Top Quark Physics: Past, Present and Future

by

Sarah D. Groulx

Department of Physics and Astronomy

University of Victoria

PO BOX 3055 STN CSC

Victoria BC V8W 3P6

Physics Co-op Work Term Report

In partial fulfillment

of the requirements of the Physics Co-op Program

Summer 1998

Sarah D. Groulx

Department of Physics and Astronomy

University of Victoria

Abstract

The top quark, discovered in 1995, eluded physicists for more than a decade. Though some work has been done at FermiLab to measure the properties of top, there is still much to learn about top. Many suspect understanding top will lead to understanding the nature of mass. The Large Hadron Collider (LHC) at CERN will be used to study the top quark, as it will produce top quarks at a far higher rate than the Tevatron Accelerator at FermiLab, where top was discovered. Currently Monte Carlo generators are being used to study top and learn techniques to isolated top events at the LHC.

Contents

List of Figures

List of Tables

1 Introduction

Particle physics is the study of the fundamental constituents of matter and their interactions. The discovery that the fundamental constituents of matter are actually composed of yet smaller constituents is a familiar pattern in the history of particle physics. During the 19th century, atoms were established as the elementary components from which molecules are constructed. The atom was subsequently discovered, during the first 30 years of this century, to be composed of quantum bound states of negatively charged electrons orbiting a positively charged nucleus. These are held together by the electromagnetic force due to the exchange of virtual photons. During the 1930s and 1940s, the nucleus of the atom was found to be composed of protons and neutrons held together by the strong interaction force due to the exchange of particles called mesons. In the 1950s and 1960s, experiments involving the scattering of these, supposedly elementary nuclear constituents, at high energies produced very large numbers of other, similar, strongly interacting particles. This led to the conclusion that protons and neutrons are simply the lightest members of a very large family of particles called baryons. Baryons and mesons form a group of particles influenced by the strong interaction, called hadrons.

In the early 1960s, the suggestion was made that hadrons could conveniently be regarded as composites of three more basic objects, called quarks. These were the up, down and strange quarks. At the time, quarks were little more than a convenient mathematical tool to model the properties of hadrons since free quarks had never been seen. Experiments at Stanford in the late 1960's to probe the structure of the proton revealed that protons did indeed contain point-like constituents. These are now recognized as being the quarks of the original quark model. Quarks are influenced by the strong interaction mediated by the exchange of gauge bosons called gluons. Gluons are responsible for holding quarks together in hadrons, and hadrons together in the nucleus. During the 1970s, colliding beam experiments implied the existence of three further quarks. These were charmed, bottom and top quarks. While charmed and bottom were discovered with relative ease, top eluded particle physicists for more than a decade.

Top was discovered at FermiLab in 1995. Although many properties of top have been measured at FermiLab, higher energies are needed for more precise measurements and larger top event samples. The Large Hadron Collider (LHC) at CERN, which will begin taking data in 2005, will provide 14TeV collision energies, high enough to produce one top event approximately every second. Physicists are preparing for top studies at LHC.

2 The Standard Model of Particle Physics

The modern theory of particle physics, called the Standard Model, attempts to explain all the observed phenomena of particle physics in terms of the properties and interactions of a small number of particles of three distinct types. These are leptons and quarks, which are spin $\frac{1}{2}$ fermions[1] and the $W^{\pm}$, Z^{o}, γ (photon), and g (gluon) gauge bosons, which are spin 1 bosons[2]. Gauge bosons act as the force carriers in the theory. In the Standard Model, these particles are all assumed to be elementary; that is, they are assumed to be point-like particles without internal structure or excited states. They are indeed point-like to the accuracy of present experimental data.

Leptons carry integral electric charges, 0 or $\pm|e|$, and three types of both neutral and charged leptons are known. The neutral leptons are called neutrinos, and have very small (possibly zero) mass. The most familiar example of a lepton is the electron e^{-}. Together with its associated neutrino, ν_e, these form a 'generation' of leptons. The Standard Model requires that leptons and

[1]Fermions are particles with odd half integral spin and are described by Fermi-Dirac statistics.[1] (spin is intrinsic angular momentum and spin $\frac{1}{2} \equiv$ spin $\frac{1}{2}\hbar$ and recall $\hbar \equiv \frac{h}{2\pi}$)

[2]Bosons are particles with integral spin and are described by Bose-Einstein statistics.[1]

Flavour	Mass (GeV/c^2)	Q	S	C	B	T
Up, u	$\sim$ 350	$-\frac{1}{3}$	0	0	0	0
Down, d	$\sim$ 350	$+\frac{2}{3}$	0	0	0	0
Strange, s	$\sim$ 550	$-\frac{1}{3}$	-1	0	0	0
Charmed, c	$\sim$ 1800	$+\frac{2}{3}$	0	$+1$	0	0
Bottom, b	$\sim$ 4500	$-\frac{1}{3}$	0	0	-1	0
Top, t	$\sim$ 20000	$+\frac{2}{3}$	0	0	0	$+1$

Table 1: The flavour quantum numbers of quarks: Q is charge; S is strangeness; C is charm; B is the bottom quantum number; and, T is top quantum number.

quarks each come in pairs, called generations. There are three known generations of leptons. The second generation contains the muon μ^- and its associated neutrino, ν_μ. The μ^- appears to be identical to the electron with the exception that it is approximately 207 times heavier, and is unstable with a lifetime of $\approx 2.2 \times 10^{-6}$s. The third generation is composed of the τ lepton and its associated neutrino, ν_τ. Charged leptons experience the electromagnetic and weak interactions, while neutrinos only experience the weak interaction because they are electrically neutral. The Standard Model also requires that each particle have an antiparticle (the concept of antiparticles is discussed in Appendix A). Thus in addition to the lepton pairs above, each generation contains the antiparticles of those leptons. This is also true for quarks.

Quarks carry fractional electric charges of $+\frac{2}{3}e$ and $-\frac{1}{3}e$. There are six known types or 'flavours' of quark, distinguished by the assignment of internal quantum numbers. These are up u, down d, strange s, charmed c, bottom b, and top t. The u and d are the lightest quarks and have approximately equal masses. These form the first generation of quarks and make up the familiar protons and neutrons. Along with the electrons, u and d make up the entire periodic table. The heavier quarks (c, s, t and b quarks) and leptons, though abundant in the early moments after the big bang, are now commonly produced only in accelerators. The s-quark is assigned an internal quantum number called strangeness, with the value $S = -1$; the c-quark is assigned the charm quantum number $C = +1$; the b-quark is assigned the bottom quantum number $B = -1$; and the t-quark is assigned the top quantum number $T = +1$. In addition to these properties, quarks possess another attribute called the colour charge. There are three types of colour charge unlike the two types of electromagnetic charge (positive and negative charge), and it is for this reason that the colour charges were named after the three primary colours. Quarks can exist in red r, green g, or blue b colour states; while antiquarks have anticolour: antired $\bar{r}$, antigreen $\bar{g}$, or antiblue $\bar{b}$. In addition to the weak and electromagnetic interactions, quarks experience the strong interaction.

Hadrons are strongly interacting particles built from two types of quark combination:

$$\begin{aligned} \text{Baryon} &= QQQ \quad \text{three quarks} \\ \text{Meson} &= Q\bar{Q} \quad \text{quark-antiquark pair} \end{aligned} \tag{1}$$

The strong interactions between quarks also gives rise to the observed strong interactions between hadrons. Hadrons can only exist in states which are colourless, that is, states that have zero colour charge. This state is obtained with the combination of a red, green and blue quark for baryons, or with a quark and an antiquark of the same colour and anticolour (eg. r and $\bar{r}$) for mesons. Quarks which have non-zero colour charge can exist only within hadrons. This is called colour confinement and is the reason free quarks are not directly observable [3] [2].

There are four fundamental forces in nature, three of which are described in the Standard Model. These are the electromagnetic force, and the aforementioned strong and weak forces. Each interaction is associated with the exchange of spin 1 particles. The electromagnetic force acts only on particles with electric charge. It is mediated by the exchange of virtual photons which 'couple' to the electric charge. The weak interaction is mediated by the exchange of the massive $W^{\pm}$ and Z^{o} bosons which results in a short range force. For the strong interaction, massless gluons are exchanged. The theory which describes strong interactions in the Standard Model is called quantum chromodynamics (QCD). It describes the strong interaction, mediated by massless spin

1 gluons coupling to the quark colour charge. The different quark flavours have identical strong interactions because they exist in the same three colour states. The forces between hadrons are the residues of the forces between their quark constituents and are negligible when hadrons are far apart.

3 The Discovery of the Top Quark

The top quark is the sixth, and quite possibly the heaviest, quark. Physicists have known that the top must exist since 1977, when its partner, the bottom, was discovered. However, the top proved very difficult to find. According to the Standard Model, the top is a fundamental particle with no discernible structure; yet the top quark turns out to have a mass of 175 GeV, as much as an atom of gold. This is far greater than most theorists had anticipated.

Immense amounts of energy concentrated into a minute region of space are required to create a top quark. Experimenters hoped to create a top quark out of the energy from the impact of two particle beams. However, the amount of energy required to produce a top quark was unknown. Many properties of the top, such as its charge and spin, are predicted by the Standard Model. Although, its mass is not predicted directly by the Standard Model, physicists at CERN could limit the allowed mass range of the top.

Although particles can be created purely from energy, the electric charge must be conserved. At LHC and the Tevatron, the most likely way to make a top is along with its antiparticle, the antitop, which is identical in mass but with opposite signs for other properties. The conserved quantities, such as electric charge, thus cancel out.

Early searches for top at the Stanford Linear Accelerator Center and at DESY put limits on the top mass. Over the years the hunt moved on to different accelerators as they came into operation with ever more energetic particle beams. In the early 1980s at CERN, beams of protons and antiprotons with energies up to 315 GeV did not find top but produced two new particles, the W and the Z bosons. These discoveries provided further confirmation of the Standard Model, which had accurately predicted the W and Z masses. Many believed that the discovery of the top quark at CERN was imminent [19].

Finding top would still be challenging. When protons and antiprotons collide at high energies, the actual interaction is between their internal quarks and gluons. Each quark or gluon carries a fraction of the total energy of its proton. The collision between these quarks and gluons must have enough energy to generate two top quarks. Such collisions are rare, and the higher the energy required (the higher the top mass), the rarer they are.

The 1980s were brought to a close, after intense competition between CERN and FermiLab, with the knowledge that the top mass could be no lower than 77 GeV [19]. CERN had reached its limit; its collisions would be unlikely to create top quarks heavier than 77 GeV. The competition was now between two experiments on the Tevatron accelerator ring at FermiLab, CDF and D0.

By August 1992, the lower limit of the top mass had increased to 91 GeV. This was an important development. The W boson has a mass of 80.4 GeV [13] and mediates interactions between quarks in the same generation. If the top were light enough, below about 75 GeV, a W might have produced a top by decaying to it and an anti bottom. This new limit confirmed the best way of discovering top, at the Tevatron, was by creating a top-antitop pair.

The final presentations, made in March 1995, showed overwhelming evidence for the top quark from both CDF and D0. Both teams reported a probability of less than one in 500,000 that their top quark candidates could be explained by background alone.

At present, the FermiLab accelerator is being modified and improved and will resume taking data in 1999. Accelerator upgrades will allow top quarks to be produced at 20 times the previous rate. Detector upgrades will improve the efficiency of identifying top quarks. Tops will be found 30 times faster than previously, allowing a more detailed look at top characteristics. By 2005 the Large Hadron Collider (LHC) at CERN will begin operation. It will produce two proton beams colliding at 14 TeV centre of mass energies. This is seven times the energy at FermiLab and will generate almost one top-antitop pair per second.

3.1 How Top Was Discovered

Individual, or 'bare', quarks are never seen. When a quark emerges from a collision, it gets 'dressed up' by a cloud of other quarks and antiquarks. The strong force pulls quark antiquark pairs out of the vacuum to combine with an individual quark so that only colourless hadrons emerge. This process is called hadronization; the quark is hadronized. What is often observed is a jet, a directed beam of particles created by the many leftover quarks and, that has approximately the same direction of motion as the original quark. Unlike the up and down quarks, which are stable, the top quark has a lifetime of only about 10^{-24} seconds, so short that the top decays before it hadronizes. The Standard Model predicts that the top quark, if heavy enough, will decay nearly all the time into a W and a bottom quark. So a top and antitop, if created, should generate two Ws, a bottom and an antibottom. Unfortunately, both the W and the bottom quark are not directly observable. The W's lifetime is about the same as the top's. The bottom, though also unstable, is much longer lived than the top [13].

The W most likely decays into a quark and an antiquark from the same generation, such as an up and an antidown. In this case, the quark and antiquark are detected as two jets. The W can also decay into a charged and a neutral lepton from the same generation. If the charged lepton is an electron or muon, it can be directly observed in the detector. If it is a τ lepton, it decays quite rapidly, making it hard to identify. Neutrinos pass through the detector completely unobserved.

Fortunately, a neutrino's presence can be inferred because it carries away momentum. When the momenta of all the particles seen in the detector are added up, if a significant amount is missing, a neutrino is assumed to have carried it off.

The bottom quark travels in a jet as part of a meson or baryon. This hadron decays roughly half a millimeter from where it was generated. By finding most of the tracks in a jet and extrapolating them backward, it is possible to find the point where the bottom quark decayed, and thereby identify it as a bottom jet [12].

By adding up the energies in the jets and leptons produced by a (presumed) top-antitop pair, a value for the mass of the top could be measured. The derived masses of true top events should fall close to some one value, the true top mass. In contrast, background events should give a much broader distribution. It was found that the mass indeed clustered in a narrow range, implying a top mass of approximately 175 GeV. Once there was enough data, the top discovery was conclusive.

4 The ATLAS Detector at LHC

Evidence indicates that new physics, and answers to some very important and profound questions at present, lie at energies above 1 TeV [10]. To look for this new physics, CERN is developing the LHC. It is designed to use the 27-kilometre LEP tunnel, and be fed by existing particle sources and pre-accelerators. A challenging machine, the LHC will use the most advanced superconducting magnet and accelerator technologies ever employed.

The LHC is an accelerator which will bring protons into head-on collision at higher energies than ever achieved before. This will allow scientists to look further into the structure of matter and recreate the conditions prevailing in the Universe just 10^{-12} seconds after the 'Big Bang' when the temperature was 10^{16} degrees. The LHC can collide proton beams with energies around 7-on-

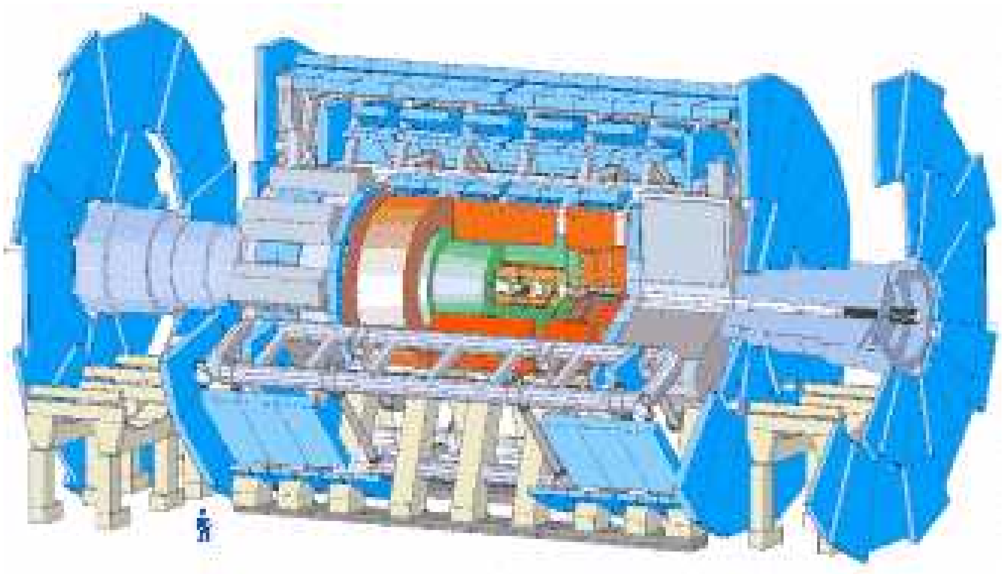


Figure 1: The ATLAS detector

7 TeV and beam crossing points of unsurpassed brightness, providing the experiments with high interaction rates [10].

The ATLAS detector is one of three detectors on LHC. The ATLAS experiment is being constructed by 1700 collaborators in 150 institutes around the world. ATLAS is designed to improve the fundamental understanding of matter and forces and will hopefully help physicists to understand the nature of mass.

The ATLAS detector consists of four major components. The inner tracker (yellow, see Figure 1) measures the paths of electrically charged particles. The inner tracker is placed is a strong magnetic field, produced by a cylindrical superconducting coil, so that charged particle paths are curves from which momentum, direction, and sign of electric charge can be determined. Semiconductor devices, providing position accuracy of 0.01mm, are surrounded by thousands of gas-filled straws, each having a wire down its axis and with high voltage between wire and straw wall. Charged particles traversing the straw induce electrical pulses that are recorded. The calorimeter (orange and green) measures the energies of charged and neutral particles. It consists of metal plates (absorbers) and sensing elements. Interactions of the incident particles with the absorbers create a 'shower', where the number of charged particles produced is proportional to the energy of the incident particle. The 'shower' is detected by the sensing elements. In the inner sections of the calorimeter, the sensing element is liquid argon. The showers in the argon liberate electrons whose collective signal is recorded. In the outer sections, the sensors are tiles of scintillating plastic which emit light caused by interaction with the showers and which is detected and recorded. The muon spectrometer (blue) identifies and measures muons. It surrounds the calorimeter and measures muon paths from which their momenta can be determined with high precision. Muons are the only detectable particles that are not stopped when they traverse all the calorimeter absorbers. The muon spectrometer consists of thousands of charged particle sensors, similar to the straws described for the inner detector but with larger tube diameters, all within the magnetic field generated by large superconducting toroidal coils. The magnet system (gray) bends charged particles for momentum measurement. They create a strong magnetic field over a huge volume. This field is utilized by the other detector components which measure the curvature of a charged particle's track in the magnetic field.

The interactions in the ATLAS detectors will create an enormous dataflow. The trigger system selects 100 interesting events per second out of 1000 million others. The first-level (LVL1) trigger works on a subset of information from the calorimeter and muon detectors. It requires about 2 micro-seconds to reach its decision during which time all of the information from the detector must be stored in pipeline memories. Information for events selected by the LVL1 trigger is transferred to readout buffers until where the second-level (LVL2) trigger decision is made. The LVL2 trigger refines the selection of candidate objects, using full-granularity information from all detectors, including the inner tracker, which is not used at LVL1.

The data acquisition system 'channels' the data from the detectors to the storage. For LVL2-selected events, event building is performed: since each readout buffer contains fragments of many events for a small part of one subdetector, the event builder must collect all the fragments from one event into a single memory, the memory of an Event Filter processor. The computing system analyzes 1000 million events recorded per year. The ATLAS computing system will be designed to analyze the data produced by the ATLAS detectors[10].

5 Top Physics at LHC

Top quark physics at future hadron colliders have been studied extensively over the past few years, both in workshops and in preparation for experiments at LHC. The production rates will be very large and therefore will lead to a richer physics potential than that of future linear e^+e^- colliders, despite the usual higher background contamination in most channels. Even at low luminosities of $10^{32}cm^{-2}s^{-1}$ approximately 6000 $t\bar{t}$ pairs would be produced per day for $m_t = 170GeV$. This will yield about 100 reconstructed $t\bar{t} \rightarrow (l\nu b)(jjb)$ decays per day and about 10 clean isolated $e\mu$ pairs per day. Both of these final states have been studied in order to evaluate the ultimate accuracy of a measurement of m_t at LHC [11] [12].

The top decays into a bottom quark and a W boson [13]. The b quark's lifetime is about 1.5 picoseconds, much longer than the lifetime of the top and long enough for the b to 'dress' itself with an ordinary quark to form a B meson. This meson travels a short but discernible distance in the detector before decaying. So the B meson can be 'tagged' by a decay vertex displaced from the production point or, in some of its decay modes, by the appearance of a low-momentum electron or muon. The W boson, like the top, lives for only a fraction of a yoctosecond. Two thirds of the time, it decays into a quark-antiquark pair, giving two narrow jets of hadrons. The remaining third of W decays are into lepton pairs giving a conspicuous high-energy charged lepton and an invisible neutrino. The characteristic channels in which top-antitop pair production can be sought are as

End Products	Fraction of Occurrence
$e^+e^-b\bar{b}$	$\frac{1}{81}$
$\mu^+mu^-b\bar{b}$	$\frac{1}{81}$
$\tau^+\tau^-b\bar{b}$	$\frac{1}{81}$
$e^\pm\mu^\mp b\bar{b}$	$\frac{2}{81}$
$e^\pm\tau^\mp b\bar{b}$	$\frac{2}{81}$
$\mu^\pm\tau^\mp b\bar{b}$	$\frac{2}{81}$
$e^\pm$2 jets$b\bar{b}$	$\frac{12}{81}$
$\mu^\pm$2 jets$b\bar{b}$	$\frac{12}{81}$
$\tau^\pm$2 jets$b\bar{b}$	$\frac{12}{81}$
4 jets$b\bar{b}$	$\frac{36}{81}$

Table 2: Each channel is represented by the end products of the top event. They occur different fractions of the time. This is represented above.

shown in Table 2 [18].

The dilepton events are produced when both W bosons decay into a lepton and a neutrino. Events in the lepton plus jets channel occur when one W decays into a lepton and a neutrino and the other decays into a quark-antiquark pair. Although the Tau, the heaviest of the three charged leptons, occurs just as often as the lighter leptons in W decay, its subpicosecond lifetime makes it very difficult to detect [9].

5.1 $t\bar{t} \rightarrow (jjb)(l\nu b)$ decays

In this channel, one W decays into a quark-antiquark pair, while the other decays into a lepton and a neutrino. These $t \rightarrow (e^\pm$ or $\mu^\pm)jjb\bar{b}$ decays provide an abundant event sample, which allows direct reconstruction of m_t, through the invariant mass of the 3-jet system (jjb). The trigger is based on the high-p_T lepton from the other top-quark decay. The main problems to contend with are [12][10][13]:

- background from other processes such as fake electrons(jets which are measured as an electron in the detector) + jets, $b\bar{b} + jets$, and W + jets;
- combinatorial background from the jets in $t\bar{t}$ events;

- systematic uncertainties due to the detector and the theoretical models used.

Although the signal rate is quite large, the background from other processes should not be underestimated:

- For $t\bar{t}$ events triggered on electrons, a potentially large source of background arises from jets faking the electron signature.
- The background from $b\bar{b} + jets$ can be reduced well below the $t\bar{t}$ signal using lepton isolation and missing transverse energy.
- The background from W + jets is the same order of magnitude as the signal. It can be reduced by requiring that at least one reconstructed high-p_T jet be tagged as a b-jet.

The signal-to-background ratio is greater than 1 when requiring that three jets [10][11][12](some kinematic cuts on these jets may apply) be reconstructed in the hemisphere opposite to the trigger lepton, that one of them be tagged as a b-jet and that the remaining pair have an invariant mass consistent with m_W. The reconstruction of the $W \to jj$ decay is important to minimize the detector-dependent systematic uncertainties on the measurement of m_t. A careful study of possible detector systematic uncertainties and also of some of the more theoretical systematic uncertainties has shown that, with this simple selection procedure, one could hope for an accuracy of $\approx \pm 3GeV$ in this channel [10] [11] [12] [18]. This channel is the most straightforward one, since it has the most abundant rate of occurrence and provides a direct reconstruction of the top mass.

5.2 Multilepton events: $t\bar{t} \to (l\nu b)(l\nu\bar{b})$

Multilepton events from top decay can also be used to extract an accurate measurement of m_t. This is done using the two leptons from the two top-quark decay. The background contribution to the signal sample, is minimized by requiring a third high-p_T lepton, which comes from a b-quark decay. This signal sample for the mass measurement thus contains two lepton from W decay and one lepton from b-quark decay. The mass distribution of lepton pairs from the same top decay is much less sensitive to the top p_T-distribution than the mass distribution of lepton pairs from different top-quarks. These multi-lepton channels are characterized by a clean signal but lower statistics.

5.3 Reconstructing the top mass in Monte Carlo studies

In the years before LHC finally starts taking data, Monte Carlo studies are being made to give physicists insight into future analysis. In the case of top mass reconstruction, Monte Carlo generators are very valuable as they provide a way to study characteristics of top-antitop pair production events, which will later be used to isolate these events from other events in the detector. Isolation of these events is important for the study of the top quark and its characteristics but also for the removal of this important source of background in the search for new physics.

5.4 Monte Carlo Studies

In Particle Physics, Monte Carlo is the name given to the simulation (normally by a computer program) of a physics process or of the response of a detector to the passage of some particles (see Appendix D). Monte Carlo simulations are used essentially for two purposes:

- In designing an experiment, the simulation allows detailed studies of all the detector options to help develop an optimum design. Simulations of the generation of the particles produced in some physics process of interest allows the physicists to study the properties of these events to lend insight into their observations.

- In analyzing experimental data, the simulation can be used to compare the actual measurements with those predicted by known or hypothesized processes. Detailed simulation of the response of some elements of the detector can be used to optimize its design, or to reach some conclusion concerning an observed 'signal'.

If the simulation is wrong in some important detail, then the conclusions drawn may be incorrect. Much work goes into the careful comparison of the predicted and actual results to ensure the underlying processes have been correctly implemented. Physicists use Monte Carlo simulations to compare their measurements with some 'control' sample of data to be sure that the experiment is indeed providing accurate results.

In the last 10 years the precision of many simulations has been greatly improved. Full knowledge of physics processes involved in the generation of particles to be measured, and in their measurement is included in the Monte Carlo whenever possible. Some details have to be approximated, to varying degrees, leading to sources of error in the simulation which must be taken into account in the analysis. One example of this is hadronization. The process by which an energetic quark becomes one or more observed hadrons that is not well understood and the simulation of phenomenological models must be made.

5.5 Top-Antitop Pair Production

The LHC collides protons at high energies. Top-antitop pairs are produced from the energy of collisions between the quarks or gluons within the interacting protons. The quarks and gluons from the proton interact to produce a top-antitop pair. There are many different processes that can produce a top-antitop pair when the proton constituents interact. Two of these are illustrated in Figure 2. The quarks or gluons from the protons collide and combine to form a very energetic gluon which decays into a top and an antitop. The yellow lines represent the remaining quarks in the proton which are not involved in the collision. The gluon collision is the dominant process in LHC, accounting for the largest fraction of top-antitop pair production.

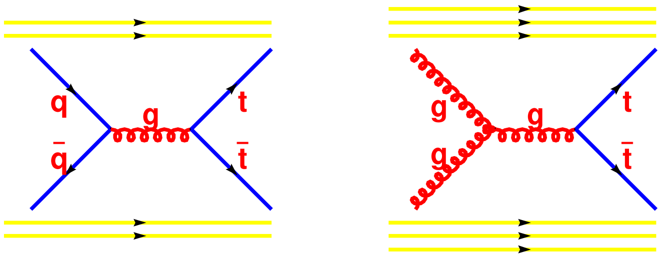


Figure 2: Above are Feynman diagrams of two possible processes by which a top-antitop pair can be produced in LHC.

ATLAS measures the tracks of particles. At the particle level, a typical top-antitop event looks like the event shown in Figure 3. In this figure, photons are drawn with green dashed lines, neutral particles with yellow dashed lines, electrons with red lines, muons in cyan, all other reconstructed particles in yellow, reconstructed electrons with thick red lines, reconstructed muons with thick cyan lines, and reconstructed jets with thick dark blue lines. This is one of thousands of generated events used to study top-antitop pair production event characteristics.

5.6 Top Event Kinematics

Of interest in the case of top-antitop pair production events are the angles between particles and jets in the event. One quantity which gives good insight into the relative positions of these particles is R. This is the distance in η and ϕ between particles (η is related to the polar angle, ϕ is the azimuthal angle, that is, the angle in the plane perpendicular to the beam, see Appendix C). R can be thought of as the separation between particles. It is defined as

$$R^2 \equiv \Delta\eta^2 + \Delta\phi^2. \tag{2}$$

Figure 4 shows the R between various particles in the interaction. For this part of the study, the exact kinematics for each particle in each event are known. This means that the graphs in Figure 4

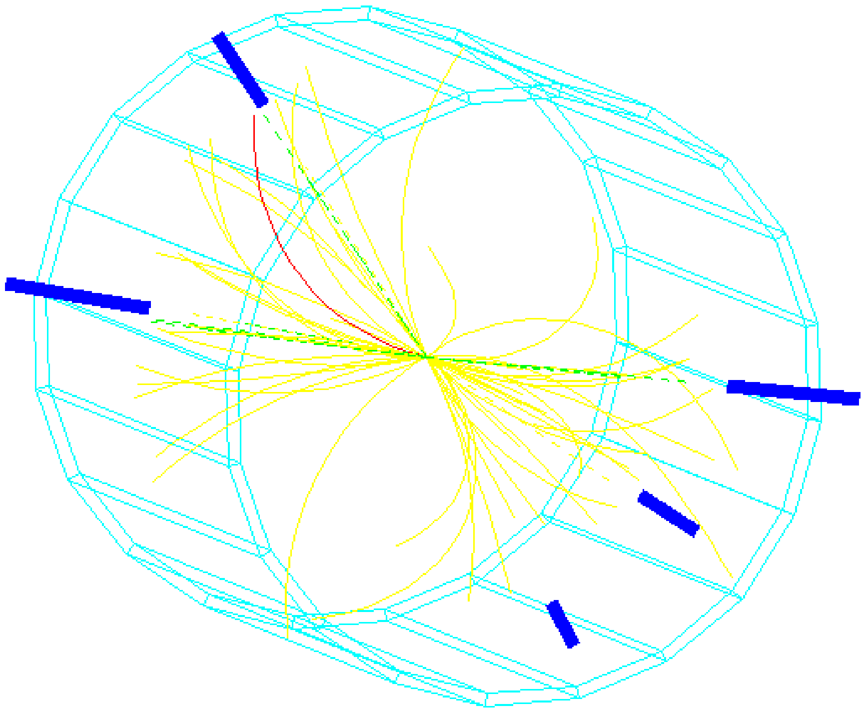


Figure 3: Above is a diagram of the tracks left in the ATLAS detector from a top-antitop pair production event. This diagram was produced in the study presented here, using ROOT.

are ideal cases, where the kinematic variable of each particle in the event are known exactly. In practice this is far from the case: some particles cannot be detected; the detector is not 100% efficient.

The narrow range of R values between the top and the antitop (plot in top left) is a result of the 180 degree difference in direction of the top and the antitop, i.e. they are produced back-to-back. It can thus be seen that ΔR between each W is centered on π as well, with a wider distribution. ΔR between quarks (jets) from the W decay is much smaller. The difficulty with top-antitop events is that the angles between the measurable particles are not clear-cut. This makes it difficult to use these 'characteristics' to select top events without rejecting many good top events.

5.7 Reconstructing the Top Mass

In this Monte Carlo study, some simple criteria are chosen for the isolation of top event. Events were chosen such that one W decays into quarks and the other into leptons. The signature of this event is a high energy isolated lepton track from either an electron or a muon, and at least four jets (two from the W decay and one from each bottom). The first step is to reconstruct the W. This is done by taking all possible combinations of two jets. A more rigorous algorithm will require these two jets to be in the hemisphere opposite to the lepton. The two jets that gives a mass closest to the W mass are accepted as the W jet candidates. To reconstruct the top, these two jets are combined with all remaining jets. Since a top mass cannot be assumed (unlike the W mass which is now well measured), certain combinations cannot be selected based on mass. In the work presented here, no further selection is needed, there is a clear top mass peak. However, in the grueling separation from background events which make the original cut, some further selections may be required. This could include a selection based on the difference in angle between the mean W jet angle (close to W direction) and the other jet used (which should be in the same hemisphere). However some angles

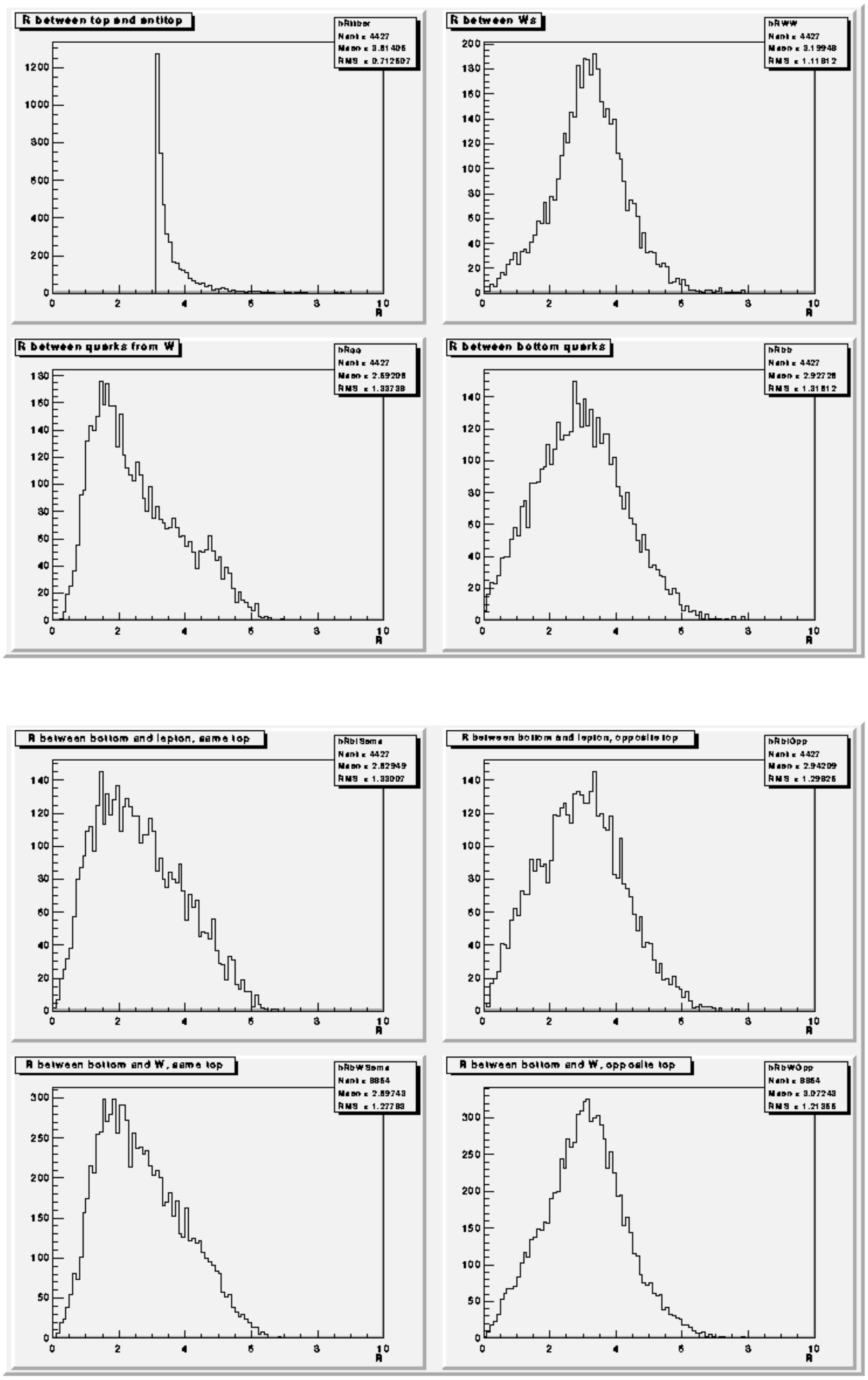


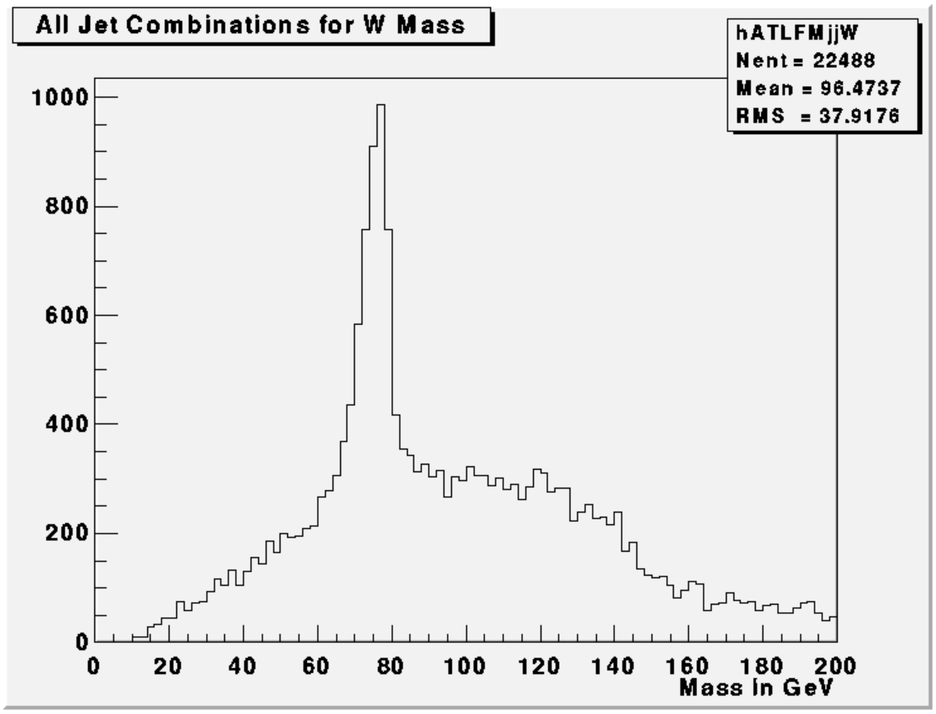
Figure 4: R between various particles in a top-antitop event.

are not clean-cut, which makes them difficult to use as a selection criteria. Also, b-tagging allows physicists to identify jets as being very likely to have originated from a b-quark. Here, b-tagging is not simulated; rather, all jets are treated equally. b-tagging should dramatically reduce the combinatoric background and improve the top mass peak.

Figure 5 shows the reconstruction of the W mass. For this part of the study, the quarks are allowed to hadronize and become jets. Also, quarks and gluons are allowed to radiate, creating extra jets. However, this is not full detector simulation (see Appendix E) because the efficiencies of the detector are not yet simulated. For full simulation the energies of the particles are 'smeared' to simulate an imperfect detector. The background in this plot is due largely to combinatorics. For an event with four jets, there are six possible combinations of two jets. In general, if there are n jets, there are $\frac{n^2-n}{2}$ possible combinations. There can be as many as 12 or more jets in an event. Also, it is possible that if the initial W decay product quark directions are close, two jets can combine into a single jet. In this case, one jet is sufficient for recreating the W, but is still combined with other jets in the event. There is thus a possibility that two jets other than the correct ones are chosen as the W jet candidates.

The two jets used for the W reconstruction are then combined with each remaining jet. Since a top mass cannot be assumed, one relies on a signal to noise ratio greater than 1. Considering only the combinatoric background, there is a clear top signal, shown in Figure 6.

In the detector simulation, the smearing of the particle energies also smears the top mass signal. The result is a broader distribution, much less convincing. It is at this stage that stricter selection criteria must be used. To see how the signal changes with the addition of detector simulation, see Figure 7 and Figure 8.



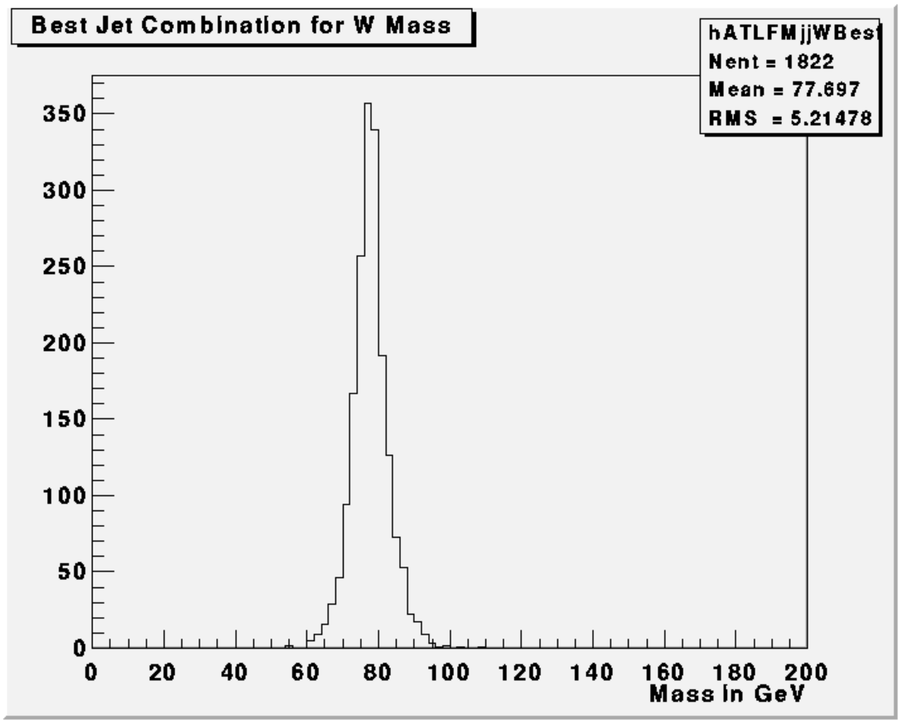


Figure 5: Top is the mass of all combinations of two jets. Bottom is the mass of the combinations closest to the W mass from each event.

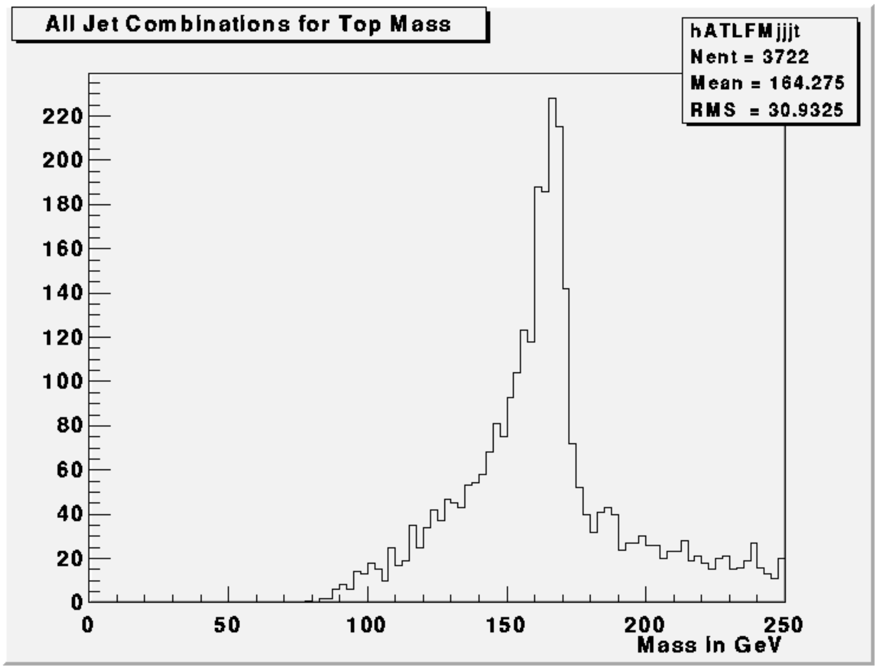


Figure 6: Above is all combinations of the chosen W jets and one other jet in the event. There is a clear peak at the top mass, above combinatoric background.

6 Conclusions

The extremely large mass of the top suggests that it may be fundamentally different from the other quarks. Herein lies the hope that it may lead to new physics. The Standard Model, although successful, leaves many questions unanswered [18][19]. One unsolved mystery is the postulated existence of the Higgs. Many believe understanding the top quarks is necessary to understand the Higgs.

Within the Standard Model and at very high energies the weak interaction and the electromagnetic interaction are unified into a single 'electroweak' interaction. These energies existed in the very early universe. In the low-energy world of present day, the electromagnetic and weak interactions behave very differently. The mechanism behind the 'breaking' of their initial symmetry is not known, but in the simplest model it is caused by a new particle called the Higgs. At high

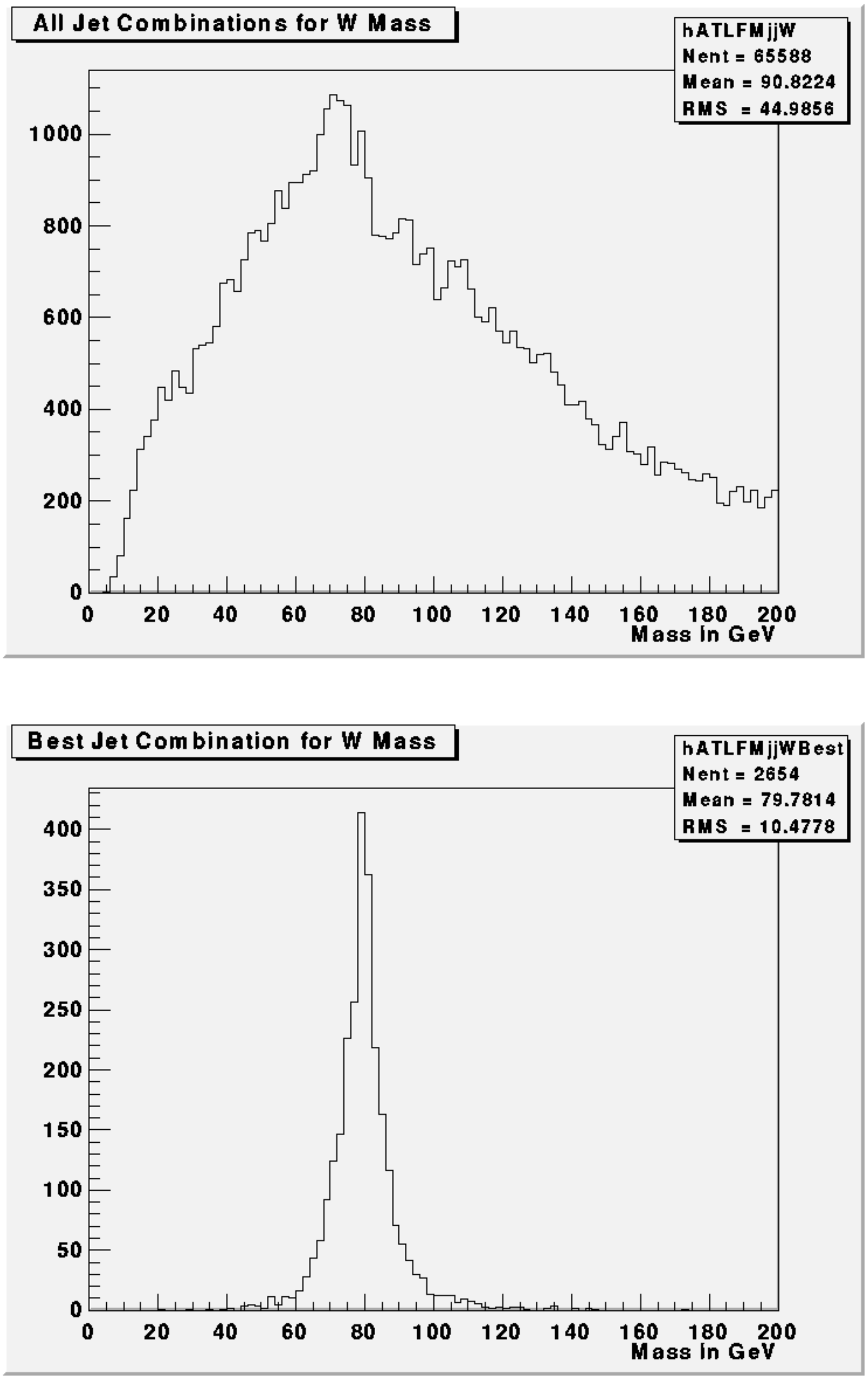


Figure 7: Above is the two jet combinations, and the chosen combination for the W.

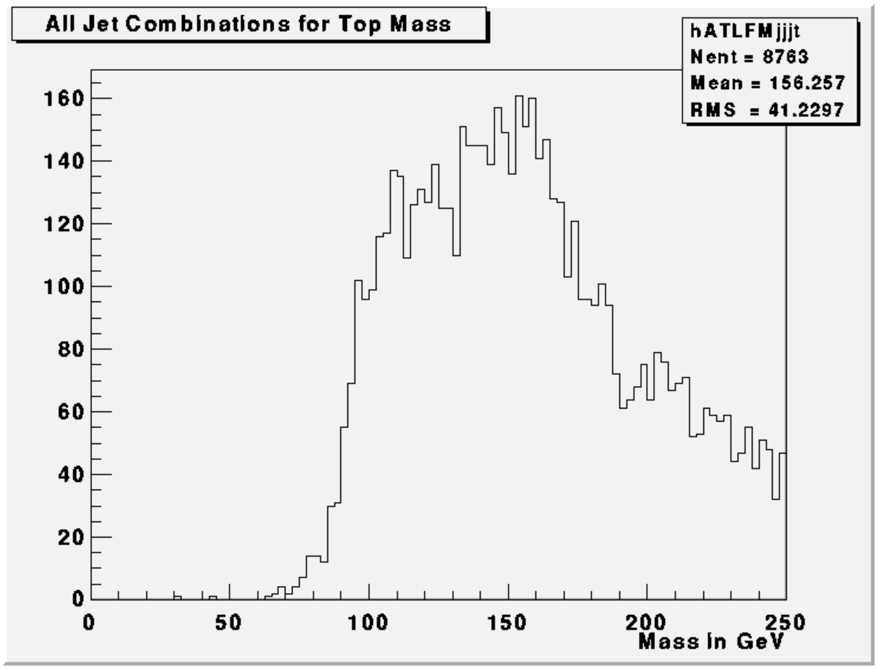


Figure 8: Combinations of the chosen W jets and one other jet in the event are shown. The top mass peak is broad and inconclusive.

energies, when the symmetry exists, the W, Z, photon, leptons and quarks are all massless. At lower energies, when the symmetry breaks, the W and the Z interact with the Higgs and become massive. The quarks and leptons also acquire masses in the process. But whereas the W and Z masses are related in the Standard Model, the quark and lepton masses have to be inserted by means of adjustable parameters that describe how strongly each type of quark or lepton couples with the Higgs [19][6].

For an electron, which is very light, the interaction strength is 3×10^{-6}. For a top quark, it is almost exactly unity. This relatively strong coupling with the Higgs, and to some extent the mystique associated with a value of unity, suggests that the top quark may have a special role. Physicists do not yet know what it is. A very precise measurement of the top's mass, for example, along with that of a W, would lead to a prediction for the Higgs' mass.

There are ways of breaking the symmetry of electroweak theory that do not invoke an elementary Higgs particle. In one candidate theory the Higgs is replaced by a top-antitop pair. This theory predicts the existence of new, heavy particles that decay into top-antitop pairs. Such an effect would enhance the rate of production of top quarks.

Experiments at CERN's Large Hadron Collider may reveal the mechanism of electroweak symmetry breaking and complete the understanding of the gauge boson masses.

The sheer enormousness of the top's mass makes its decays fertile ground for new particle searches. Some theorists have speculated that a few of the events collected by CDF and D0 may contain supersymmetric particles [6]. Supersymmetry is a postulated symmetry that assigns as yet undiscovered partners to every particle in the Standard Model. If such partners exist and are lighter than the top, they might show up in top events. For instance, a top may decay to its own supersymmetric partner,the 'stop'; or supersymmetry could allow a gluino (hypothetical partner to a gluon) to decay into a top-antitop pair. Such effects might even cancel each other out, leading to no net change in the observed production of tops and antitops. Supersymmetry predicts not just one Higgs but a family of five or more. If they exist and are lighter than the top, some of these particles could be found in top decays.

Another critical question is whether quarks, especially the massive top, are really fundamental particles with no substructure. In a few years, physicists will start using the top to try to answer the many questions that still remain about matter and the forces that govern the physical world. What new tenets of physics may arise beyond what is now known is a matter of active speculation that will end only when measurements start to unravel the workings of nature. [18]

7 Acknowledgments

I would like to extend thanks to: Dr. Michel Lefebvre for his guidance, support, and encouragement throughout this project; Dugan O'Neil for his insight, explanations, encouragement and editorial comments; and the University of Victoria Physics Co-op administration for providing me with the opportunity to work with the Particle Physics Group at the University of Victoria.

References

[1] W. Pauli, Phys. Rev., **58** (1940) 716.

[2] B. R. Martin, G. Shaw., *Particle Physics*, John Wiley & Sons Ltd.: Chichester (1992).

[3] D. H. Perkins, *Introduction to high Energy Physics*, Addison-Wesley Publishing Company Inc.: Menlo Park, California (1987).

[4] F. Mandl, G. Shaw, *Quantum Field Theory* John Wiley & Sons Ltd.: Chichester (1984).

[5] P. D. B. Collins, A. D. Martin *Hadron Interactions*, Adam Hilger Ltd.: Bristol, (1984).

[6] H. E. Haber, G. L. Kane *Is Nature Supersymmetric?* Scientific American.: June 1986.

[7] A. Levy, DESY 97-013 (1997).

[8] M. Brkić, DESY F35D-95-10 (1995).

[9] B.C Barish, R. Stroynowski. *The Physics of the τ-Lepton.* Phys. Reports., **157** (1998) 1.

[10] ATLAS Collaboration, *ATLAS, Technical Proposal for a General-Purpose pp Experiment at the Large Hadron Collider at CERN* CERN/LHCC/94-43. LHCC/P2. (1994).

[11] M. Cobol. *Top Physics at LHC* ATLAS Internal Note PHYS-No-93. (1996).

[12] D. Froidevaux. *Top Quark Physics at LHC/SCC* CERN/PPE/93-148. (1993).

[13] Particle Data Group. *Review of Particle Physics.* Eur. Phys. J. **C3** 1-794 (1998).

[14] D. Amidei. R. Brock. *Future ElectroWeak Physics at the FermiLab Tevatron, Report of the tev_2000 Study Group.* FERMILAB-PUB-96/082. (1996)

[15] U. E. Schroder. *Special Relativity.* World Sci. Lecture Notes in Phys. **33** (1990)

[16] D. Griffiths. *Introduction to elementary particles.* John Wiley & Sons Inc.. New York. (1987).

[17] F. Abe et al. *Observation of top quark production in pp collisions with the collider detector at FermiLab.* Phys. Rev. Let. **74** 14 (1995).

[18] C. Quigg. *Top-ology.* Physics Today. **50** 5 (1997).

[19] T. M. Liss, P. L. Tipton. *The Discovery of the Top Quark.* Scientific American. September 1997.

[20] A. P. French. *Special Relativity.* The M.I.T. Introductory Physics Series. W.W. Norton & Company. New York. (1968).

[21] T. Sjostrand. *PYTHIA 5.7 and JETSET 7.4 Physics and Manual.* LU TP 95-20 W5035/W5044 (1995)

[22] R. Brun et al. *ROOT, Overview.* http://root.cern.ch/root/

[23] R. Brun, E. Richter-Was. *Getting started with ATLFast++* http://root.cern.ch/root/html/atlfast/USER_Index.html

A Relativistic Wave Equations

Interaction of particles at a distance is classically described as a potential or field due to one particle acting on another. In quantum theory, however, it is viewed in terms of the exchange of specific quanta, which carry energy and momentum (bosons, for example), associated with the particular type of interaction. The laws of the conservation of energy and momentum can be satisfied only if the process takes place over a time limited by the Uncertainty Principle, that is, $\Delta E \Delta t \geq \hbar$. These exchange quanta are said to be virtual.

In the study of microscopic processes, the particles involved often move with velocities approaching the speed of light. Thus, relativistic effects must be taken into account. A particle with momentum $\mathbf{p}$ in free space can be described by a de Broglie wave function

$$\Psi(\mathbf{r}, t) = N e^{\frac{i(\mathbf{p}\cdot\mathbf{r} - Et)}{\hbar}}, \tag{3}$$

where $\mathbf{r}$ is the position (x, y, z) of the particle, E is the energy of the particle, $\Psi(\mathbf{r}, t)$ is the wave amplitude, and N is a normalization constant. The particle has frequency ν given by

$$\nu = \frac{E}{h} \tag{4}$$

and wavelength λ given by

$$\lambda = \frac{h}{p}, \tag{5}$$

where $p = |\mathbf{p}|$.

The corresponding wave equation describes the propagation of a spinless particle with mass m in free space. It depends on the relation between energy and momentum which is, relativistically, given by

$$E^2 = p^2 c^2 + m^2 c^4. \tag{6}$$

The corresponding wave equation was first proposed by de Broglie in 1924; however it is commonly referred to as the Klein–Gordon equation because of their success in incorporating the electromagnetic interactions into the equation. It is given by [2][3]

$$-\hbar^2 \frac{\partial^2 \Psi(\mathbf{r},t)}{\partial t^2} = -\hbar^2 c^2 \nabla^2 \Psi(\mathbf{r},t) + m^2 c^4 \Psi(\mathbf{r},t). \tag{7}$$

Solving the Klein-Gordon equation shows that for every plane wave solution of the form of Equation 3 corresponding to momentum $\mathbf{p}$ and positive energy

$$E = \sqrt{p^2c^2 + m^2c^4} \geq mc^2, \tag{8}$$

there is also a solution

$$\Psi^*(\mathbf{r},t) = N^* e^{\frac{i(-\mathbf{p}\cdot\mathbf{r}+E_p t)}{\hbar}} \tag{9}$$

corresponding to momentum $-\mathbf{p}$ and negative energy

$$E = -\sqrt{p^2c^2 + m^2c^4} \leq -mc^2. \tag{10}$$

The existence of these negative-energy states is interpreted in terms of an antiparticle, first postulated by Dirac in 1928. For every charged particle there is an associated particle, its antiparticle, with the same mass but opposite quantum numbers (i.e. charge). This is a natural consequence of the existence of the negative energy state solutions of Equation 7, the existence of which otherwise leads to unacceptable consequences. For a detailed account of the Dirac equation see, for example [3] [2]. In the case of neutrinos, the distinction between particle and antiparticle is slightly different. Neutrinos are longitudinally spin-polarized with spin $J_z = -\frac{1}{2}$ ('left-handed'), where z is the direction of the velocity vector, while antineutrinos have $J_z = +\frac{1}{2}$ ('right-handed').

B Particle exchange

The forces of elementary particle physics are associated with the exchange of particles. The electromagnetic interaction is mediated by the exchange of virtual photons. The weak interaction is mediated by the exchange of virtual $W^{\pm}$ or Z^{o} bosons. The strong interaction between quarks is mediated by the exchange of gluons. The strong interaction between hadrons can be described by the exchange of virtual mesons.

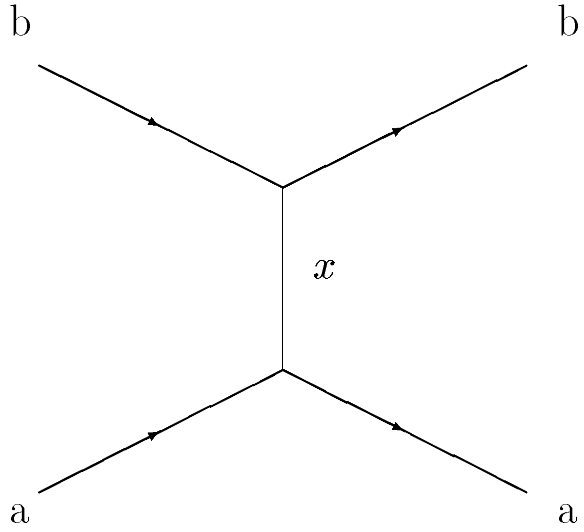


Figure 9: Feynman diagram showing the exchange of particle x between particles a and b.

In the 1940s, Feynman developed a pictorial technique for describing particle interactions [2]. This is an important tool used in the analysis of elementary particle interactions because the mathematics of an interaction can be derived from the Feynman diagram. The Feynman diagram in Figure 9 represents the elastic scattering of two particles a and b, of masses M_a and M_b respectively, via the exchange of a third particle x of mass M_x. In the rest frame of the incident particle a, the lower vertex represents the virtual process

$$a(M_a c^2, \mathbf{0}) \rightarrow a(E_a, \mathbf{p}) + x(E_x, -\mathbf{p}), \tag{11}$$

where $E_x = \sqrt{p^2c^2 + M_x^2c^4}$ and $E_a = \sqrt{p^2c^2 + M_a^2c^4}$. The energy difference is given by

$$\begin{aligned} \Delta E = E_x + E_a - M_a c^2 \quad &\rightarrow \; pc \quad \text{as } p \rightarrow \infty \\ &\rightarrow M_x c^2 \quad \text{as } p \rightarrow 0 \end{aligned} \tag{12}$$

and $\Delta E \geq M_x c^2$ for all p. Since such an energy violation can only persist for a time $\Delta t \approx \frac{\hbar}{\Delta E}$, one can immediately obtain the maximum distance over which x can propagate before being absorbed by particle b, given by

$$r \approx R \equiv \frac{\hbar}{M_x c}. \tag{13}$$

This maximum distance is called the range of the interaction and is inversely proportional to the mass of the exchanged particle. In many applications, this range is very small compared to the de Broglie wavelengths of all the particles involved. For massive exchange particles such as the $W^{\pm}$ and Z^o bosons and mesons, the interaction can then be approximated by a zero range or point interaction, corresponding to the limit $M_x \rightarrow \infty$ as shown in Figure 10.

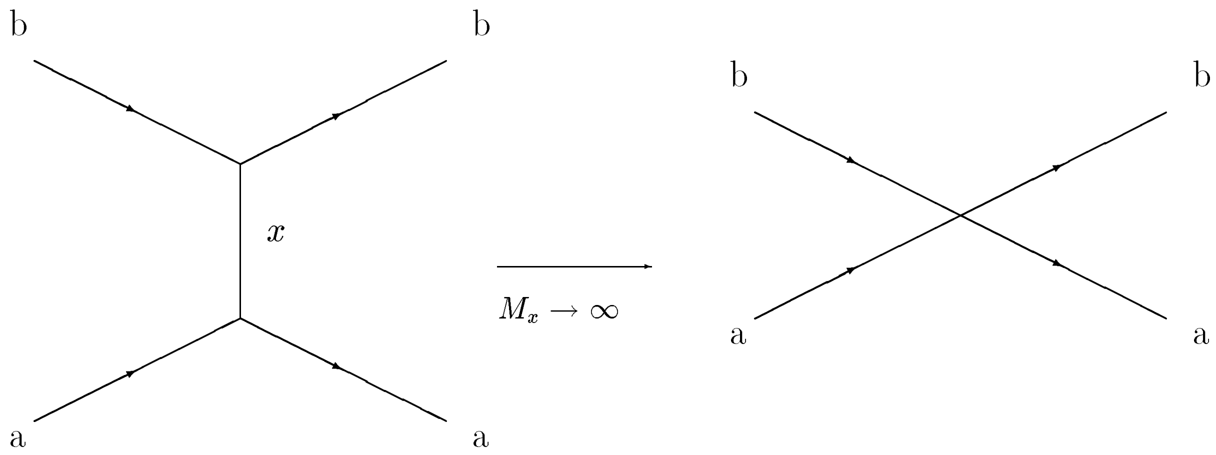


Figure 10: The exchange of particle x as $M_x \rightarrow \infty$ results in a zero range or point interaction.

C Kinematics of Two Body Scattering

Many relations in particle physics are conveniently expressed in terms of the measurable quantities, the kinematic variables. The following is a description of the kinematic variables and relations general to all particle interactions [7] [8]. Figure 9 shows the interaction $ab \to cd$ mediated by the exchange of x. This situation is used in the following definitions.

The energy-momentum four-vector or four-momentum is denoted by the symbol of its corresponding particle, and is given by (for particle a) $a = (E_a, \mathbf{p_a})$ where E is the energy and $\mathbf{p}$ is the momentum three-vector (with x, y, and z components). The transverse momentum p_T of a particle is given by:

$$p_T = \sqrt{p_x^2 + p_y^2}. \tag{14}$$

The rapidity y of a particle is a quantity often used to describe particle production because dy is a Lorentz invariant under longitudinal boosts and so easily translates from the particle's frame of reference to the lab frame. It is related to the longitudinal momentum of the particle and is defined by:

$$y \equiv \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right). \tag{15}$$

The pseudorapidity η is easily measurable because it is dependent only on the scattered polar angle θ of a particle with respect to the z direction (forward direction) and is used to approximate the rapidity. The pseudorapidity is given by:

$$\eta \equiv \tanh^{-1} \cos\theta = \ln \tan \left(\frac{\theta}{2} \right). \tag{16}$$

The pseudorapidity exactly equals the rapidity for a massless particle. Pseudorapidity is a good approximation to rapidity for particles with low masses. The centre of momentum energy squared s is needed to determine the reaction kinematics, and is given by:

$$s = (a+b)^2 = m_a^2 + m_b^2 + 2a \cdot b \approx 4E_aE_b, \tag{17}$$

where m denotes the mass of the subscripted particle. The four-momentum transfer t is given by:

$$t = (a - c)^2 \approx m_a^2 + m_c^2 - E_a E_c \left(\theta^2 + \frac{m_a^2}{E_a} + \frac{m_c^2}{E_c} \right), \tag{18}$$

where θ is the scattered polar angle of c and the two assumptions $m \ll E$ and that θ is small are made. Another frequently used quantity, Q^2, is the square of the magnitude of the four momentum transferred q between the particle b and d, and is given by:

$$Q^2 \equiv -q^2 \equiv -(b - d)^2. \tag{19}$$

D PYTHIA, The Monte Carlo Generator

The study presented in this report used the Monte Carlo Generator PYTHIA. The PYTHIA program is frequently used for event generation in high-energy physics. PYTHIA emphasizes multiparticle production in collisions between elementary particles. This program is intended to generate complete events in as much detail as experimentally observable ones. PYTHIA is limited by the current understanding of the underlying physics of events so many models have been developed and implemented for a number of aspects not covered in the standard model.

D.1 Event Generators

Computers are used to generate events as detailed as could be observed by a perfect detector. This is done by 'factorizing' the full problem into a number of components, each of which can be handled reasonably accurately. The original process is used as input to generate bremsstrahlung corrections, and the result of this exercise is thereafter left to hadronize. In the actual generation procedure, most steps therefore involve the branching of one object into two, or at least into a very small number, each of which are free to branch in turn.

The output of an event generator should be in the form of 'events', with the same average behaviour and the same fluctuations as real data. In the data, fluctuations arise from the quantum mechanics of the underlying theory. In generators, Monte Carlo techniques are used to select all relevant variables according to the desired probability distributions, and thereby ensure randomness in the final events.

An event generator can be used in many different ways. The five main applications are probably the following:

- To give physicists a feeling for the kind of events one may expect/hope to find, and at what rates.

- As a help in the planning of a new detector, so that detector performance is optimized, within other constraints, for the study of interesting physics scenarios.
- As a tool for devising the analysis strategies that should be used on real data, so that signal-to-background conditions are optimized.
- As a method for estimating detector acceptance corrections that have to be applied to raw data, in order to extract the 'true' physics signal.
- As a convenient framework within which to interpret the observed phenomena in terms of a more fundamental underlying theory (usually the Standard Model).

Events are observed by detectors, and the interesting ones are written to disk by the data acquisition system. Afterwards the events may be reconstructed and analyzed. In Monte Carlo studies, the production of events is done by the event generators. The behaviour of the detectors, how particles traverse the detector, spiral in magnetic fields, shower in calorimeters, or sneak out through cracks, is simulated in programs such as ATLFast. This is called detector simulation. Ideally, the output of this simulation has exactly the same format as the real data recorded by the detector, and can therefore be put through the same event reconstruction and physics analysis chain, now providing the 'right answer' according to theory.

There are still many holes in understanding of the full event structure, despite an impressive amount of work and detailed calculations. To put together a generator therefore involves making a choice on what to include, and how to include it. At best, the spread between generators can be used to give some impression of the uncertainties involved. Major approximations are related to the almost complete neglect of the non-'trivial' higher-order effects. It can therefore only be hoped that the 'trivial' higher order parts give the bulk of the experimental behaviour.

E ATLFAST, The Detector Simulator

ATLFast is a fast simulation package for ATLAS. Starting from the list of particle produced in generated events it provides the following output information after simulating the key aspects of the ATLAS detector response in a parameterized way:

- reconstructed jets (labeled as light jets, b-jets, c-jets, τ-jets),
- isolated electron and photons,
- isolated and non-isolated muons,
- missing transverse energy,
- a simplified trigger response,
- charged track parameters (for B-physics studies mostly).

ATLFast is based on the ROOT collection classes, data base and graphics. ATLFast has been designed as a general framework illustrating how to develop independent modules (called Makers), but communicating via a common object-bus and data base.

F ROOT

ROOT is a data analysis and graphics program designed for use in high-energy physics, but adaptable to other fields. The ROOT project started in the context of the NA49 experiment at CERN. NA49 generates approximately 10 Terabytes of raw data per run. This data rate is of the same order of magnitude as the rates expected to be recorded by the LHC experiments. Therefore, NA49 is an ideal environment in which to develop and test new data analysis tools and to study the problems related to the organization and analysis of such large amounts of data. Root is a data analysis tool designed to manage these large amounts of data. With ROOT, a basic framework that offers a common set of features and tools is provided. Currently the emphasis of ROOT is on data analysis.

The ROOT system provides a set of object oriented frameworks with all the functionality needed to handle and analyze large amounts of data in a very efficient way. The data is defined as a set of objects. Specialized storage methods are used to get direct access to the separate attributes of selected objects without having to touch most of the data. Included in ROOT are histograming methods in 1, 2 and 3 dimensions, curve fitting, function evaluation, minimization, graphics and visualization classes. These allow the setup of an analysis system that can query and process data interactively or in batch mode.

The command language, the macro language and the programming language are all C++. The time consuming compile/link cycle is removed to allow fast prototyping of macros. The interpreter allows for fast prototyping of the macros since it removes the time consuming compile/link cycle. If further performance is required the interactively developed macros can be compiled using a C++ compiler.

ROOT is an open system that can be dynamically extended by linking external libraries. This makes ROOT a good platform on which to build data acquisition, simulation and data analysis systems.

The backbone of the ROOT architecture is a layered class hierarchy with, currently, 310 classes grouped in 24 frameworks which are divided in 14 categories. Most of the classes inherit from a common base class, TObject. This organization enables the implementation of some essential infrastructure inherited by all descendants of TObject. However, classes not inheriting from TObject are possible, when appropriate.

G ROOT Details

As ROOT is a new software package, some of the details of the program are included here, for the interested reader. Most of this information can be found from the ROOT Web Site: http://root.cern.ch/root/

G.1 ROOT Classes

The basic ROOT classes contain the low-level building blocks of ROOT. The TObject class implements common behaviour for all ROOT classes. The class TClass and its helper classes provide support for extended runtime type information. The storage manager TStorage handles all memory allocation and de-allocation operations and performs basic error checking. The class TFile provides a hierarchical sequential and direct access store objects for writing and reading. The operating system abstraction layer TSystem and the concrete operating system interfaces TUnixSystem, TWin32System and TMacSystem concentrate all operating system dependent behaviour for the three main platforms supported by ROOT.

The container classes provide general purpose data structure classes such as arrays, lists, sets, B-trees, and maps which are heavily used in the implementation of ROOT itself.

The histogram and minimization classes offer advanced statistical data analysis features, like 1D, 2D and 3D histograming of short, long, float or double values, with fixed or variable bin sizes, as well as profile histograms, data fitting, formula evaluation and minimization.

The tree and ntuple classes contain the tree system. Trees extend the concept of ntuples to all complex objects and data structures found on raw data tapes and DSTs. The idea is that the same data model, same language, same style of queries can be used on all data sets in an experiment. Trees are designed to support not only complex objects, but also a very large number of them in a large number of files. Ntuples are simple trees with only one branch.

The 2D graphics classes contain the low-level graphics primitives such as lines, arrows, rectangles, ellipses, and text. They also contain the higher level constructs like pads and canvases. They also handle basic style and attribute management.

The 3D graphics and detector geometry classes provide basic 3D graphics primitives such as 3D polylines and 3D polymarkers as well as higher level geometrical shapes (boxes, cones, polygons, tubes, etc.) which can be efficiently assembled into very complex detector geometries.

G.2 Basic ROOT commands

ROOT can be acquired from the ROOT web cite: http://root.cern.ch.

To run ROOT, simply type `root` at the prompt. To quit root, type `.q` or `.qqq`.. Once in ROOT, execute a macro or program by typing `.x filename`. The command language in C++ so most commands must end in a semicolon.

To run ROOT in batch mode, type `root -opt filename > output` where opt can be: b, run in batch mode without graphics; q, exit after running all of the macros listed on the command line; l, do not show the logo screen when running ROOT; n, do not run the logon and logoff macros in .rootrc; and ?, list these options.

G.3 How to write objects to a file

Objects derived directly or indirectly from the class TObject can be written to a file. Before writing an object to a file, one must first create a new file or open an existing file in 'UPDATE' or 'RECREATE' mode. For example,

```
TFile f1("file1.root","NEW");
TFile f2("file2.root","RECREATE");
TFile f3("file3.root","UPDATE");
```

Objects, inheriting from TObject, in a file are identified by a key (see class TKey). A key is itself an object with all the necessary information to locate a user object in a file (its name, a title, size, position and a few other parameters). A file has a directory consisting of the list of keys. A key is automatically created by an operation such as `obj->Write("keyname");`. If the file compression attribute is set (TFile::SetCompressionLevel), the buffer holding an object is compressed before being written to the file. Write is a member function of the TObject class (TObject::Write) and performs the following operations:

- It creates a buffer object b (see class TBuffer).
- It fills the buffer by invoking `obj->Streamer(b)`.
- It writes the buffer to the file.
- It adds a new key with the name "keyname" to the list of keys.

The Streamer function is responsible for scanning the object data structure and copy the data to the buffer. This function may be user-written, but is, in general, automatically generated by the ROOT dictionary generator.

G.4 How to Read objects from a file

There are two ways to read an object from a file. An object can be read from a file by using TFile::Get or TDirectory::Get when in a sub-directory. First load the existing file, giving it a variable name f, using the command

```
TFile *f = new TFile("myfile.root");
```

All the objects (keys) in the file can be listed using the command f.ls() or by using the browser which can be executed using the command `TBrowser b;`. To read an object in memory (example a TShape object), use the command:

```
TShape *shape = (TShape*)f.Get(shape_name);
```

Here, f.Get will create an object of class TShape with the following sequences of operations:

- Find the key (eg. shape_name) in the list of keys.
- Create a TBuffer object.
- Read the buffer from the file.
- Create an empty object by calling the default constructor for the referenced class in TKey.
- Call the Streamer function for this new object.

G.5 How to Draw objects

An object is drawn using the `object->Draw(option)` command. This function adds a reference to the object in the list of objects to be drawn in the current pad or canvas. If a canvas does not exist, a default canvas (named c1) is automatically created.

A canvas object has an associated screen window (except when in batch mode) and a list of pads. The canvas itself is a pad. Pads are resizable and movable areas in the canvas. Pads in turn may contain other pads to build a hierarchy. Pads may be automatically generated by TPad::Divide. If a canvas or pad is resized, its contents are automatically redrawn. For example, to draw two histograms h1 and h2 in a canvas c1, the commands:

```
TCanvas *c1 = new TCanvas("c1","Angle in R -1",200,10,600,480);
c1->Divide(2,1);
c1->cd(1);
h1->Draw();
c1->cd(2);
h2->Draw();
```

are used. The TCanvas::Update function is automatically called in a ROOT interactive session when carriage return is typed. When executing a macro, one can force a screen update by calling gPad->Update(), where gPad is a global variable pointing to the current pad. The current pad can be changed with pad->cd() where pad is the pointer to an existing pad.

When an object has been drawn, its drawing option can be redefined by invoking

```
object->SetDrawOption(newoption).
```

A logarithmic scale in x, y and/or z can be set using the functions TPad::SetLogx, TPad::SetLogy, and TPad::SetLogz. Some useful examples are,

```
c1->SetFillColor(10);
c1->GetFrame()->SetFillColor(21);
c1->GetFrame()->SetBorderSize(6);
c1->GetFrame()->SetBorderMode(-1);
```

When a pad is created, the user coordinates are by default set to the range [0,1] for both x and y. The function TPad::Range can be used to specify different values. When drawing high level objects such as histograms or graphs, the pad range is automatically set by the Paint member function of these classes.

G.6 How to Create or Modify a style

All objects that can be drawn in a pad inherit from one or more attribute classes. When the objects are created, their default attributes are taken from the current style. The current style is an object of the class TStyle and can be referenced using the global variable gStyle. ROOT provides two styles called 'Default' and 'Plain'. The 'Default' style is created simply by:

```
TStyle *default = new TStyle("Default","Default Style");
```

The 'Plain' style to get a 'conventional' Postscript output. The instructions in the ROOT constructor to create the 'Plain' style are as follows.

```
TStyle *plain = new TStyle("Plain","Plain Style");
plain->SetCanvasBorderMode(0);
```

```
plain->SetPadBorderMode(0);
plain->SetPadColor(0);
plain->SetCanvasColor(0);
plain->SetTitleColor(0);
plain->SetStatColor(0);
```

The current style can be set using: `gROOT->SetStyle(style_name);`. A pointer to an existing style can be obtained using: `TStyle *style = gROOT->GetStyle(style_name);` Additional styles can be created using:

```
TStyle *st1 = new TStyle("st1");
st1->Set....
st1->cd();
```

This becomes the current style gStyle. Note that when an object is created, its attributes are taken from the current style. For example, a histogram created in a previous session and saved it in a file when redrawn after style changes will be drawn with the original attributes. In order to force the current style attributes to be set when reading an object from a file, the command

```
gROOT->ForceStyle();
```

must be used before reading the objects from the file. Objects can be forced to have the attributes of the current style using:

```
canvas->UseCurrentStyle();
```

G.7 How to Use the histogram classes

The histogram classes are named THdp where d, the dimension, can be 1, 2 or 3. The precision p can be C (for Char), S (for Short), F (for Float) or D (for Double). For all dimensions, both fix and variable bin size options are supported. For example, TH1F is a class for 1-D histograms where a Float is used to increment the sum of weights for each channel. Therefore a histogram with 1, 2, 4 or 8 bytes per channel can be created.

A special class TProfile handles profile histograms. Each histogram class is derived from the base class TH1. The TH1 class derives from TNamed. Histograms are identified by a name and have a title. TH1 also derives from the standard ROOT attribute classes: TattLine, describing the line attributes; TAttFill, describing the fill area attributes; TAttmarker, describing the marker attributes. The TProfile class derives from TH1D. The TH2p classes are derived from the corresponding TH1p classes and also from the auxiliary class TH2. The TH3p classes are derived from the corresponding TH1p classes and also from the auxiliary class TH3.

The TH1C class can be used to create histograms where a maximum sum of weights (or number of entries per bin in case weights are equal to 1) is less than 256. The TH1S class has a similar limit at 65536. The class TH1F uses a 32 bits floating point to store the sum of weights per channel (equivalent to HBOOK1). The TH1D class may be used where the sum of weights requires high precision.

Creating histograms

A histogram is created by invoking one of the histogram class constructors. For example, a 1-D histogram with a fixed number of bins is created with:

```
TH1F *hfix = new TH1F("hfix","hfix title",nbins,xlow,xup);
```

A variable bin size histogram is created with:

```
TH1F *hvar = new TH1F("hvar","hvar title",nbins,xbins)
```

where xbins is an array of (nbins+1) floats, the low-edge of each bin and the upper-edge of the last bin. 2-D histograms are created in the same way (4 different constructors for each precision) with fixed/variable bins along x and/or y.

Filling histograms

Histogram of all types are filled via the `hist->Fill(..)` functions:

- TH1::Fill(x) increment by 1 the bin corresponding to x.
- TH1::Fill(x,w) increment by w the bin corresponding to x.
- TH1::Fill(x,y) increment by 1 the cell corresponding to x and y.
- TH1::Fill(x,y,w) increment by w the cell corresponding to x and y.
- TH1::Fill(x,y,z), etc.. for 3-D histograms.

In all cases, parameters x and y are of type Float_t (32 bits floating point). The parameter w is of type Stat_t (64 bits floating point). If the function TH1::Sumw2 has been called before filling the histogram, the sum of squares of weights is also incremented for each channel or cell.

Drawing histograms

A histogram of any type is drawn by TH1::Draw(option). Many different options are supported. Once a histogram has been drawn, one can modify its attributes or drawing options by using the context menu item DrawPanel.

Writing histograms to a file

Histograms can be written to a file by using `hist->Write()`. To write the full list of histograms stored in memory for a file, the command `file->Write()` where file is the pointer to the file object, is used.

Reading histograms from a file

In an interactive session, a histogram can be read directly into memory using `TH1F *hname = (TH1F*)f.Get(hname` where f is a pointer to an open file containing the object hname. The browser can also be used to read an object. Clicking on a histogram loads the histogram in memory and draws it.

Fitting histograms

The function TH1::Fit is used to fit histograms. By default, when invoking TH1::Fit, the fitted function object replaces the previously fitted function in the list of fitted objects for this histogram. Several associated fitted functions can be used simultaneously if the fit option "+" is specified. By default, the specified function is fitted in the full histogram range. If a range for the fitted function has been set, it can be accounted for by specifying the option "R". The histogram context menu item FitPanel can be selected for a visual inspection of the fit options and to set the fit range using a slider.

G.8 How to Fit histograms or data points

Fitting in ROOT is based on the TMinuit class. The user has the choice of using the TMinuit class directly or using specialized functions provided in the histogram class TH1 or the graph classes TGraph and TGraphErrors.

Fitting 1-Dim histograms

Fitting histograms is done using the function TH1::Fit. The name of the fitted function (the model) which may be one of the ROOT pre-defined function names or a user-defined function, is passed as the first parameter.

- Using pre-defined functions

The following functions are automatically created by ROOT:

- gaus. A Gaussian with 3 parameters: $f(x) = p_0 * e^{\frac{-1}{2}(\frac{x-p_1}{p_2})^2}$.
- expo. An exponential with 2 parameters: $f(x) = e^{p_0+p_1 x}$.
- polN. a polynomial of degree N: $f(x) = p_0 + p_1 x + p_2 x^2 + \ldots$

The command `hist.Fit("gaus");` is an example of how to fit histogram object hist with a Gaussian. By default, the fitting function object will be added to the histogram structure and drawn in the current pad. There is no need to set initial values for the parameters.

- Using user-defined functions

  A TF1 object must be created. For example, to create a function called 'myfit' with 3 parameters in the range between 0 and 2, use the commands

```
TF1 *myfit = new TF1("myfit","[0]*sin(x) + [1]*exp(-[2]*x)", 0,2);
myfit->SetParName(0,"c0");
myfit->SetParName(1,"c1");
myfit->SetParName(2,"slope");
myfit->SetParameter(0, 1);
myfit->SetParameter(1, 0.05);
myfit->SetParameter(2, 0.2);
```

  Parameter names may be set (as above). Parameter start values must be set. To fit this function, use the command `hist->Fit("myfit");`.

Fitting a sub-range of the histogram bins

By default,TH1::Fit will fit the function on the defined histogram range. By specifying the option "r" in the second parameter, the fit can be restricted to the range specified in the TF1 object.

Fitting several functions on the same histogram

By default, a Fit operation deletes the previously fitted function in the histogram data structure. The option "+" can be specified in the second parameter of TH1::Fit to add the newly fitted function to the existing list of fitted functions for this histogram. Note that the fitted function(s) are saved with the histogram when the histogram is written to a ROOT file.

Fitting non-equidistant data points

The classes TGraph or TGraphErrors are used to fit non-equidistant data points, using the function TGraph::Fit. For example, to fit the graph called "gr" in the above examples, with a polynomial of degree 4, the command:

```
gr->Fit("pol4");
```

is used.

Access to the Fit parameters and results

Once a histogram has been fitted, the fitted function parameters can be accessed using the 'Get' functions of the TF1 and TFormula classes. The display of the fit parameters can be enabled in the current canvas by selecting the corresponding item in the Options canvas tool bar menu.

Fitting with one or more parameters between some bounds

To set bounds for one parameter, use TF1::SetParLimits:

```
func->SetParLimits(0, -1, 1);
```

where func is the pointer to the function to be fitted.

G.9 The concept of Trees

ROOT introduces a new concept called Trees. Trees provide the functionality of Ntuples and much more. The Tree architecture extends the concept of the Ntuple to all complex objects or data structures found in Raw Data tapes and Data Summary Tapes. The idea is that the same data model, same language, and same style of queries can be used to all data sets in one experiment. Trees are designed to support not only complex objects, but also a very large number of them in a large number of files.

A Tree (class TTree) is made of branches. Each branch (class TBranch) is described by its leaves (class TLeaf). The leaves can be simple variables, structures, arrays or objects. An array may be of variable length, the length itself being a variable in the same branch or another branch. When the fruits of one branch (detector data) are ready to be picked, they are collected into baskets (class TBasket). When the Baskets become full, they go to the store, the file.

Each Branch goes to a different buffer (Basket). Different buffers can be organized to be written to the same file or to different files.

An interactive analysis program can make selections on the event flag and then only read buffers of the selected events. This scheme allows insertion of a new Branch in an existing file or set of files, at any time.

Only the branch buffers containing these variables are read into memory, when queries are executed on one or more variables or objects.

The Tree data structure allows direct access to any event, direct access to any branch or any leaf, even in the case of variable length structures.

G.10 How to Create and fill a tree

There are two methods to create and fill a tree: the serialization method and the branch method. Alternating between these methods implies no changes in the object model.

Using the serialization method and creating a key

The serialization method is perfectly appropriate when it does not make sense to access sub-object components.

This case does not use a Tree at all. Once the event is filled, it is serialized using `event->Write` in one buffer and written to the file using a key (in this case the string 'event' followed by the event number). This technique can be used in the context of a data acquisition system where the speed of producing the output file is important.

Making a Tree with one single branch

To create a tree with a single branch, use the commands

```
TTree *tree = new TTree("T","An example of a ROOT tree");
Int_t split = 0;
Int_t bsize = 64000;
Event  *event = 0;
tree->Branch("event", "Event", &event, bsize, split);
```

The variable bsize specifies the initial basket size. If this size is not enough to accommodate at least one event, it will be resized accordingly. When the variable split is set to zero, the command creates one branch only. The branch is of type Event. When the structure event has been filled with the appropriate values, the branch can be saved using the command `tree->Fill():`.

This method offers advantages compared to the serialization method.

- It generates files slightly smaller than the first case.
- It minimizes the number of keys.

- Looping on the data set is a bit simpler
- One can use the TTree::Draw command to histogram a member function of the Event class.

Making a Tree with many branches

Setting split equal to one will generate several branches, one for each data member of the Event class (see above example). The following rules are applied:

- If a data member is a basic type, it becomes one basic branch of class TBranch.
- A data member can be an array of basic types. In this case, one single branch is created for the array. If a data member is a basic type, it becomes one basic branch of class TBranch.
- If a data member is an object of a class derived from TObject, the data members of this object are also split into branches (one level only).
- If a data member is pointer to an object, a special branch of type TBranchObject is created.
- In split mode, a data member cannot be a pointer to an array of basic types. A variable size array must be encapsulated inside another object derived from TObject.
- In split mode, a data member cannot be a C structure.
- In split mode, a data member cannot be an array of objects.
- If a data member is a pointer to a TClonesArray object, one super branch of class TBranch-Clones is created. This super branch will have a buffer where to store the number of objects in the array. This super branch has the name of the data member. ROOT will also automatically generate additional branches, one branch for each data member of the class referenced by the array. Note that the data members of this class must be basic types only or arrays of basic types. Data members of this class cannot be pointers.

- In split mode, a data member cannot be a TClonesArray. Only pointers to TClonesArray are accepted.

- Note that the split method can quickly generate many branches. Each branch has its own buffer in memory. In case of many branches, the buffer size should be adapted accordingly. A recommended buffer size is: 32000 bytes for less than 50 branches; 16000 bytes for less than 100 branches; and 4000 bytes for more than 500 branches.

This method provides the highest possible level of granularity, with direct access to any single event. A subset of all the branches can be read selectively This method requires more memory, because one buffer must be created for each branch.

G.11 How to Read a tree

There are 2 ways of reading a Tree and making selections.

The simple way

The function TTree::Draw is used to quickly histogram a few variables with selection criteria. The loop on the Tree is done automatically by ROOT. The command

```
void Draw(Text_t *varexp,Text_t *select,Option_t *opt,Int_t nevnt,Int_t firstevnt);
```

is used, where the parameters have the following meaning:

- varexp is an expression with one of these forms e1, e1:e2 or e1:e2:e3. These generate 1-D, 2-D or 3-D distributions, respectively. Each of e1,e2,e3 are expressions with combinations of the original Tree variables.

- select is an expression with a combination of the Tree variables in which a selection all the C++ arithmetic and mathematical operators are valid.

- opt is the drawing option.
- nevnt is the number of events to process (default is all).
- firstevnt is the first event to process (default is 0).

By default the temporary histogram created is called htemp. If varexp contains `>>hnew`, the new histogram created is called hnew and it is kept in the current directory. To append data to an existing histogram, use "+" in front of the histogram name.

The general way

In general, the user must loop over all events of a tree. In a loop, one may want to generate many histograms, or correlate variables from one Tree with variables of another Tree. The following example is an exert from Appendix H and illustrates:

- How to loop on a tree.
- How to read a sub-branch only (the branch containing the number of tracks).
- How to histogram the result of a member function of the event object.

```
{
  gROOT->Reset();
  gSystem->Load("libUser.so");
  TFile *file = new TFile("htree.root");

  TH1F *h1 = new TH1F("h1","Child Particle Code",39,-19.5,19.5);
  User *myuser = new User;
  TBranch *branch2  = Tree.GetBranch("Wplus_Branch");
  branch2->SetAddress(&myuser);
  Int_t nevent = Tree.GetEntries();

// Start main loop on all events

  for (Int_t i=0;i<nevent;i++)
  {
    Tree.GetEvent(i); // Read complete accepted event in memory
```

```
    h1->Fill(myuser->CurrentFirstChild()->GetKF()); //Fill Histogram
  }
}
```

This method requires access to the class implementation when one of the branches is an object. Note that in the above example, the object of class User must be created using the new operator. It cannot be a static object because a new object will be recreated for each event.

ROOT includes a function TTree::MakeCode to automatically generate the code for a skeleton analysis function.

G.12 How to Compile and link with the ROOT libraries

The ROOT distribution kit includes a directory root/test with a suite of test programs to be compiled and linked with the standard ROOT libraries. This directory includes also a Makefile specific to each platform to compile and link a given test program. For example, to generate an executable module of the Event test program, execute the shell script command:

```
make Event
```

This command will:

- Compile Event.C if it has been modified since the previous make.
- Invoke the rootcint utility in case the Event.h file has been modified and will compile it.
- Generate the shared library libEvent.so.
- Generate the executable module Event.

In an interactive ROOT session, dynamically linking to an existing shared library is done by executing the command:

```
gSystem->Load("libEvent.so");
```

The user then has interactive access to the member functions of the classes in that library.

G.13 How to Convert HBOOK/PAW files to ROOT files

The ROOT distribution kit includes a program called h2root that converts HBOOK/PAW histogram or ntuple files into ROOT files. To use this program, type the shell script command:

```
h2root  hbookfile  rootfile
```

If the second parameter is not specified, a file name is automatically generated. If the hbookfile name is of the form file.hbook, then the ROOT file name will be called file.root.

H A ROOT Example

This section contains an example of ROOT. It incorporates the ATLFast classes without detector simulation. The files included here are as follows:

- LinkDef.h – Tells the compiler how to link the following two files and other libraries it may need.
- User.h – User class definition. There are data members for the found W and for two decay products of the W.
- User.cxx – Implementation of the User class. This is fairly general. For example, one can change to W production by changing the run.C and User_main_TTbar.C routines.

Any changes to the above two files requires compilation before running ROOT.

- run.C – A file where all the Pythia cards are set. This is where to change what type of events one wants, as well as what sort of Atlfast smearing one wants.
- User_main_TTbar.C – This is the main program from which the tree is defined and the user routines are called. The number of events, and the particle one is looking for in the event are set here.
- Plot_TTbar.C – This is a ROOT macro which plots the child KF code, Pt of all children, Pt of muon children only, and the KF code of the found particle.

Any changes to the above three files do not require recompilation before running ROOT.

The program User_main_TTbar generates two files:

- htree.root – This is the output file where the tree is stored. It is written over every time ROOT is run.

- atlfast.root – The lines to generate this file are commented out in User_main_TTbar.C. This file would contain the atlfast tree.

To compile, use the following Makefile:

```
# Makefile for Atlfast for HP-UX 9.x and 10.x with CC
CXX           = CC
CXXFLAGS      = -w +a1 +Z -I$(ROOTSYS)/include -I$(ATLFAST) -I$(ATLFASTB)
OPT           = -O
NOOPT         =
LD            = CC
LDFLAGS       = -g +a1 -z
SOFLAGS       = -b
SOGFLAGS      = -b
ROOTLIBS      = $(ROOTSYS)/lib/*.sl
LIBS          = $(ROOTLIBS) -lm
GLIBS         = $(ROOTLIBS) $(ROOTGLIBS) -L/usr/lib/Motif1.2
-L/usr/lib/X11R5 \
                -lXm -lXt -lX11 -lm -lPW -ldld
PYTHIALIB     = -L$(PYTHIA) -lPythia
ATLFASTLIB    = $(ATLFAST)/libAtlfast.so
ATLFASTBLIB   = $(ATLFASTB)/libAtlfastB.so

all:libUser.so

UserDICT.cxx: User.h LinkDef.h
        @echo "Generating dictionary ..."
        @$(ROOTSYS)/bin/rootcint -f UserDICT.cxx -c User.h LinkDef.h

UserDICT.o: UserDICT.cxx
        $(CXX) $(CXXFLAGS) -c UserDICT.cxx

libUser.so: UserDICT.o User.o
        $(LD) $(SOFLAGS) -o libUser.so User.o UserDICT.o

User.o: User.cxx
        $(CXX) $(CXXFLAGS) -c User.cxx

clean:
        rm -rf *.o *.so *~  *.eps core
```

Run ROOT by typing root at the prompt. For more information on how to set up ROOT, see:

http://wwwhep.phys.uvic.ca/ uvatlas/atlas/software/

To run User_main_TTbar.C, type the following at the command line:

```
root [0]   .x User_main_TTbar.C
```

H.1 User.h

```
#include <TMCParticle.h>
#ifndef User_H
#define User_H

//* This is the User class.  These functions are all called from
//* User_main.C

class User {

  protected:

//*  These are the data members.  particle stores the information of the
//* desired particle in the event.  child1 and child2 store the
//* information of the first two decay particles of particle.

   TMCParticle *particle;
   TMCParticle *child1;
   TMCParticle *child2;

  public:
                User();  //This is a dummy constructor.
   void         Init();  //Tree and Branches are created here.
   void         Filter(Int_t&); //Dummy routing for filtering data.
   void         SetRunParameters();  //Set Pythia Cards, Seed,
                                    // Hadronization Mode and ClusterMaker
   void         SetMakerParameters(); //Set Muon Smearing Option
   void         GetParticle(Int_t, Int_t, Int_t&);//Finds a Particle
   void         GetChildren(Int_t, Int_t&, Int_t&);//Finds children of
                                                  // a particle
   TMCParticle* CurrentParticle(); //returns particle
   TMCParticle* CurrentFirstChild(); //returns child1
   TMCParticle* CurrentSecondChild(); //returns child2

ClassDef(User,1)
};
#endif //User
```

H.2 User.cxx

```
//=========================================================================
//  An example of a User class is given below:                          //
//=========================================================================
#include <iostream.h>
#include <stdio.h>
#include <TFile.h>
#include <TCanvas.h>
#include <TMCParticle.h>
#include <TMath.h>
#include <TPythia.h>
#include <TSystem.h>
#include <TROOT.h>
#include <ATLFast.h>
#include <ATLFMCMaker.h>
#include <ATLFMiscMaker.h>
#include <ATLFClusterMaker.h>
#include <TRint.h>
#include <TH1.h>
#include <stdlib.h>
#include "User.h"
//////////////////////////////////////////////////////////////////////////
//                                                                      //
// User                                                                 //
//                                                                      //
// Example of a user class to perform user specific tasks when running  //
// the ATLfast program.                                                 //
//                                                                      //
// This class illustrates some simple uses of the TTree class.  Here,   //
// the TTree tree has two branches: a branch for each of the W+ and     //
// W- which contain the kinematics of the W originating from production;//
// and the kinematics of the first two children of the W.               //
//                                                                      //
// The variables stored in this tree can be plotted using Plot_TTbar.C  //
// which currently plots the child particle codes, the Pt of all        //
// children, the Pt of muon children, and the particle code of the      //
// found W.                                                             //
//                                                                      //
//////////////////////////////////////////////////////////////////////////

//* Implementation of the User Class
ClassImp(User)

//* These are variables used to fill the W Branches and the
//* Child TMCParticle members of the of the TTree tree, respectively.
```

```
TMCParticle *particle;
TMCParticle *child1;
TMCParticle *child2;

//___________________________________________________________________________
//* This constructor does not set any initial values or call any
//* functions.

User::User()
{
particle=NULL;
child1=NULL;
child2=NULL;
}

//___________________________________________________________________________
//* This function initializes some parameters thru function calls to
//* SetRunParameters and SetMakerParameters.  This function then creates a
//* TTree object with two branches of type TMCParticle (class type for
//* storing basic information of a particle, such as four-momentum,
//* particle code, parent, etc).

void User::Init()
{
//* set parameters for a job

//* If the file named "run.C" exists - execute it as macro - otherwise use
//* predefined settings from SetRunParameters and SetMakerParameters
//* which choses top-antitop pair production with no smearing.

  if (gSystem->AccessPathName("run.C",kReadPermission) == 0 )
    gROOT->Macro("run.C");
  else {
   fprintf(stderr,"WARNING in TopMassGen::Init -> File `run.C` with ");
   fprintf(stderr,"settings does not exist.\n Using default settings.\n");
    SetRunParameters();    //change default parameters
    SetMakerParameters();  //change default parameters
    }
}

//___________________________________________________________________________
//* This function loops through each particle of the current event.
//* If the particle is the desired particle (KF code for desired particle
//* is passed into KFcode) originates from the desired parent (KF code
//* for desred parent particle is passed into ParentKF), its
//* information is saved in the datamember particle of the current
//* instance of the user class. The line number of the particle (C++
```

```
//* version, ie starting at 0) is passed back to calling funcion
//* through i.

void User::GetParticle(Int_t KFcode, Int_t ParentKF, Int_t &i)
{
//* Print out the passed in KFcode and ParentKF.  For debugging
//* purposes.

//   cout << "LOOKING FOR:  "<<KFcode<< "  FROM:  "<<ParentKF<<endl;

//* Dummy integer for indexing particle.

   Int_t j=0;

//* This TMCParticle instance is for the parent particle.

   TMCParticle *particle2;
   particle2 = NULL;

//* Saves the event information (fruits) from the current event to a
//* TClonesArray(class for an array of identical objects, actually a
//* sophisticated linked list) of TMCParticle's.

   ATLFMCMaker *mcmaker = gATLFast->MCMaker();
   TClonesArray *particles = mcmaker->Fruits();


//* Create an integer which sores the number of particles in the event.

   Int_t nparticles = particles->GetEntriesFast();

//* This forloop loops over every particle in a single event,
//* looks for particles of type KFcode with parent of type ParentKF
//* and saves the particle in particle.

   for (i=0; i<nparticles; i++)
   {
//* Put particle at line number i (first particle is line number 0)
//* into particle.

     particle=(TMCParticle*)particles->At(i);

//* Makes j the line number of the parent particle.  Since this
//* function simply returns the LUJETS value of K(3) where the
//* first particle starts at line number 1, 1 must be subtracted
//* from j to make it the right ROOT line number.  This is a bug
//* in the ATLFast interface to Pythia.  This is used to insure
```

```
//* the parent particle is of type ParentKF.

     if (particle->GetParent()!=0)
        j=(particle->GetParent()) - 1;
     else
//* If the Parent is 0, then don't subtract 1, or else it will have
//* an illegal array index
        j=(particle->GetParent());

//* Put particle at corrected line number j (Parent Particle) into
//* particle2.

     particle2=(TMCParticle*)particles->At(j);

//* Since j is 0 if the parent is the beam particle or if there is
//* no parent, here is an if statement for the special case of
//* looking for a particle with no parent.  If there is a match
//* break out of the loop so the current particle is the one that
//* will be saved.

     if ((ParentKF==0) && (particle->GetKF()==KFcode) &&
        (particle->GetParent()==0))
        {
        cout << nparticles << ", " << i << ", " <<
             particle->GetKF() << " FOUND IN GETW" << endl;

          break;
        }
//* If the current particle satisfies KFcode and ParentKF conditions,
//* break out of the loop so the current particle is the one that
//* will be saved.

     if ((particle->GetKF()==KFcode)&&(abs(particle2->GetKF())==ParentKF))
        {
        cout << nparticles << ", " << i << ", " <<
             particle->GetKF() << " FOUND IN GETW" << endl;

          break;
        }
   } //end of for loop

//* If every particle in the event was looped over check to see if the
//* last particle was the desired particle.  If it wasn't, don't save
//* it.  i=-99 is a flag value for the line number of the current
//* particle.
   if ((i==(nparticles))&&(particle->GetKF()!=KFcode))
     {
```

```
      i=-99;
      particle = NULL;
      }
}

//____________________________________________________________________________
//* This function loops through each particle of the current event.
//* If the particle is the desired particle originates from the desired
//* parent (the line number for desred parent particle is passed into i),
//* its information is saved in the datamember child1 or child2 of the
//* current instance of the user class, depending on whether or not it is
//* the first or second child found. The line number of each particle (C++
//* version, ie starting at 0) is passed back to calling funcion
//* through line1 and line2.

void User::GetChildren(Int_t i, Int_t &line1, Int_t &line2)
{
//* Temporaty TMCParticle object for looping through the particles.

   TMCParticle *temp;

//* Nubmer of particles found to date is saved in ChildNum

   Int_t ChildNum=0;

//* Saves the event information (fruits) from the current event to a
//* TClonesArray(class for an array of identical objects, actually a
//* sophisticated linked list) of TMCParticle's.

   ATLFMCMaker *mcmaker = gATLFast->MCMaker();
   TClonesArray *particles = mcmaker->Fruits();

//* Create an integer which sores the number of particles in the event.

   Int_t nparticles = particles->GetEntriesFast();

//* Dummy variable for looping through particles.

   Int_t j=0;

//* Check to make sure the parent line number is valid.
   if (i<0)
     {
//* Flag value of line1 and line2.
      line1=-99;
      line2=-99;
      child1 = NULL;
```

```
     child2 = NULL;
     cout <<"INVALID PARENT-- NO CHILDREN SAVED"
          << endl;
     return;
     }

//* This for loop loops over every particle in a single event,
//* looks for particles with parent at line number i
//* and saves the first two children into child1 and child2.

   for (j=0; j<nparticles; j++)
   {
//* Put particle at line number j (first particle is line number 0)
//* into temp.

     temp=(TMCParticle*)particles->At(j);

//* If the line number of the parent particle is equal to i, than
//* save this particle.  If it is the first child found save it
//* in child1.  If it is the second child found than save it in
//* child2 and break out of the loop. Since this function simply
//* returns the LUJETS value of K(3) where the first particle starts at
//* line number 1, 1 must be added to the desired parent number to make it
//* the right LUJETS line number.  This is a bug in the ATLFast interface
//* to Pythia.

     if (temp->GetParent()==(i+1)) ChildNum++;
     if (ChildNum==1) {
        child1=temp;
        line1=j;
        cout << nparticles << ", " << j << ", " <<
           particle->GetKF() << ", " <<
           child1->GetParent() <<  endl;

     }
     if (ChildNum==2) {
        child2=temp;
        line2=j;
        cout << nparticles << ", " << j << ", " <<
           particle->GetKF() << ", " <<
           child2->GetParent() << endl;
        break;
     }
   }

//* If less than two children were found, don't save any children.
   if ((j==(nparticles-1))&&(abs(child2->GetParent())!=i+1))
```

```
     {
     j=-99;
     child1 = NULL;
     child2 = NULL;
     cout <<"DID NOT DECAY INTO 2+ PARTICLES-- NO CHILDREN SAVED"
          << endl;
     }
}

//___________________________________________________________________________
//* This function is used for filtering events in the ATLFast Tree.
//* I accept all events.  I don't think this affects the user tree
//* at all, but I could be wrong.

void  User::Filter(Int_t &kFilter)
{
   // Accept all events

    kFilter = 1;

}

//___________________________________________________________________________
void User::SetRunParameters()
//* This is where the Phythia cards are set.  Also, the random number
//* seed, the Hadronization Mode and ClusterMaker parameters are set.
{
//* change ATLfast default parameters

//* This sets the mode to "WH, H-->bb, W-->lnu"
//* See:
//* http://root.cern.ch/root/html/atlfast/src/ATLFMCMaker.cxx.html
//* for other options.

//   gATLFast->MCMaker()->SetProductionMode(7);

//* Turn off default event types.  I think this is done so that
//* the exact Pythia cards can be set, but I cannot find any
//* documentation on this, or event source code for this option.

   gATLFast->MCMaker()->SetProductionMode(0);

//* The following code chooses the event type; use PYTHIA options
//* to choose ttbar events.

//* Create a TPythia object to run Pythia and set Pythia Cards.
```

```
   TPythia *pythia = new TPythia();

//* See the Pythia Manual for more information on the Pythia cards.
//* Equivalent Pythia Commands are provided here.

   pythia->SetMSEL(0);                   //MSEL=0
   pythia->SetMSUB(81,1);                //MSUB(81)=1
   pythia->SetMSUB(82,1);                //MSUB(82)=1

   pythia->SetMSTP(48,2);
   pythia->SetMSTP(51,5);
   pythia->SetMSTP(61,0);

   pythia->SetMSTP(71,0);
   pythia->SetMSTP(81,0);

   for(Int_t i=172; i<192 ;i++)          //Example:
     pythia->SetMDME(i,1,1);             //MDME(172,1)=1

   for(i=41; i<48 ;i++)                  //Example:
     pythia->SetMDME(i,1,0);             //MDME(41,1)=0

   pythia->SetMDME(46,1,1);              //MDME(46,1)=1

   pythia->SetMDME(188,1,1);             //MDME(188,1)=1
   pythia->SetMDME(189,1,1);             //MDME(189,1)=1

//* set hadronisation type, random number seed

//* This is the default Hadronisation Mode.  It set the mode to
//* "ISR+FSR+HP+HADRONIZE".
//* See:
//* http://root.cern.ch/root/html/atlfast/src/ATLFMCMaker.cxx.html
//* for other options.

   gATLFast->MCMaker()->SetHadronisationMode(3);

//* This is the default random number seed (51111).  I do not know what
//* the valid range for random numbers is, but I think it's the same as
//* the Pythia range (between 0 and 900000000, I think).

   gATLFast->MCMaker()->SetRndNumber(51111);

//* I can't find any documentation on the Save member function, so
//* I don't know the difference between the following two lines of
//* code.  However, the ClusterMaker deposites the energy of
//* generated particles into the calorimeter and reconstructs clusters.  I
```

```
//* assume that it is not being used here.

   gATLFast->ClusterMaker()->Save(0);

//* Save event kinematics for use with event display

   gATLFast->MCMaker()->Save();

}
//_____________________________________________________________________________
//* All this does is set the Muon Smear Option to 2.  This means the Smear

   gATLFast->SetSmearMuonOpt(1);
   gATLFast->SetSmearMuonFun(1);
   gATLFast->SetSmearing(0);
}

//_____________________________________________________________________________
TMCParticle* User::CurrentParticle()
{
return particle;
}
//_____________________________________________________________________________
TMCParticle* User::CurrentFirstChild()
{
return child1;
}
//_____________________________________________________________________________
TMCParticle* User::CurrentSecondChild()
{
return child2;
}
```

H.3 run.C

```
//* This is where the Phythia cards are set.  Also, the random number
//* seed, the Hadronization Mode and ClusterMaker parameters are set.
{
//* change ATLfast default parameters

//* load ATLFAST libraries
  gROOT->Macro("$ATLFAST/loadlibs");


//* Turn off default event types.  I think this is done so that
//* the exact Pythia cards can be set, but I cannot find any
//* documentation on this, or event source code for this option.

  gATLFast->MCMaker()->SetProductionMode(0);


//* The following code chooses the event type; use PYTHIA options
//* to choose top-antitop production events.

//* Set Pythia parameters:

/******************************************************************************
  // This is a bit tricky!
  // We must have an access to Pythia generator before Atlfast->Init()
  // is called. But TPythia object is normally created in Atlfast->Init() !!!
  // It doesn't matter, because we always have only one generator - and every
  // TPythia object is just another interfacing object to this generator.
  // so: If we create a TPythia object here, we create an interface to
  // generator , and we can use this instance of TPythia interface to set our
  // switches. It doesn't matter that ATLFast will create another TPythia
  // object - it will create just another interface to THE SAME Pythia
  // generator, not changing anything without our commitment. And of course
  // it doesn't matter that it will use its own interface - not this one...
  // Once again:
  // using gATLFast->MCMaker->Generator() would return NULL pointer , because
  // TPythia object is not created in ATLFast::ATLFast(). We create just
  // another interfacing object to communicate with Pythia - we can make it
  // local - it won't be useful after exiting this routine.
******************************************************************************/

//* Create a TPythia object to run Pythia and set Pythia Cards.

  TPythia *pythia = new TPythia();

//* See the Pythia Manual for more information on the Pythia cards.
```

```
//* Equivalent Pythia Commands are provided here.

  pythia->SetMSEL(0);           //MSEL=0        Processes must be switched
                                //              on in MSUB.

//* This chooses top-antitop production events.
  pythia->SetMSUB(81,1);        //MSUB(81)=1
  pythia->SetMSUB(82,1);        //MSUB(82)=1

//* This chooses Drell-Yan W production.
//  pythia->SetMSUB(2,1);       //MSUB(2)=1

  pythia->SetMSTP(48,2);        //MSTP(48)=2    top decays before
                                //              fragmentation
  pythia->SetMSTP(51,5);        //MSTP(51)=5    parton-distribution
                                //              density=5
  pythia->SetMSTP(61,0);        //MSTP(61)=0    turn initial state
                                //              radiation off
  pythia->SetMSTP(71,0);        //MSTP(71)=0    turn final state radiation
                                //              on
  pythia->SetMSTP(81,0);        //MSTP(81)=0    turn multiple interactions
                                //              off

//* Switches for possible decays.
  for(Int_t i=172; i<192 ;i++)          //Example:
     pythia->SetMDME(i,1,1);            //MDME(172,1)=1


  for(i=41; i<48 ;i++)                  //Example:
     pythia->SetMDME(i,1,0);            //MDME(41,1)=0

  pythia->SetMDME(46,1,1);              //MDME(46,11)=1

  pythia->SetMDME(188,1,1);            //MDME(188,1)=1
  pythia->SetMDME(189,1,1);            //MDME(189,1)=1


//* This is the default Hadronization Mode.  It set the mode to
//* "ISR+FSR+HP+HADRONIZE".
//* See:
//* http://root.cern.ch/root/html/atlfast/src/ATLFMCMaker.cxx.html
//* for other options.

  gATLFast->MCMaker()->SetHadronisationMode(3);

//* This changes the random number seed (default=51111).  I don't know
//* what the valid range for random numbers is, but I think it's the same
```

```
//* as the Pythia range (between 0 and 900000000, I think).

  gATLFast->MCMaker()->SetRndNumber(51112);


//* change ATLfast makers' parameters


//* Turn atlfast smearing off
  gATLFast->SetSmearing(0);
//* Make muon smearing exact (i.e. no smearing)
  gATLFast->SetSmearMuonFun(1);
//* If there is muon smearing, smear tracks.
  gATLFast->SetSmearMuonOpt(2);

//* Save event kinematics for use with event display

  gATLFast->ClusterMaker()->Save(0);
  gATLFast->TrackMaker()->Save(0);
  gATLFast->MCMaker()->Save(1);
}
```

H.4 User_main_TTbar.C

```
//*  This is an example of a main program/macro to run ATLFast

//*  This Macro loads a shared library created by the user.
//*  The library fills user variables which are then entered
//*  into the tree defined in this Macro.

{

//* Reset  root memory and load ATLFAST libraries

   gROOT->Reset();
   gROOT->Macro("$ATLFAST/loadlibs");

//* Open the root output file -- not necessary
//  TFile file("atlfast.root","recreate","ATLFast root file",1);

//* Create main object to run atlfast
   ATLFast atlfast("atlfast");

//* Load user's dynamic library.  This is created with Make.
   gSystem->Load("libUser.so");



//* Define pointers "userWplus" and "userWminus". This is a member of the
//* User class defined in the libUser.so library

   User *userWplus  = new User();
   User *userWminus = new User();


//* create output file
   hfile = new TFile("htree.root","RECREATE","Demo ROOT file with
                     working trees");

//* Initialize user routine -- sets Pythia cards.
   userWplus->Init();
   userWminus->Init();


//* Initialise event (maker histograms,etc)
   atlfast.Init();


//* Create a tree with two branches. Branch is defined as a reference
```

```
//* to the user object. So, anything filled from operations
//* on the user object will appear in the the branches of the tree.
//* See User.h for class definitions.

   tree = new TTree("Tree","Example Tree");

//* Autosave when 1 Gbyte written
   tree->SetAutoSave(1000000000);

//* Create two branches of type user.  This branch is for
//* the W information, but it also contains information on the
//* two W decay products.

   tree->Branch("Wplus_Branch", "User", &userWplus, 16000, 1);
   tree->Branch("Wminus_Branch", "User", &userWminus, 16000, 1);


//* Create the Root tree -- not necessary
//   atlfast.MakeTree();


//* Initialize PYTHIA
   TPythia *pythia = (TPythia *)gATLFast->MCMaker()->Generator();


//* Variable for W and Children of W line numbers and KF codes.
//* These are used in the calls to the User routines.
   Int_t W, Child1, Child2;
   Int_t Wplus = 24;
   Int_t Wminus= -24;
   Int_t Parent= 6;     // Uses absolute value of parent
   Int_t nevents=100;


//* loop over events
   for (Int_t i=0; i<nevents; i++)
      {
//* Generate and Reconstruct event
      atlfast.Make(i);
//* Print event listing for first evennt only.
      if (i==0) pythia->LuList(1);

//* Fill user variables. In this case, find the W and W decay
//* particles, and fill this information into tree.  This is
//* done for every event.

      userWplus->GetParticle(Wplus, Parent, W);
```

```
      if (userWplus->CurrentParticle()!=NULL)
        userWplus->GetChildren(W, Child1, Child2);

      userWminus->GetParticle(Wminus, Parent, W);
      if (userWminus->CurrentParticle()!=NULL)
        userWminus->GetChildren(W, Child1, Child2);

//* Fill the tree this event
        if ((userWplus->CurrentFirstChild()!=NULL)&&
            (userWminus->CurrentFirstChild()!=NULL))
          tree->Fill();

//* Clear reconstructed event lists
      atlfast.Clear();
   }

   atlfast.Finish();
//* save objects in Root file -- not necessary
//   atlfast.Write();  //save main atlfast object (and run parameters)


//* Print summary of the tree, then write it out to a file
   tree->Print();
   tree->Write();

//* close file
   hfile->Close();
}
```

H.5 Plot_TTbar.C

```
{
//* Reset  root memory and load ATLFAST libraries
   gROOT->Reset();
   gROOT->Macro("$ATLFAST/loadlibs");

//* Create main object to run atlfast

  ATLFast atlfast("atlfast");

//* load user's dynamic library

  gSystem->Load("libUser.so");


//* Connect file generated by User_main_W.C

   TFile *file = new TFile("htree.root");

//* Create four useful histograms

   TH1F *h1 = new TH1F("h1","Child Particle Code",39,-19.5,19.5);
   TH1F *h2 = new TH1F("h2","Child Pt",120,0,120);
   TH1F *h3 = new TH1F("h3","Muon Pt",120,0,120);
   TH1F *h4 = new TH1F("h4","W Particle Code",101,-50.5,50.5);

//* Variables needed to calculate transverse momentum.

   Float_t Px;
   Float_t Py;
   Float_t Pt;

//* Create a User instance to access User functions

   User *myuser = new User;

//* Set the branch to loop over to be "Wplus_Branch" (title set in
//* User_main_W.C)

   TBranch *branch2  = Tree.GetBranch("Wplus_Branch");
   branch2->SetAddress(&myuser);

//* Now get the number of events in the tree

   Int_t nevent = Tree.GetEntries();
```

```
//* Start main loop on all events
      Px=myuser->CurrentSecondChild()->GetPx();
      Py=myuser->CurrentSecondChild()->GetPy();
      Pt=sqrt(Px*Px+Py*Py);

      h2->Fill(Pt);

      if (abs(myuser->CurrentFirstChild()->GetKF())==13)
        h3->Fill(Pt);

      h4->Fill(myuser->CurrentParticle()->GetKF());

   }
//*  Draw the histograms on one canvas.

   TCanvas *c1 = new TCanvas("c1"," ",200,10,600,480);
   c1->Divide(2,2);
   c1->cd(1);  h1->Draw();
   c1->cd(2);  h2->Draw();
   c1->cd(3);  h3->Draw();
   c1->cd(4);  h4->Draw();

}
```

H.6 LinkDef.h

```
#ifdef __CINT__

#pragma link on all globals;
#pragma link on all classes;
#pragma link on all functions;

#pragma link C++ class User;

#pragma link C++ class ATLFast;
#endif
```