 = 0;
        theta2 = 0;

        //calculate theta1 and theta2
        //——————————
        sin_theta1 = (velocity*(time1−time2)/hydro_sep1);
        sin_theta2 = (velocity*(time2−time3)/hydro_sep2);

        //to avoid nans, make sure sin theta never goes > or < 1
        if((sin_theta1)>1.0){
          sin_theta1 = 1.0;
        }
        if((sin_theta2)>1.0){
          sin_theta2 = 1.0;
        }
        if((sin_theta1)<−1.0){
          sin_theta1 = −1.0;
        }
        if((sin_theta2)<−1.0){
          sin_theta2 = −1.0;
        }

        //calculate theta
        //——————
        theta1 = asin((sin_theta1));
        theta2 = asin((sin_theta2));

        //calculate the actual angle from the log file
        //————————————————————
        act_angle1 = atan( (positionz[int(counter1/100)+1]−hy_locationz[hydro1]) /
                          pow(pow(positionx[int(counter1/100)+1]−hy_locationx[hydro1],2)+
                              pow(positiony[int(counter1/100)+1]−hy_locationy[hydro1],2),0.5));

        act_angle2 = atan((positionz[int(counter2/100)+1]−hy_locationz[hydro2]) /
                         pow(pow(positionx[int(counter2/100)+1]−hy_locationx[hydro2],2)+
                             pow(positiony[int(counter2/100)+1]−hy_locationy[hydro2],2),0.5));

        cos1 = cos(theta1);
        cos2 = cos(theta2);
        sin1 = sin(theta1);
        sin2 = sin(theta2);

        //calculate radius
        //———————-
        simonr_temp = −(((hydro_sep1+hydro_sep2)/2)/sin2)/(cos1/cos2 − sin1/sin2);

        time_scattered[t]->Fill(act_radius ,simonr_temp);

      }
    }
  }

  recon_anal->cd();
  time_scattered[t]->Write();

}
```

```
  }

  recon_anal->Close();

      }                                                                    450
```

## B.23 rateCalc.C.tex

```
//————————————————————-//
// Created by Simon Bevan 17/03/04      //
// Root macro to count the number of events //
// in an energy range and then give a rate //
// calculation                          //
//————————————————————-//

#include "TROOT.h"
#include <stdio.h>
#include <iostream.h>                                                      10
#include <fstream.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <vector>

#include "TFile.h"
#include "TNtuple.h"
#include "TLeaf.h"
#include "TH1F.h"                                                          20
#include "TH2F.h"
#include "TRandom3.h"
#include "TVector3.h"


double cs(double energy) {

  // v-N cross section. From Terry Sloan and a ruler.
  // log10(sigma/cm^2) = -32.60 + (0.362*(log10(E/GeV)-7))
  // INPUT : energy in eV                                                  30
  // OUTPUT : cross section in cm^2
  double egev = energy/1E09;
  double log10cs = -32.60 + (0.362*(log10(egev)-7.0));
  return pow(10,log10cs);
}

using std::cout;
using std::vector;


                                                                          40
int main(int argc, char *argv[]) {

  int    nCS    = 1000000;
  double eMax = 1E21;
  double eMin = 1E20;
  double eMinGeV = eMin/1E09;
  double eMaxGeV = eMax/1E09;
  double wbsum = 0.0;
```

```
double ess_sum = 0.0;
                                                                    50
 TRandom3 ranGen;
 ranGen.SetSeed(89);

   for (int i = 0; i<nCS; ++i) {

    // Waxman-Bahcall rate estimate :
    // ————————————
    // (should use a smarter prior than uniform in energy)
    double ewb = (ranGen.Rndm()*(eMaxGeV-eMinGeV))+eMinGeV; // energy
    double wbf = 2.0E-8/(pow(ewb,2)); // flux                        60
    wbsum += wbf*cs(ewb*1E09);

    // GZK rate estimate :
    // ————————-
    // Use following fit for the ESS GZK flux in the log10(E^2*flux) vs. log10(E[GeV]) plane :
    // *** only valid in the range 8<log10(E[GeV])<12.4 ***
    double pa0 = -2.32762e+02;
    double pa1 = 6.73384e+01;
    double pa2 = -4.93946e+00;
    double pa3 = -3.26750e-01;                                      70
    double pa4 = 5.69413e-02;
    double pa5 = -1.96822e-03;

    double log10egev = log10(ewb);
       double ess_flux =
       pa0 +
      pa1*log10egev +
      pa2*pow(log10egev,2) +
      pa3*pow(log10egev,3) +
      pa4*pow(log10egev,4) +                                        80
      pa5*pow(log10egev,5);
    double relweight = pow(10,ess_flux)/(2.0E-08);
    ess_sum += relweight*wbf*cs(ewb*1E09);
   }
  wbsum = wbsum/nCS; // average
  wbsum = wbsum*(eMaxGeV-eMinGeV); // integral

  // Multiply by number of nucleons of water in a cm^3 :
  // ———————————————————————
  wbsum *= 6.0E23;                                                  90

  printf(" eMin = %e \n",eMin);
  printf(" eMax = %e \n",eMax);
  printf(" Integrated Waxman-Bahcall count rate in water = %e cm-3 sec-1 sr-1 \n",wbsum);

  // Same for GZK rate estimate :
  // ————————————-
  ess_sum = ess_sum/nCS;
  ess_sum = ess_sum*(eMaxGeV-eMinGeV);
  ess_sum *= 6.0E23;                                                100
  printf(" Integrated GZK (ESS) count rate in water = %e cm-3 sec-1 sr-1 \n",ess_sum);


  //calculate number of events in a given energy range
  //——————————————————————-
  double sum = 0;
```

```
TFile *pulse[1];
TH1F *histo[1];

//input array you want to calculate the rate for
//--------------------
pulse[1] = new TFile("array_5_anal_50000_comp.root");
histo[1] = (TH1F*) pulse[1]->Get("array_5_pres_cut_0.025000_energy_3");

//sum over given energy region (here E > 1E20)
for (int i = 60; i <=80; i++){
  sum = sum + histo[1].GetBinContent(i);
}

//calculate rate
//------

double rate = 0;
rate = sum/ess_sum;

cout << "the rate is " << rate;

   }
```

## B.24 rangauss.hh.tex

```
#include <math.h>
#include "TRandom3.h"

double rangauss(double mean, double sigma, TRandom3* ranGen) {

  // From Numerical Recipes.
  // Two Gaussian random numbers are generated at once,
  // hence the following static state information :
  static int iset=0;
  static double gset;
  double fac,rsq,v1,v2;

  // if (iset == 0) {
    // Must regenerate :
    do {
      v1 = 2.0*(ranGen->Rndm())-1.0;
      v2 = 2.0*(ranGen->Rndm())-1.0;
      rsq = v1*v1+v2*v2;
    } while (rsq >= 1.0 || rsq == 0.0);

    fac=sqrt(-2.0*log(rsq)/rsq);
    gset=(v1*fac)*sigma+mean;
    iset=1;
    return (v2*fac)*sigma+mean;
    //} else {
    //iset=0;
    // return gset;
    //}
```

```
}
```