T.C.
KAFKAS ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
FİZİK ANABİLİM DALI

An Object-Oriented Data Analysis Framework (ROOT) ve CMS-CERN Deneyinde HF Detektörleri ile İlgili Kaynak Data Analizi

YÜKSEK LİSANS TEZİ
Serkan ULU

DANIŞMAN
Doç. Dr. MİTHAT KAYA

Ekim-2006

Üniversite : Kafkas Üniversitesi
Enstitü : Fen Bilimleri Enstitüsü
Anabilim Dalı : Fizik
Program : Yüksek Enerji ve Plazma Fiziği
Tez Danışmanı : Doç. Dr. Mithat KAYA
Tez Türü ve Tarihi : Yüksek Lisans – Ekim 2006

ÖZET

An Object-Oriented Data Analysis Framework(ROOT) ve CMS-CERN Deneyinde HF Detektörleri ile İlgili Kaynak Data Analizi

Serkan ULU

Bu çalışmada, detaylı olarak ROOT programı ve İleri Hadron kalorimetresinin kaynak kalibrasyonu anlatılmıştır. ROOT, C/C++ yorumcusuna sahip, bir programlama ve uygulama terminali olup CERN'de NA49 deneyi tarafından geliştirildi.

Hadronik ileri kalorimetre CMS'deki bileşik detektör sistemlerinden biridir. HF kalorimetre erimiş silika merkezli çelik soğuruculara yerleştirilmiş optik fibere dayanır. Bu çalışmada radyoaktif kaynak (Co^{60}) kullanılarak CMS ileri kalorimetrenin (HF) kalibrasyonu için üç farklı metot gösterilmistir.

University	: Kafkas University
Institute	: Institute of Science and Technology
Science Programme	: Physics
Programme	: High Energy and Plasma Physics
Supervisor	: Doç. Dr Mithat KAYA
Degree Awarded and Date	: Master of Science – October-2006

ABSTRACT

An Object-Oriented Data Analysis Framework (ROOT) and Source Data Analysis Related to HF Detector in CMS-CERN Experiment

Serkan ULU

In this study, the details of ROOT programming and source calibration of Forward Hadron calorimeters were studied. ROOT is a programming and an executing terminal which has C/C++ interpreter. ROOT was developed in the context of the NA49 experiment at CERN.

Hadron forward calorimeter is one of compound detector systems in CMS. The HF calorimeter is based on steel absorber with embedded fused silica core optical fiber. In this study three different methods are showed for the calibration of the CMS forward calorimeter (HF) using a Co^{60} radioactive source.

İÇİNDEKİLER

KISALTMALAR

CINT **:** C/C++ interpreter

bash **:** Bourne Again Shell

GIMP **:** GNU Image Manipulation Program

vi **:** Visual Editor

RMS **:** Root Mean Square

QIE **:** Charge integrator and encoder

HEP **:** High Energy Physics

CMS **:** Compact Muon Solenoid

HF **:** Hadron Forward Calorimeter

LHC **:** Large Hadron Collider

SPE **:** Single Photoelectron

EM **:** Electromagnetic

HAD **:** Hadronic

L-Fiber : Long Fiber

S-Fiber **:** Short Fiber

QCD **:** Quantum Chromodynamics

PMT **:** Photomultiplier tube

MC **:** Mean Charge

TABLO LİSTESİ

ŞEKİL LİSTESİ

Sayfa No

1. GİRİŞ

ROOT yüksek enerji fiziğinin veri analizindeki sorunları çözmeyi amaçlayan bir nesne-yöneltme çatısıdır. ROOT gelişimini sürdüren yüksek enerji fiziği veri analizlerin tüm ihtiyaçlarını yerine getirmektedir. Kurulumu hala devam eden Hadronik ileri kalorimetrenin test ışını (beam) ve kaynak kalibrasyonlarının data analizleri bu program ile yapılmaktadır.

CERN'de inşa edilmekte olan Büyük Hadron Çarpıştırıcısında (LHC) yapılacak CMS deneyi yüksek enerji fiziğinde yanıtlanamayan sorulara yanıt arayacaktır. İleri kalorimetre, CMS detektörünün bir alt detektörü olup ileri jetlerin doğrulanmasında önemli bir yer tutmaktadır.

HF ileri jetlerin doğrulanmasını ve dikine kayıp enerjinin daha hassas bir şekilde ölçülmesini sağlamaktadır. Bu sayede başta Higgs parçacığının keşfi olmak üzere birçok önemli fizik konusuna katkı sağlamaktadır. Higgs bozonunun WW veya ZZ bozonlarının füzyonu ile üretilmesi ileri iki jetin varlığıyla karakterize edilir. Bu jetler oldukça yüksek enerjili olup momentumlarının dik bileşeni yaklaşık olarak W bozonunun kütlesi mertebesindedir ve $2 < |\eta| < 5$ bölgesinde yayımlanırlar. Bu jetlerin dedekte edilmesi sinyali arttırır ve verilen bir seçim için kapsama alanını $\eta = 5$'e çıkartmak QCD fonu olan $pp \rightarrow Z +$ jetler kanalını oldukça bastırır(1).

HF kalibrasyonu birbirinden bağımsız üç donanım sistemine dayanır: Lazer kazanç izleme sistemi, LED'ler ve radyoaktif kaynak hareketleri. Bu sistemler ayrıntılı bir enerji ölçeğindeki sonuçlar için gerekli kalibrasyonu vermektedir. HF modüller çelik soğurucular içine yerleştirilmiş kuvars fiberlerden oluşmaktadır. Kuvarstaki ışığın hızından daha büyük bir hızla kuvars fiber içinde hareket eden yüklü parçacık Cerenkov etkisinden dolayı foton yayarlar. Fiberlere bağlanan detektör tarafından gözlenen ışığın miktarı parçacığın hızına, olay açısına ve parçacığın yörüngesi ile fiber merkezi arasındaki mesafeye bağlıdır.

2. LINUX SİSTEMLERİNİN TEMEL DİZİN YAPISI

Linux işletim sisteminde tüm veriler, programlar ve her biri aslında bir program olan komutlar dosyalarda (file), dosyalarda dizinlerde gruplanmış olarak saklanır. Tüm UNIX'lerde olduğu gibi Linux'ta da dosya-dizin isimlerinde kullanılacak harflerin büyük veya küçük olması farklıdır ve önemlidir. Olası karışıklıkları önlemek için mümkün olduğunca küçük harflerden oluşan dosya ve dizin isimleri kullanılır. Linux'ta dosya ve dizin isimlerinde "uzantı" (extension) yoktur. Yani bir dosyanın adının uzantısına bakıp o dosyanın bir program dosyası olup olmadığı anlaşılamaz(2).

Bulunduğunuz dizinde ls komutunu verdiğiniz zaman dizin içinde bulunan diğer dizin ve dosyalar terminal ekranına gelir. Bu listede dizinler, isimlerinin sonuna yerleştirilen " / " karakteriyle, program veya komut dosyalarıysa " * " ile belirtilmiş olarak görüntülenir. Herhangi bir eki olmayan isimler ise program dosyası veya ismi olmayan sıradan dosyalardır.

Bir Linux sistemi yüklendiği zaman yükleme programı belirli dizinler oluşturarak bu dizinlere çeşitli dosyaları yerleştirir. Linux sistemlerinde dizin geçişleri \ ile değil / ile yapılmaktadır. En dıştaki dizine kök dizin (root directory) denilir.

/bin Dizini

Bu dizinde sistem için gerekli çalışabilen dosyalar bulunur. Tipik Linux komutları bu dizindedir.

/dev Dizini

Bu dizinde sistem için gerekli olan aygıt dosyaları bulunur.

/etc Dizini

Bu dizin içinde çeşitli konfigürasyon dosyaları bulunur.

/lib Dizini

Burada uygulama programlarının kullandığı kütüphane dosyaları bulunur.

/home Dizini

Burada kullanıcılara ilişkin dizinler bulunur.

/mnt Dizini

Bu dizin mount işlemleri için ana dizin biçiminde düşünülmüştür.

/usr Dizini

Bu dizinde kullanıcılar için gerekli olan uygulama programları tarafından kullanılan çeşitli dosyalar yer almaktadır.

/tmp Dizini

Geçici dosyalar için düşünülmüş bir dizindir.

Özellikle sistem yönetimine yönelik bir iş yaparken doğrudan LINUX işletim sistemine komut vermek tercih edilir. Bunun için de "konsol" işlevi gören bir uygulama penceresine gereksinim duyulmaktadır. "Konsol" eskiden büyük bilgisayarları denetlemek için kullanılan grafik özellikleri olmayan, doğrudan sisteme bağlı, genellikle de sistemin üzerinde bulunan bir ekran ve bir klavyeden oluşan bir cihazdır. Artık " terminal donanımları" pek kalmadı ama "konsol" kavramı aynen kullanılmaya devam ediyor. İstenildiği kadar konsol penceresi açılabilir. Her biri tamamen bağımsız "terminaller" gibi çalışırlar. Üstelik her birinin görsel özellikleri de farklı olabilir(3).

Kabuk (shell) kavramı UNIX için iyi anlaşılması gereken bir kavramdır. Birçok işletim sisteminden farklı olarak UNIX'te kullanıcının tercihine bağlı olarak kullanabileceği birden fazla komut yorumlayıcısı (kabuk=shell) vardır. Bu kabuklara örnek olarak

sh Bourne shell
csh C-Shell
ksh Korn Shell
bash Bourne Again Shell
tcsh Geliştirilmiş C-Shell

gösterilebilir. Linux dünyasının en çok beğenilen ve aksi belirtilmedikçe kullanıcıların her terminal bağlantısında varsayılan (default) kabuk olarak başlatılan “bash” kabuk programıdır. Kullanılan kabuğun hangisi olduğunu öğrenmek için hazır işaretinin karşısına **echo $SHELL** komutu yazılır. Hazır işareti **(prompt ([**user@localhost **~]$))** Linux’un komut almaya hazır olduğu anlamına gelir(4).

Hangisi olursa olsun kabuk programları, oldukça gelişmiş bir o kadar da karmaşık yeteneklerle donatılmıştır. Kabuk programları ile uygulama programları bile yazılabilir. Gelişmiş programlama dillerinde yer alan “while”, “if”, “case”, “call” gibi programlama araçlarının hepsi şu veya bu şekilde kabuk programlarında da bulunmaktadır.

3. LİNUX TERİMLERİ

Kernel (Çekirdek) : Aslında Linux, pek çok kullanıcının kafasında masaüstüyle, uygulama yazılımlarıyla, sunucu programlarıyla komple bir işletim sistemi olarak girmiş durumdadır. Ancak durum biraz farklıdır. Linux, sistemin yönetimini ele alan çekirdeğin adına deniyor. Derlenmiş hali yaklaşık 500 KB tutan çekirdek, kaynak kodlarıyla sıkıştırılmış halde 20 MB’den fazla yer kaplar. Bu kadar alanda tüm donanım sürücüleri de bulunmaktadır.

GIMP: GIMP, Linux altında en gelişmiş serbest yazılım grafik programıdır. Açılımı GNU görüntü çalıştırma programı (GNU Image Manipulation Program) olarak yazılabilir.

CopyLeft : Copyright'ın tam tersi şekilde, bir yazılımın istenildiği gibi kopyalanabileceğini, kaynak kodunun serbestçe dağıtılabildiği yazılım lisansına verilen isimdir. Bu serbest lisans, bilgiyi daha geniş kitlelere çok daha kısa sürede yaymak için birebir bir çözüm haline geliyor. Aslında GNU GPL'nin diğer adı da CopyLeft'tir.

Root : Sistemdeki en yetkili kullanıcı, bir başka deyişle sistemin yöneticisidir. Linux işletim sistemini yönetmek için root kullanıcısı parolasına gerek vardır. Genellikle ağ

servisleri, sistem servisleri, kullanıcı açma ve kapama, yedekleme yapma için tüm işlemler root kullanıcısının görevidir ve sistemin tüm sorumluluğu da bu kullanıcıdadır.

Compilation (Derleme) : Kaynak kodun çalıştırılabilir şekle getirmesine denir. Eğer bir Linux yazılımını kaynak kod halinde bulunuyorsa, öncelikle kullanıma sokmak ve makinenin anlayabileceği dile dönüştürmek için bu kod derlemelidir. Derlemek için pek çok araçlar bulunur. Bunların tamamı kullanılan Linux CD'sinde vardır. Normalde Linux kurmak ve kullanmak için C bilmek ya da derleme yapmak zorunda değilsiniz. Bu işlemler sadece çok ileri düzey kullanıcılar içindir.

4. ROOT'TA KULLANILAN C/C++ TERİMLERİ

ROOT komut satırı yorumcusu ve indis işlemcisi CINT (C++ Interpreter) alt yapısına sahiptir. Yorumcu bir programı alır her bir talimatı inceleyerek programı dışa taşır ve makine dilinin anlayabildiği şekilde uygulayarak geri döner. Bu C/C++ programlama dillerinde kullanılan terim ve komutların ROOT'ta geçerli olduğu anlamına gelir(5). ROOT dosyasının anlamlı en küçük parçalarına "atom" denilir. Atomlar birkaç guruba ayrılabilir. C/C++ yapısındaki ROOT için atomlar aşağıdaki gibi sıralanabilir.

Anahtar Sözcükler (Keywords) : Anahtar sözcükler programlama dili tarafından daha önceden belirlenmiş anlamlara sahip atomlardır. Bu atomlar kaynak dosya içinde farklı anlamlara gelecek şekilde kullanılamazlar. Örnek olarak standart ANSII C dilinin 32 anahtar sözcüğü vardır.

** `auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while`

C ve C++'da bütün anahtar sözcükler küçük harf olarak tanımlanmıştır.

İşleçler : İşleçler **(**örneğin +, -, *, /, <=, => **)** önceden tanımlanmış bir takım işlemleri yapan atomlardır. C dilinde 45 tane işleç vardır. Bu işleçlerden bazıları iki karakterden oluşurlar, bu karakterler bitişik yazılmalıdır eğer aralarında boşluk olursa işleçler anlamını yitirir. İşleçler mikroişlemcinin bir işlem yapmasını ve bu işlem sonunda bir değer üretmesini sağlar. Programlama dillerinde her işleç bir makine komutuna karşılık gelir.

Değişmezler : Değişmezler doğrudan işleme sokulan, değişken bilgi içermeyen atomlardır. Veriler ya nesnelerin içinde ya da doğrudan değişmez biçiminde bulunurlar. Değişmezler nesne olmayan, doğrudan sayısal büyüklük olarak girilen verilerdir. Örnek olarak; `x = y + z` ifadesi y ve z içindeki değerlerin toplanıp sonucun x değişkenine aktarılacağı anlamına gelir. Nesnelerin türleri olduğu gibi değişmezlerinde türleri vardır. Bunlar

1. Tam sayı değişmezleri [-25, 2347L(işaretli uzun tamsayı değişmezi), 15000U (işaretsiz int türümden tamsayı değişkeni)]
2. Karakter değişmezleri [‘a’, ‘J’, ‘>’ (karakter değişmezleri ‘’ arasında kullanılır.)]
3. Ters bölü karakter değişmezleri[‘\b’, ‘\0’]
4. Gerçek sayı değişmezleri[2, 10, 1000]

İsimler-Dizgeler-Ayraç ve Noktalama İşaretleri : Değişkenler, makrolar, işlevler, değişmezler gibi yazılımsal varlıklara programlama dili tarafından belirlenmiş kurallara uygun olmak koşulu ile isim verilebilir. Yazılımsal varlığa verilen isme ilişkin atomlar isimlerdir. İki tırnak içindeki ifadelere dizge denilir. Dizgeler C/C++ programlama dillerinde genellikle tek bir atom olarak alınır. Ayraç ve noktalama işaretler yukarıda sayılan atom sınıfları dışında kalan tüm atomları kapsar. Genellikle atomları birbirinden ayırmak için kullanılırlar.

5. KONTROL DEYİMLERİ

Kontrol deyimleri programın akışını değiştirebilen deyimlerdir. Kontrol deyimleri ile programın akışı farklı noktalara yönlendirilebilir. Bunlar C/C++ dilinin önceden belirlenmiş kurallarına uyarlar ve kendi söz dizimleri içerisinde en az bir anahtar sözcük içerirler. ROOT içende de yer alan en önemli kontrol deyimleri: **if** deyimi, **while** döngü

deyimi, **for** döngü deyimi, **break** deyimi, **continue** deyimi, **switch** deyimi, **goto** deyimi, **return** deyimi(6).

if Deyimi : ROOT'daki program akışını denetlemeye yönelik en önemli deyim **if** deyimidir. En basit biçimi ile genel söz dizimi [`if (söz dizimi) deyim;`] şeklindedir. if ayracı içindeki ifadeye koşullu ifade (conditional expression), ve bu ayracı izleyen deyime de if deyiminin doğru kısmı (true path) denir. Doğru kısmı oluşturan deyim bir yalın deyim (simple statement) olabileceği gibi, bir boş deyim (null statement), bir bileşik deyim (compound statement) ya da başka bir kontrol deyimi olabilir. if deyiminin yürütülmesi sırasında derleyici önce kontrol ifadesinin sayısal değerini hesaplar. Hesaplanan sayısal değer mantıksal doğru ya da yanlış olarak yorumlanır. Kontrol ifadesinin sayısal değeri 0 ise yanlış, 0 dışında bir değer ise doğru olarak yorumlanır. Yalın if deyimi bir ifadenin doğruluğuna ya da yanlışlığına göre bir deyimin yapılmasına ya da yapılmamasına dayanır. Örnek olarak

```
int main()
{
   int x;
   printf("bir sayı girin : ");
   scanf("%d",&x);
   if (x > 10);
        printf("if deyiminin doğru kısmı!\n");
   return 0;
}
```

Burada kullanılan if deyimiyle, klavyeden girilen sayının 10'dan büyük olması durumunda `printf` çağrısı yürütülür aksi halde bu çağrı yürütülmez. Bununla birlikte, if deyimi **else** anahtar sözcüğü ile birlikte kullanılabilir. Böyle if deyimine yanlış kısmı olan if deyimi denir.

```
if (ifade)
     deyim1;
else
     deyim2;
```

Burada else anahtar sözcüğünden sonra yazılan ikinci deyime if deyiminin yanlış kısmı (false path) denir. Bundan dolayı, koşul ifadesinin mantıksal değerinin doğru olması durumunda deyim1, yanlış olması durumunda deyim2 yürütülür.

```
#include <stdio.h>

int main()
{
   char ch;
   printf("bir karakter girin : ");
   ch = getchar();
   if (ch => 'a' && ch <= 'z')
      printf("%c küçük harf!\n",ch);
   else
      printf("%c küçük harf değil!\n",ch);
   return 0;
}
```

Yukarıdaki örnekte main işlevi içinde standart getchar işlevi kullanılarak klavyeden bir karakter alınıyor, alınan karakterin sıra numarası ch isimli değişkene atanıyor. Koşul ifadesinin doğru ya da yanlış olması durumuna göre, klavyeden alınan karakterin küçük ya da büyük olması durumu ekrana yansıtılıyor.

while Döngü Deyimi : while döngü deyiminin genel sözdizimi [`while (sözdizimi) deyim;`] olarak alınır. while ayracı içindeki ifadeye kontrol ifadesi (control expression), bu ayracı izleyen deyime döngü gövdesi (loop body) adı verilir. while deyiminin yürütülmesi sırasında önce kontrol ifadesinin sayısal değeri hesaplanır. Bu ifade mantıksal olarak hesaplanır; ifade eğer 0 değerine sahipse döngü gövdesindeki deyim yürütülmez, programın akışı döngü deyiminden hemen sonraki deyimle sürer. Kontrol ifadesi sıfır dışındaki bir değere sahipse, doğru olarak yorumlanır ve döngü gövdesindeki deyim yürütülür. Kontrol ifadesindeki değer sıfır dışında bir değere sahip olduğu sürece döngü gövdesindeki deyim yürütülmeye devam eder. Döngü gövdesinden ancak ifadenin yanlış olarak yorumlanmasıyla çıkılır. while döngüsü, bir koşul doğru olduğu sürece bir ya da birden fazla işin yaptırılmasını sağlayan bir döngü deyimidir.

```
#include <stdio.h>

int main()
{
   int i = 0;
   while (i < 100) {
      printf("%d", i);
      ++i;
   }
   return 0;
}
```

Bu programa göre i < 100 olduğu sürece printf işlevi çağrılır ve i'nin değeri 1 artırılır. i < 100 olduğu durumda ifade yanlıştır ve döngüden çıkılır.

for Döngü Deyimi: for döngü deyiminin genel sözdizimi [`for (ifade1;ifade2; ifade3) deyim;`] ile ifade edilir. Derleyici for anahtar sözcüğünden sonra bir ayraç açılmasını ve iki noktalı virgül atomu bulunmasını bekler. Bu iki noktalı virgül for ayracını üç kısma ayırır. Oluşan kısımların her biri birbirinden ayrı ifadelerdir. Bunun dışındaki durumlarda sözdizimi hatası oluşur. for ayracından hemen sonra döngü gövdesi yer alır. for ayracının ikinci kısmını oluşturan ifade kontrol ifadesidir ve döngünün sürdürülmesi için bu ifade kullanılır. Bu ifadenin değeri 0 dışında bir değerse, yani mantıksal olarak doğru yorumlanırsa, döngü sürer. Programın akışı for deyimine gelince, for ayracındaki birinci ifade değerlendirilir. Birinci kısımdaki ifade genellikle döngü değişkenine ilk değeri verme amacıyla kullanılır. for döngüsünün üçüncü kısmındaki ifade döngü gövdesindeki deyim ya da deyimler yürütüldükten sonra, kontrol ifadesine gelinmeden önce ele alınır. Bu kısım çoğunlukla bir döngü değişkeninin artırılması ya da azaltılması amacıyla kullanılır.

```
#include <stdio.h>

int main()
{
   int i;
   for (i = 0; i < 2; ++i)
      printf("%d",i);
   printf("\nson değer = %d\n", i);
   return 0;
}
```

Programın akışı for deyimine gelince, önce for ayracı içindeki 1. ifade ele alınır, yani i değişkenine 0 değeri atanır. Daha sonra i < 2 ifadesi kontrol edilir eğer değer mantıksal olarak doğru ise döngü gövdesi devreye geçer ve i değişkenin değeri ekrana yazılarak imleç bir alt seviyeye geçer. Bundan sonra 3. ifade kontrol edilir ve i değişkenine 1 değeri atanır yani i 1 artırılır ve 2. ifade tekrar kontrol edilir. Eğer değer yine 0 dışında bir değerse döngü tekrarlanır, eğer 0 olursa döngü sonlandırılır.

6. İŞLEMCİ KOMUTLARI

Önişlemci, kaynak program üzerinde bir takım düzenlemeler ve değişiklikler yapan bir ön programdır. Önişlemci programın bir girdisi bir de çıktısı vardır. Önişlemcinin girdisi kaynak dosyanın kendisidir, çıktısı ise derleme modülünün girdisini oluşturur. C/C++ dillerinde # ile başlayan bütün satırlar önişlemci programa verilen komutlardır. # karakterinin sağındaki sözcüklere önişlemci komutları denir. Bunlar #include, #define, #line, #pragma, #error, #if, #else, #elif, #ifdef, #ifndef, #endef, #undef olarak sıralanabilir. Önişlemci komutları anahtar sözcükler değildirler. Yani önişlemci komutları önlerindeki # işareti ile birlikte kullanıldıkları zaman anlam kazanırlar. Önişlemci amaç kod oluşturmaya yönelik hiçbir iş yapmaz, kaynak kod içinde bazı metinsel düzenlemeler yapar, kendisine verilen görevleri yerine getirdikten sonra # ile başlayan dosyaları kaynak dosyadan siler.

7. LİNUX İŞLETİM SİSTEMİNE ROOT YÜKLENMESİ

Linux sisteminde geleneksel olarak komut satırı derleme işlemi kullanılır. Yani komut ayrı bir editörde yazılır ve komut satırında derlenir. Linux sistemlerindeki C derleyicilerinin ismi **cc** (c compiler) olarak verilir. GNU projesi kapsamında yazılmış olan C derleyicisine **gcc** denilir. Aslında bu günkü Linux sistemlerinde gcc yalnızca bir C derleyicisi değil, tüm derleyicileri çalıştıran ana bir program görevindedir.

Linux sistemlerinde kullanılan C++ derleyicisi g++ olarak adlandırılır. Burada gcc programı dosya uzantısı C++'a ilişkinse otomatik g++ derleyicisini sisteme sokmaktadır. gcc ve g++ derleyicilerinin sürümleri Linux işletim sistemine yüklenecek olan C ve C++ tabanlı programlar için büyük bir önem teşkil eder. Örneğin

bilgisayarınıza yükleyeceğiniz C tabanlı ROOT yazılım ve uygulama programının 5.10 versiyonu kullanılacaksa yüklenileceği sistemin, Linux'un Redhat sürümü yüklü ise, Linux Redhat FC3 ve gcc 3.4.3, versiyon 5.10/00 seçeneğini bulunmaktadır. Bu, tercih edilen ROOT versiyonu için Linux'un Redhat FC3 sürümü ve gcc 3.4.3 olması gerektiği anlamına gelmektedir.

Linux'a ROOT yüklemek için izlenmesi gereken birkaç adım vardır. Yüklenecek olan ROOT belirlendikten ve kullanılan Linux ile gcc tutarlığı kontrol edildikten sonra, ilk adım ROOT'a ait kurulumun sisteme yüklenmesidir. Yükleme internetten veya başka bir Linux sisteminden sağlanabilir. Eğer internet üzerinden bir yükleme yapılıyorsa:

1- FTP alanına giriş alınır. Bunun için Linux terminalinde

```
prompt% ftp root.cern.ch
user:anonymous
password:<e-mail adresiniz>
```

2- Dizine gidilir ve dosyanın ikili transferi için düzenlenir:

```
ftp> cd / root
ftp> bin
```

3- Kaynakların tar-ball'ı (uygun versiyon numaralı) alınır ve FTP isteminden çıkılır:

```
ftp> get root-<version>.source.tar.gz
ftp> bye
```

4- Dağılım açılır:

```
prompt% gzip -dc root-<version>.source.tar.gz | tar -xf -
```

İnternet üzerinden kurulum alındıktan sonra sıra ROOT'un sistemde çalışır hale getirilmesine geliyor. Yine bunun içinde gerekli birkaç adım var bunlardan ilki ve en önemlisi ROOT PATH'inin (yolunun) kullanıcı tarafından üzerinde çalışılan kabuğa tanıtılmasıdır. Kabuk üzerinde işlem yapabilmek için Linux terminalinde `nano .bashrc` yazılmalıdır. Bu kabuk üzerinde değişiklik yapma ya da bir başka deyişle programların sistem içindeki çalışma yollarını belirlemeye yarar. Eğer ROOT'un sistemde /data/root dizini altında çalışacağını varsayarsak buna uygun PATH:

```
prompt% nano .bashrc
```

```
export ROOTSYS=/data/root/root
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

kabuğa eklenmelidir. Bundan sonraki işlem alınan kurulum dosyasını kabukta belirlenen yol ve dizin altında açılmasıdır

.

```
prompt% tar -xvf root_v3.04.02.Linux.RH7.3.gcc32.tar
```

komutu kullanılarak kurulum dosyası içindeki gerekli tüm dosyalar daha önceden belirlediğimiz /data/root/root dizini altında sisteme yüklenmiş olur.

8. ROOT OTURUMUNU BAŞLATMAK VE KESMEK

Root başlatmak için sistem hazır durumda iken `root` yazılabilir. Bu C/C++ yorumcusunu ROOT komut satırında CINT ile başlatır ve ROOT hazır işaretini (`root [0]`) verir.

```
% root
  ************************************************
  *                                              *

  *                 WELCOME to ROOT              *
  *                                              *
  *    Version    3.05/03      25 March 2003      *
  *                                              *
  *    You are welcome to visit our Web site     *
  *                http://root.cern.ch           *
  *                                              *
  ************************************************
FreeType Engine v2.1.3 used tor ender TrueType fonts.
Complied for Linux with thread support.

CINT/ROOT C/C++ interpreter version 5.15.80 Mar 17 2003
Type ? for help. Commonds must be C++ statements.
Enclose multiple statements between { }.
```

Sonraki pencerede gösterildiği gibi bazı komut satırı seçenekleriyle ROOT başlatmak mümkündür.

```
% root -/?
Usage:root [-l] [-b] [-n] [-q] [file1.C...fileN.C]
Options:
   -b:grafiksiz grup modunda çalış
   -n:.root'de belirtildiği gibi logon ve logoff makrolarını
uygulamaz
   -q:komut satırı indis dosyalarında işlem yaptıktan sonra çık
   -l:logo resmi (splash ekranı) göstermez
```

Örnek olarak arka planda bir indis çalıştırmak istersek -uygulamadan sonra çık ve bir dosyaya çıktıyı tekrar yönlendir- aşağıdaki sözdizimi kullanılır.

```
root –b –q myMacro.C > myMacro.log
```

Root komut satırıyla tüm elde edilebilir ROOT sınıflarına, evrensel değişkenlere ve fonksiyonlara girişi veren C/C++ yorumcusuna ulaşılabilir. Hazır durumda C++ ifadeleri yazılarak, nesneler oluşturulabilir, fonksiyonlar çağırılabilir, indisler uygulanabilir.

```
root [] 1+sqrt()9
 (double)4.00000000000e+00
root [] for (int i=0; i<4, i++)count<<"Hello"<<i<<end1
Hello 0
Hello 1
Hello 2
Hello 3
root [] .q
```

9. ROOT KOMUT SATIRI

Root komut satırları uygulanışına veya kullanım amacına göre değişik karakterler içerir. Komutların farklı türleri vardır.

1- "." ile başlayan CINT komutları;

```
root [] .?                  //tüm CINT komutlarını listeler
root [] .L <dosya ismi>     //verilen dosyayı yükler
root [] .x <dosya ismi>     //verilen dosyayı yükler, uygular
```

2- ".!" ile başlayan SHELL (kabuk) komutları;

```
root [] .! ls
```

3- C++ sözdizimi ile devam eden C++ komutları

```
root [] TBrowser *b = new TBrowser
```

C++'deki bazı şeyler standart değildir, bu nedenle CINT yorumcusu birkaç uzatmaya sahiptir. Çoklu satır komutları uygulamak için komut satırları kullanılabilir. Çoklu satır komuta başlamak için, komutun başlangıcında sola kıvrık parantez {ve sonunda sağa kıvrık parantez } kullanılmalıdır(7).

10. KURALLI İFADELER

Aşağıdaki meta-karakterler kurallı ifadelerle kullanılabilir.

'^'	anchor satırının başlangıcı
'$'	anchor satırının sonu
'.'	Birkaç karakteri eşler
'['	Karakter sınıfının başlangıcı
']'	Karakter sınıfının sonu
'^'	Eğer ilk karakterse karakter sınıfını reddeder
'*'	Kleene kapanma (eşleme 0 ya da daha fazla)
'+'	Pozitif kapanma (1 ya da daha fazla)
'?'	Seçmeli kapanma (0 ya da 1)

Wildcard (birden fazla sembol olması) kullanıldığında kurallı ifadelerin '$' (EOL) ile bitişinin '^' (BOL)'den önce olduğu farz edilir. Tüm '*' (kapanmalar), _/_. işaretini hariç tutarak, '.''den önce olduğu farz edilir. Özel işleyişi yol isimlerinin kolay eşleşmesine izin verir.

Kaçış karakterleri;

\\	geri kesik
\b	geri boşluk
\f	şekli beslemek
\n	yeni satır
\r	duruşa geri dön
\s	boşluk
\t	tab
\e	ASSII ESC karakteri ('\033')

TRegexp sınıfı bir giriş dizgisinden kurallı ifade yaratmak için kullanılabilir. Eğer `wildcard` gerçekse o zaman giriş dizgisi bir wildcard ifadesi içerir.

```
TRegexp(const char *re, Bool_t wildcard)
```

Kurallı ifadeler ve wildcard, `string`'deki kurallı ifadenin ilk olayını bulan ve yerine geri dönen,

```
Ssiz_t Index(const TString& string, Ssiz_t* len, Ssiz_t i) const
```

benzer metotlarda kolaylıkla kullanılabilir.

10.1 TObject

ROOT'da, hemen hemen tüm sınıflar TObject olarak adlandırılan sıradan temel bir sınıfın alt üyeleridir. TObject sınıfı varsayım davranışını ve ROOT sistemdeki tüm nesneler için protokolü sağlar. Bu yaklaşımdaki ana avantaj elde edilmiş sınıfların sıradan davranışlarını yerine getirir ve dolayısıyla bütün sistemin tutarlılığını sağlar.

- Object I/O (`Read(), Write()`)
- Hata işlem (`Warning(),Error(),SysError(),Fatal()`)
- Sınıflama (`IsSortable(),Compare(),IsEqual(),Hash()`)
- Denetleme (`Dump(),Inspect()`)
- Yazdırma (`Print()`)
- Çizme (`Draw(),Paint(),ExecuteEvent()`)
- Küçük işlem (`SetBit(),TestBit()`)
- Bellek Hissesi (`operator new and delete`, `IsOnHeap()`)
- Meta bilgiye giriş (`IsA(),InheritsFrom()`)
- Nesne gözden geçirmek (`Browse(),IsFolder()`)

TObject sınıfı yukarıdaki protokolleri sağlar.

11. EVRENSEL DEĞİŞKENLER

Root, oturuma uygulanan, bir evrensel değişkenler grubuna sahiptir. Örnek olarak, **gDirectory** daima güncel dizini tutar ve **gStyle** güncel tarzı tutar. Bütün evrensel değişkenler "**g**" ile başlar büyük harfle devam eder.

11.1 gROOT

TROOT'un tek örneğine evrensel gROOT yoluyla etki edilebilir ve güncel oturum için bağıl bilgiyi tutar. Aslında gROOT işaretçisi kullanılmasıyla ROOT programında oluşturulmuş tüm nesnelere giriş elde edilir. TROOT nesnesi ana ROOT nesnelerine işaret eden birkaç listeye sahiptir. Bir ROOT oturumu boyunca gROOT**,** nesneleri yönetmek için bir derleme serisini saklar. Bunlara **gROOT::**`GetListOf...` metotları ile girilebilir.

```
gROOT->GetListOfClasses()
gROOT->GetListOfColors()
gROOT->GetListOfTypes()
gROOT->GetListOfGlobals()
gROOT->GetListOfGlobalFunctions()
gROOT->GetListOfFiles()
gROOT->GetListOfMappedFiles()
gROOT->GetListOfSockets()
gROOT->GetListOfCanvases()
gROOT->GetListOfStyles()
gROOT->GetListOfFunctions()
gROOT->GetListOfSpecails()
gROOT->GetListOfGeometries()
gROOT->GetListOfBrowsers()
gROOT->GetListOfMessageHandlers()
```

Bu metotlar, nesnelerin derlemesi anlamına gelen, bir TSeqCollection'a geri dönerler ve nesne bulmak ya da listeye geçmek gibi liste uygulamaları yapmak için kullanılırlar ve üyelerin her biri için bir metot çağrılır. Örnek olarak `c1` isimli bir tablo bulmak için;

```
root [] gROOT->GetListOfCanvases()->FindObject("c1")
```

yazmak yeterlidir. Bu **TObject** için bir işaretçiye geri döner.

11.2 gPad

Grafik bir nesne daima aktif pad üzerinden çizilir. gPad'a sahip olmak için daima aktif pad'e işaret edilir. Örnek olarak, aktif pad'in dolu olan rengini maviyle değiştirmek istenirse, ama istenen rengin ismi bilinmiyorsa, gPad kullanılır

```
root [] gPad->SetFillColor(38)
```

Bununla renklerin listesi elde edilebilir.

11.3 gRandom

gRandom güncel rasgele numara üreticisine işaret eder. Varsayım olarak, bir TRandom nesnesine işaret eder. Kaynağı 0'a ayarlamak, kaynağın zamandan üretileceği, anlamına gelir. Herhangi bir diğer değer, sabit olarak kullanılabilir. Aşağıda, temel rasgele dağılımlar verilmektedir:

```
Gaus(mean, sigma)
Rndm()
Landau(mean, sigma)
Poisson(mean)
Binomial(ntot, prob)
```

Rasgele numara üreticisinin yeniden yapılandırılmasıyla ROOT oturumu kişiselleştirilebilir. gRandom silinebilir ve yeni gRandom tekrar oluşturulabilir.

```
root [] delete gRandom;
root [] gRandom = new TRandom3();     //kaynak = 0
```

TRandom, gRandom'dan türetilir ve gRandom'dan daha yüksek periyodiklik ile çok hızlı bir üreticidir.

Diğer everensel değişkenler **gFile** güncel açılmış dosyaya işaret eder, **gDirectory** güncel dizine işaret eder ve **gEnv** tüm çevre düzenleriyle güncel oturum için evrensel değişkendir.

12. ÇEVRE DÜZENİ

ROOT oturumunun davranışı `.rootrc` dosyasındaki seçeneklerle şekillendirilir. Açılışta, ROOT aşağıdaki düzende bir `.rootrc` dosyası için görünür.

- `./.rootrc //local directory`
- `$HOME/.rootrc //user directory`
- `$ROOTSYS/etc/system.rootrc //global ROOT directory`

Yukarıda aranan yollarda birden daha fazla `.rootrc` dosyası bulunursa, seçenekler yerel, kullanıcı, evrensel üstünlükle birleştirilir. `.rootrc` dosyası tipik olarak

```
#Path used by dynamic loader to find shared libraries
Unix.*.ROOT.DynamicPath:     .:~/rootlibs:$ROOTSYS/lib
Unix.*.ROOT.MacroPath:       .:~/rootmacros:$ROOTSYS/macros

#Path where to lok for TrueTypefonts
Unix.*.ROOT.UseTTFonts:      true
Unix.*.ROOT.TTFontPath:
…
#Activate memory statistics
Rint.Root.MemStat:           1
Rint.Load:                   rootalias.C
Rint.logon                   rootlogon.C
Rint.Logoff:                 rootlogoff.C
```

gibi görünür. Değişik seçenekler `$ROOTSYS/etc/system.rootrc`'de açıklanılır. `.rootrc` dosya unsurları birleştirilir.

12.1 Logon ve Logoff İndisleri

`rootlogon.C` ve `rootlogoff.C` dosyaları açılışta ve kapanışta indise yüklenir ve uygulanılır. `rootalias.C` yüklenir ama uygulanmaz. Tipik olarak, küçük yarar fonksiyonlarını içerir. Örnek olarak, ROOT dağılımlarıyla gelen rootalias`.C` indisi `$ROOTSYS/tutorials`'daki `edit(char*file)` fonksiyonunu tanımlar. Bu komut satırından editörü çağırmak için kullanıcıya izin verir. Bu özel fonksiyon, çevre değişkeni `EDİTOR` ayarlanmazsa, vi editörünü başlatacaktır.

13. HİSTOGRAMLAR

13.1 Histogram Sınıfları

ROOT aşağıdaki histogram türlerini destekler:
1-D Histogramlar:

- **TH1C** : kanal başına bir bytelı histogramlar. Maksimum içerik aralığı = 255
- **TH1S** : kanal başına bir kısalı histogramlar. Maksimum içerik aralığı = 65 535
- **TH1F** : kanal başına bir boyutlu histogramlar. Maksimum keskinlik = 7 digit (1 digit=20 milimetre)
- **TH1D** : kanal başına bir çiftli histogramlar. Maksimum keskinlik = 14 digit

2-D Histogramlar:

- **TH2C** : kanal başına bir bytelı histogramlar. Maksimum içerik aralığı = 255
- **TH2S** : kanal başına bir kısalı histogramlar. Maksimum içerik aralığı = 65 535
- **TH2F** : kanal başına bir boyutlu histogramlar. Maksimum keskinlik = 7 digit
- **TH2D** : kanal başına bir çiftli histogramlar. Maksimum keskinlik = 14 digit

3-D Histogramlar:

- **TH3C** : kanal başına bir bytelı histogramlar. Maksimum içerik aralığı = 255
- **TH3S** : kanal başına bir kısalı histogramlar. Maksimum içerik aralığı = 65 535
- **TH3F** : kanal başına bir boyutlu histogramlar. Maksimum keskinlik = 7 digit
- **TH3D** : kanal başına bir çiftli histogramlar. Maksimum keskinlik = 14 digit

Profil Histogramlar:

- **TProfile** : bir boyutlu profiller
- **TProfile2D** : iki boyutlu profiller

Profile histogramlar Y'nin ortalama değerini ve X'deki her bir aralık için RMS'sini (Root Mean Square) göstermekte kullanılır. Profil histogramlar birkaç durumda iki boyutlu histogramların zarif bir yenileyicisidir. İki ölçülü X ve Y niceliklerinin iç bağlantısı daima iki boyutlu bir histogram ya da yayılma-arsası ile göz önüne getirilebilir. X'in yaklaşık fonksiyonunun tek değeri biliniyor Y bilinmiyorsa, bir profil histogram yayılma arsasındakinden daha büyük keskinliklere sahip olabilir. Tüm histogram sınıfları temel TH1 sınıfından elde edilmiştir.

13.2 Histogram Oluşturmak

Histogramlar aşağıdaki kurucuyla oluşturulur:

```
TH1F *h1 = new TH1F ("h1","h1 title",100,0,4.4);
TH2F *h2 = new TH2F ("h2","h2 title",40,0,4,30,-3,3)
```

TH1 kurucusundaki parametreler: histogramın ismi, unvanı, aralıkların numarası, x minimum ve x maksimumdur.

Histogramlar

- Var olan histogramın `Clone` metodunun çağırılması
- 2-D ya da 3-D'den bir çıkıntı yapılması
- Dosyadan bir histogramın okunması

ile de oluşturulabilir. Histogram oluşturulduğu zaman, ona ait referans otomatik olarak açık olan dosya ya da dizin için hafızadaki nesnelerin listesine eklenir.

13.3 Sabit ya da Değişmeyen Aralık Büyüklüğü

Tüm histogram türleri sabit ya da değişken aralık büyüklüklerini destekler. 2-D histogramlar X boyunca sabit büyüklük aralıklarına ve Y boyunca değişken büyüklük aralıklarına sahip olabilir. Doldurulan, çalıştırılan, çekilen fonksiyonlar ya da girilen histogramlar her iki durumda aynıdır. Değişken aralık büyüklüğünün biriyle histogram oluşturmakta:

```
TH1 (const char name, const "title", Int_t nbins, *xbins)
```

bu kurucu kullanılabilir. Bu kurucudaki parametreler:

- `title` : histogram unvanı
- `nbins` : aralıkların numarası
- `xbins` : her bir aralık için düşük-kenar sıralanışı. Bu büyüklüğü nbins +1'li bir sıralanıştır.

Her bir histogram daima üç **TAxis** nesnesini (`fXaxis, fYaxis` ve `fZaxis`) içerir. Eksen parametrelerine ilk girişte histogramdaki eksen alınır, ve ondan sonra **TAxis** giriş parametreleri çağrılır.

```
TAxis *xaxis = h->GetXaxis();
Doube_t binCenter = xaxis->GetBinCenter(bin);
```

13.4 Histogramları Doldurmak

Bir histogram genel olarak

```
h1->Fill (x);
h1->Fill (x,w);       //ağırlıkla
h1->Fill (x,y);
h1->Fill (x,y,w);
h1->Fill (x,y,z);
h1->Fill (x,y,z,w);
```

benzeri ifadelerle doldurulur. `Fill` metodu verilen x, y ya da z çekişmesine karşılık aralık numarasını hesaplar ve verilen ağırlıkla bu aralığı artırır. `Fill()` metodu 1-D histogramlar için aralık numarasını ya da 2-D ve 3-D için evrensel aralık numarasını geri getirir. Eğer **TH1**: : `Sumw2()` doldurmadan önce çağrılırsa, alanların toplamı da biriktirilir.

13.5 Rasgele Numaralar ve Histogramlar

TH1 : : `FillRandom` var olan **TF1** fonksiyonu ya da bir başka **TH1** histogramının kullanılan içeriğini bir histograma rasgele bir şekilde doldurmak için kullanılabilir. Örnek olarak, aşağıdaki iki ifade mean 0 ve sigma 1'li yerin getirilen bir Gaussian dağılımı ile 10000 kere bir histogramı oluşturur ve doldurur.

```
TH1F h1("h1","histo from a gaussian"100,-3,3);
H1.FillRandom("gaus",10000)
```

TH1 : : `GetRandom` bir histogramın içeriklerine uygun olarak dağılan rasgele bir numarayı geri getirmekte kullanılabilir. Var olan bir histogramdaki dağılımı takip eden bir histogramı doldurmakta **TH1** : : `FillRandom`'un ikinci işareti kullanılabilir. Kesilen bu kod var olan bir histogramın (**TH1**) h olduğunu farz eder.

```
root [] TH1F h2("h2","Random Histo", 100, -3, 3);
root [] h2->FillRandom(h,1000);
```

h (**TH1**) histogramında bulunan dağılım kanal içerikleri üzerinden tamamlanılır. Daha sonra 1'e normalleştirilir. Bir rasgele numara almak:

- 0 ve 1 arasında bir rasgele numara üretilir
- r = 1 için normalize edilmiş integralde aralık bulur
- histogram kanalı doldurulur

beraberinde yukarıdaki seçenekleri de getirir. İkinci parametre (1000) kaç tane rasgele numara üretildiğini işaret eder.

13.6 Histogram Çizmek

İlk kez bir histogramın Draw metodunu (**TH1::**`Draw`) çağrıldığı zaman, bir THistPainter nesnesi oluşturulur ve histogramın bir veri üyesi olarak çiziciye bir işaretçi kaydedilir. THistPainter sınıfı histogramın çizilmesinde önemli durumdadır. CONT çizim seçeneği kullanıldığında logaritmik eksenlere (x,y,z) izin verir. THistPainter sınıfı grafiksiz histogramlara sahip olabilmek için histogramlardan ayrılır. Görüntülenen histogram tekrar doldurulduğunda, Draw metodu tekrar çağrılamaz. Şekil yenilenir sonra pad güncellenir. Pad aşağıdaki üç hareketten birinden sonra güncellenir.

- ROOT komut satır üzerinde tavır kontrol edilir
- Pad'in içine girilir
- **TPad ::**`Update` çağırılır.

Varsayım olarak, aynı histogram farklı padlerde farklı grafik seçenekleriyle çizilebilir. Görüntülenen histogram silindiğinde, şekli otomatik olarak pad den kaldırılır. Çizildiğinde histogramın bir kopyasını oluşturmak için, **TH1::**`DrawClone` kullanılır. Bu, histogramı klonlar ve klona tesir etmeden orijinal birini değiştirmeye ve silmeye olanak sağlar.

13.6.1. Çizim Seçenekleri

Aşağıdaki çizim seçenekleri tüm histogram sınıflarında desteklenir.

- "AXİS" Sadece eksen çizmek
- "HİST" Histogramlar hata verdiğinde varsayılan hata çubuğunu gösterir. Hatasız göstermek için gerekli seçenekle birlikte HIST seçeneği kullanılır.
- "SAME" Aynı pad içinde önceki resmi ilave etmek
- "CYL" Silindirik koordinatlarda kullanmak
- "POL" Kutupsal koordinatlarda kullanmak
- "SPH" Küresel koordinatlarda kullanmak
- "PSR" sahte-hız/phi koordinatlarını kullanmak
- "LEGO" Gizli satır taşınmasıyla lego arsası çizmek
- "LEGO1" Gizli yüzey taşımasıyla lego arsası çizmek
- "LEGO2" Hücre unsurlarını göstermek için renk kullanarak lego arsası çizmek
- "SURF" Gizli satır taşınmasıyla yüzey arsası çizmek
- "SURF1" Gizli yüzey taşımasıyla yüzey arsası çizmek
- "SURF2" Hücre unsurlarını göstermek için renk kullanarak yüzey arsası çizmek
- "SURF3" Üstten dış bakışla SURF'daki çizimin aynısı
- "SURF4" Gouraund gölgelemesini kullanarak yüzey arsası çizmek
- "SURF5" SURF3'ün aynısı ama sadece renkli dış hat çizilir.

Aşağıdaki seçenekler 1-D histogram sınıfları için desteklenir

- "AH" Histogram çizmek, ama eksen işaretleri ve işaret markaları yok
- "B" Çubuk grafik çizmek
- "C" Histogram aralıkları boyunca düzgün eğriler çizmek
- "E" Hata çubuğu çizmek
- "E0" 0 unsurlu aralıklar içeren hata çubuğu çizmek.

- “E1” Kenarda düşey satırlarla hata çubuğu çizmek
- “E1” Dikdörtgenle hata çubuğu çizmek
- “E3” Dikey hata çubuğunun son noktası boyunca dolu alan çizmek
- “E4” Hata çubuğunun son noktası boyunca düzgünleşmiş dolu alan çizmek
- “L” Aralık unsurları boyunca satır çizmek
- “P” Histogramın güncel markacı tarz kullanarak her bir aralıkta markacı (Poly) çizmek
- “P0” Boş aralık içeren her bir aralıkta markacı çizmek
- “*H” Her bir aralıkta * ile histogram çizmek
- “LF2” L seçeneğindeki gibi ama dolu alana histogram çizmek
- “9” Yüksek çözünürlük modunda çizilmiş olan histogram gücü
- “][” İlk ve son aralıklar için dikey satırlar olamadan histogram çizmek

Aşağıdaki seçenekler 2-D histogram sınıflarında desteklenir.

- “ARR” Ok modu. Komşu hücreler arasındaki eğimi gösterir.
- “BOX” Orantılı unsurlara yüzey ile her hücre için kutu çizmek
- “COL” Renk ölçeğini değiştirerek her hücre için kutu çizmek
- “COLZ” Çizilmiş renkli paletle COL’un aynısı
- “CONT” Dış hat arsası çizmek
- “CONT0” Dış hattı ayırt etmek için renkli yüzey kullanarak dış hat arsası çizmek
- “CONT1” Dış hattı ayırt etmek için satır tarzı kullanarak dış hat arsa çizmek
- “CONT2” Tüm dış hatlar için aynı satır tarzını kullanarak dış hat arsası çizmek
- “CONT3” Renkli dolu alanlar kullanarak dış hat arsası çizmek
- “CONT4” Renkli yüzey kullanarak yüzey arsası çizmek
- “LIST” Her bir dış hat için **TGraph** nesnelerinin listesini üretmek
- “FB” LEGO ya da SURFACE ile kullanılmış olan, ön kutuyu yok etmek

- "BB" LEGO ya da SURFACE ile kullanılmış olan, arka kutuyu yok etmek

13.6.2 İstatistik Görüntüleme

Varsayım olarak histogram çizmek istatistik kutu çizilmesini içirir. İstatistik kutuyu yok etmek için **TH1::**SetStats(kFALSE) kullanılır. İstatistik kutu çizilirse, gStyle->SetOptStat (mode) ile görüntülenmiş bilginin türü seçilebilir. mode açık (1) ya da kapalı (0) olarak ayarlanabilen yedi rakama bağlanmaktadır. mode = iourmen (default = 0001111)

- n = 1 histogramın ismini yazdırır
- e = 1 girişlerin sayısını yazdırır
- m = 1 kastedilen değeri yazdırır
- r = 1 root kastedilen alanı yazdırır
- u = 1 alt akışların sayısını yazdırır
- o = 1 üst akışların sayısını yazdırır
- i = 1 aralıkların integralini yazdırır.

"same" seçeneği ile istatistik kutu tekrar çizilemez.

14. FİZİK VEKTÖR SINIFLARI

Root ile fizik vektör sınıflarını kullanabilmek için fizik kütüphanesini yüklemek gereklidir.

```
gSystem.Load("libPhysics.so");
```

Bu pakette dört sınıf vardır. Bunlar

TVector3 genel üç vektördür. Bir **TVector3** kartezyen, kutupsal ya da silindirik koordinatlarda ifade edilmelidir. Metotlar vektörler, rotasyonlar ve artmalar arasındaki açıları nokta ya da çarpı ürünlerini, birim vektörleri ve büyüklükleri içerir. Sahte-hız, izdüşüm ve bir **TVector3**'deki çapraz bölüm gibi, HEP' de (High Energy Physics) kullanılan belirli fonksiyonlar ve parçacık çiftleri ya da konteynerlerin sabit kütlesi gibi dört vektördeki kinetik metotlarda vardır.

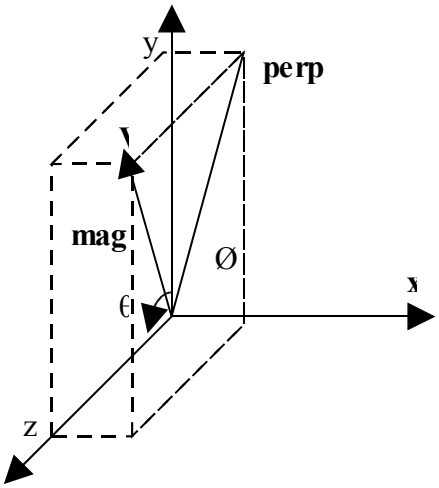
TLorentzVector ya yer ve zamanın (x,y,z,t) ya da momentum ve enerjinin (px,py, pz,E) tanımlanması için kullanılabilen, genel bir dört vektör sınıfıdır.

TRotation bir **TVector3** nesnesindeki rotasyonu tanımlayan sınıftır.

TLorentzRotation Lorentz arttırmalarını ve rotasyonlarını içeren Lorentz dönüşümlerini tanımlayan sınıftır.

CLHEP (HEP için temel bir sınıf kütüphanesi) çevrisinin parçası olmayan, iki boyuttaki bir vektörün temel yer değiştirmesi olan **TVector2**'de vardır.

14.1 TVector3



TVector3 3D'deki farklı vektörleri tanımlamak için kullanılabilen genel üç vektör sınıfıdır.

x,y,z temel unsurlardır

θ = azimuth açı

Ø = kutupsal açı

Büyüklük = mag = sqrt($x^2+y^2+z^2$)

Çapraz unsur = perp = sqrt(x^2+y^2)

TVector3 sınıfını kullanarak bazı belirli vektör değerleri için, yalnız vektörün ve gereksinim metotlarının, üç ortak özelliğini içerir.

14.1.1 Bildiri/ Unsurlara giriş

TVector3, kartezyen koordinatlar belirtilerek, üç Double_t değişkeninin bir vektörü olarak uygulanmaktadır. Default'la değer sıfıra getirilir, bununla birlikte yapıcıda koordinatlar değiştirilebilir.

```
TVector3 v1;                  //v1 = (0,0,0)
TVector3 v2(1);               //v2 = (1,0,0)
TVector3 v3(1,2,3);          //v3 = (1,2,3)
TVector3 v4(v2);            //v4 = v2
```

Double_t yada Float_t C düzeniyle **TVector3** sıfırlamakta mümkündür.

14.1.2 Diğer Koordinatlar

Küresel (rho, phi, theta) ya da silindirik (z, r, theta) koordinatlarda **TVector3**'de bilgi elde etmek için, aşağıdaki metotlar kullanılabilir:

```
Double_t m = v.Mag();
//get magnitude (=rho=sqrt(x*x+y*y+z*z))
Double_t m2 = v.Mag2();          //büyüklüğün karesini almak
Double_t t = v.Theta();          //kutupsal açı almak
Double_t ct = v.CosTheta();      //theta'nın cos'nu almak
Double_t p = v.Phi();            //azimutal açı
Double_t pp = v.Perp();          //çapraz unsur almak
Double_t pp2 = v.Perp2();        //çaprazın karesini almak
```

Başka bir vektörle ilgili olarak çaprazlama unsurunu elde etmekte mümkündür.

```
Double_t ppv1 = v.Perp(v1);
Double_t pp2v1 = v.Perp2(v1);
```

TVector3 sınıfı ekleme, çıkarma, ölçmek ve vektörleri karşılaştırmak için gerekli operatörlere sahiptir.

```
v3  =  -v1;
v1  =  v2+v3;
v1 +=  v3;
v1  =  v1-v3;
v1 -=  v3;
v1 *=  10;
v1  =  5*v2
if(v1 == v2) (...)
if(v1 != v2) (...)
```

Bu **TVector3** sınıfı için aritmetik ve karşılaştırma olarak adlandırılır.

14.2 TRotation

`TRotation` sınıfı `TVector3` nesnesinin bir rotasyonunu tanımlar. Bu `Double_t`'nin 3*3 bir matrisidir.

$$\begin{vmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{vmatrix}$$

Sözde bir aktif rotasyon tanımlar, yani nesnelerdeki bir rotasyon koordinatların durgun bir sistem içindedir. Çerçeveyi döndürmek istediğin ve dönmüş sistemdeki nesnelerin koordinatlarını bilmek istediğin takdirde, nesnelere ters dönme uygulanabilir. Eğer koordinatları dönmüş çerçeveden orijinal çerçeveye değiştirmek istenirse, direk değişim uygulanmalıdır. Belirtilmiş bir eksen etrafındaki dönme saat yönünün tersinde eksenin pozitif yönü etrafında dönme anlamına gelir.

14.2.1 Bildiri, Giriş, Karşılaştırmalar

```
TRotation r;          //r sıfırlanmış kimlik gibidir
TRotation m(r);       //m = r
```

Daima gerçek bir dönme tanımlayan **`TRotation`**'u sağlamak için, matris elemanlarını ayarlamanın direk bir yolu yoktur. Ama `XX()` `..ZZ` metotları ya da (,) operatörü ile değerler alınabilir.

```
Double_t xx = r.XX();              //aynı xx = r(0,0) gibi
         xx = r(0,0);
if(r==m)  {...}                    //eşitlik için test
if(r!=m)  {..}                     //eşitsizlik için test
if(r.Isıdentity())   {...}         //kimlik için test
```

14.2.2 Eksen Etrafında Dönme

Aşağıdaki matrisler koordinat eksenleri etrafındaki saat yönünde dönmeleri tanımlar ve `RotateX()`, `RotateY()` ve `RotateZ()`'de uygulanır.

$$Rx(a) = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) \\ 0 & \sin(a) & \cos(a) \end{vmatrix} \qquad Ry(a) = \begin{vmatrix} \cos(a) & 0 & \sin(a) \\ 0 & 1 & 0 \\ -\sin(a) & 0 & \cos(a) \end{vmatrix}$$

$$Rz(a) = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) \\ 0 & \sin(a) & \cos(a) \end{vmatrix}$$

14.2.3 Yerel Eksenlerin Dönmesi

`RotateAxes()` metodu güncel dönmeye yerel eksen dönmesini ekler ve sonuca geri döner.

```
TVector3 newX(0,1,0);
TVector3 newY(0,0,1);
TVector3 newZ(1,0,0);
a.RotateAxes(newX,newY,newZ);
```

`ThetaX(), ThetaY(), ThetaZ(), PhiX(), PhiY, PhiZ()` metotları dönmüş eksenlerin azimut ve kutupsal açılarına geri döndürür.

```
Double_t tx,ty,tz,px,py,pz;
tx = a.ThetaX()
...
pz = a.PhiZ
```

TRotation, matematiksel notasyona benzer bir **TVektor3'**ün dönmesini anlatmaya izin veren, bir `operator` sağlar.

$$\begin{vmatrix} x' \\ y' \\ z' \end{vmatrix} = \begin{vmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{vmatrix} \begin{vmatrix} x \\ y \\ z \end{vmatrix}$$

```
TRotation r;
TVector3 v(1,1,1);
V = r*v;
```

Böylelikle TVector3'ün rotasyonu elde edilir. `Transform()` metodu da kullanılabilir. Bunun için `v.Transform(r)` yazmak yeterlidir.

14.3 TLorentzVector

`TLorentzVector` hem yer ve zamanın (x, y, z, t) hem de momentum ve enerjinin (px, py, pz, E) tanımlanması için kullanılan genel dört vektör sınıfıdır. **`TLorentzVector`** **`TVector3`** ve `Double_t` değişkeninin bir seti gibi uygulanmaktadır. Varsayım olarak tüm bileşenler sıfır tarafından başlangıç durumuna getirilmiştir.

```
TLorentzVector v1;          //(0.,0.,0.,0.) tarafından
                                                sıfırlanmış
TLorentzVector v2(1.,1.,1.,1.);
TLorentzVector v3(v1);
TLorentzVector v4(TVector3(1.,2.,3.),4.)
```

Tersine uyum için `Double_t` ve `Float_t` düzeninden iki yapıcı vardır.

14.3.1 Bileşenlere Giriş

`TLorentzVector`'ün bileşenlerine [`X(),Y(),Z(),T()` ve `Px(),Py(),Pz()`ve `E()`] giriş fonksiyonlarının iki seti vardır. Setlerin ikisi de aynı değerlere geri döner ama ilk set **`TLorentzVector`**'de yer ve zamanın birleşimini tanımlamakta kullanıldığında daha uygundur ve ikinci set **`TLorentzVector`**'de momentum ve enerjiyi tanımlamakta daha uygundur.

```
Double_t xx =v.X();
...
Double_t tt =v.T();
Double_t px =v.Px();
...
Double_t ee =v.E();
```

TLorentzVector'ün bileşenlerine göstergeyle de girilebilir.

```
xx =v(0);          or      xx =v[0];
yy =v(1);                  yy =v[1];
zz =v(2);                  zz =v[2];
tt =v(3);                  tt =v[3];
```

TLorentzVector'ün vektör bileşenini elde etmek için `Vect()` metodu kullanılabilir.

```
TVector3 p = v.Vect()
```

Bileşenleri ayarlamak için iki metot vardır: `SetX(),..,SetPx(),..,`

```
v.SetX(1.);        or      v.SetPx(1.);
...
v.SetT(1.);                v.SetE(1.);
```

14.3.2 Kartezyen Olmayan Koordinatlardaki Vektör Bileşenleri

Küresel koordinat sisteminde parametrelerin **TVector3** kısmını elde etmek ve ayarlamak için birkaç yöntem vardır.

```
Double_t m, theta, cost, phi, pp, pp2, ppv2, pp2v2;
m =v.Rho();
t =v.Theta();
cost =v.CosTheta();
phi =v.Phi()
v.SetRho(10.);
v.SetTheta(TMath::Pi()*.3);
v.SetPhi(TMath::Pi());
```

ya da silindirik sistemde r-koordinatı hakkında bilgi elde etmek için:

```
Double_t pp, pp2, ppv2, pp2v2;
pp =v.Perp();              //çapraz bileşen elde etmek
pp2 =v.Perp2();            //çapraz bileşen alan elde etmek
ppv2 =v.Perp(v1);          //başka bir vektöre göre çapraz     //
                     bileşen elde etmek
pp2v2 =v.Perp(v1);
```

14.3.3 Büyüklük/Sabit kütle, beta, gama, skaler ürün

İki dört vektörün skaler ürünü (-, -, -, +) metresi ile hesaplanılır.

s = v1*v2 = `t1*t2-x1*x2-y1*y2-z1*z2`

O nedenle dört vektörün büyüklüğünün karesi `mag2`:

mag2 = v*v = t*t-x*x-y*y-z*z

olur. Eğer büyüklüğün karesi `mag2` negatif ise

mag = -sqrt(-mag*mag)

olarak ifade edilir. Metotlar:

```
Double_t s, s2,
s =v1.Dot(v2);             //skaler ürün
s =v1*v2;                  //skaler ürün
s2=v.Mag2();      or       s2 =v.M2();
s =v.Mag();                s =v.M();
```

Momentum ve enerjinin büyüklüğü durumunda sabit kütle kastedildiğinde **TLorentzVector** daha anlamlı ifadeler `M2()` ve `M()` sağlar. `Beta()` ve `Gamma()` metotları `beta` ve `gamma =1/Sqrt(1-beta*beta)`'ya geri döner.

14.3.4 Lorentz Artışı

Genel bir yöndeki artış vektörün **b=**`(bx,by,bz)` bileşenleri gibi alınabilen üç parametre ile parametre edilebilir. `x=(x,y,z)` ve `gamma=1/Sqrt(1-beta*beta)` ile, keyfi bir aktif Lorentz artışı dönüşümü (çubuk çerçeveden orijinal çerçeveye)

```
x = x' + (gamma-1)/(beta*beta)*(b*x')*b + gamma*t'*b
t = gamma(t'+b*x)
```

olarak yazılabilir. `Boost()` metodu çubuk çerçevenin orijinal çerçeveye bir artış dönüşümü yapar. `BoostVector()` zaman bileşeni tarafından bölünmüş uzamsal bileşenlerdeki **`TVector3`**'e geri döner.

```
TVector3 b;
v.Boost(bx,by,bz);
v.Boost(b);
b = v.BoostVector();       //b = (x/t,y/t,z/t)
```

Bir **`TLorentzVector`** bileşeni **`TVector3`**'e döndürmek için dört fonksiyon grubu vardır.

1-Eksen Etrafında Dönme

```
v.RotateX(TMath::Pi()/2.);
v.RotateY(.5);
v.RotateZ(.99);
```

2-Keyfi Bir Eksen Etrafında Dönme

```
v.Rotate(TMath::Pi()/4.,v1);     //v1 etrafındaki dönme
```

3-Dönmüş Çerçeveden Dönüştürme

```
v.RotateUz(direction);     //Yön bir TVector3 olmalı
```

4-TRotation Tarafından Dönme

```
TRotation r;
v.Transform(r);            //yada v* = r (v = r*v)
```

14.4 TLoreztRotation

`TLorentzRotation` sınıfı Lorentz artışlarını ve dönmelerini içeren Lorentz dönüşümlerini sağlar.

$$\texttt{lambda} = \begin{vmatrix} xx & xy & xz & xt \\ yx & yy & yz & yt \\ zx & zy & zz & zt \\ tx & ty & tz & tt \end{vmatrix}$$

Varsayım olarak kimlik matris için başlangıç durumuna getirilir, ama diğer `TLorentzRotation` tarafından, `TRotation` ya da bir artış tarafından da başlangıç durumuna getirilmiş olabilir.

```
TLorentzRotation l;        //l kimlik olarak sıfırlanmışdır
TLorentzRotation m(l);     //m = l
Trotation r;
TLorentzRotation lr(r);
TLorentzRotation lb1(bx,by,bz);
TVector3 b;
TLorentzRotation lb2(b);
```

`b=(bx,by,bz);` `gamma=1/Sqrt(1-beta*beta);` `gamma'=(gamma-1)/beta*beta` artış vektörleri ile bir Lorentz artışı için matris

$$\begin{vmatrix} 1+\texttt{gamma'*bx*bx} & \texttt{gamma'*bx*by} & \texttt{gamma'*bx*bz} & \texttt{gamma'*bx} \\ \texttt{gamma'*bx*bz} & 1+\texttt{gamma'*by*by} & \texttt{gamma'*by*by} & \texttt{gamma'*by} \\ \texttt{gamma'*bz*bx} & \texttt{gamma'*bz*by} & 1+\texttt{gamma'*bz*bz} & \texttt{gamma'*bz} \\ \texttt{gamma'*bx} & \texttt{gamma'*by} & \texttt{gamma'*bz} & \texttt{gamma} \end{vmatrix}$$

olarak elde edilir. Matris bileşenlerine giriş `XX(),XY()... TT()` metotları ve `operator(int,int)` ile mümkündür.

```
Double_t xx;
TLorentzRotation l;
xx = l.XX();               //xx bileşenini elde eder
xx = l.(0,0);              //xx bileşenini elde eder
if (l == m)(...)           //eşitlik için test
if (l != m)(...)           //eşitsizlik için test
if (l.IsIdentity()) (...) //kimlik için test
```

TLorentzVector'ü **TLorentzRotation** uygulamakta hem `VectorMultiplication()` metodu hem de * `operatörü` kullanılabilir. Aynı zamanda `Transform()` fonksiyonu ve **TLorentzVector** sınıfının `*=` operatörü de kullanılabilir.

```
TLorentzVector v;
TLorentzrotation l;
...
v = l.VectorMultiplication(v);
v = l*v;
v.Transform(l),
v *= l;                    //v = l*v
```

14.5 Örnek Analiz

Bu bölüm tipik bir fizik analizinin örneğidir. Büyük veri dosyaları birlikte kayıt altına alınır ve TSelector sınıfı kullanılarak analiz edilir. Bu indiste DESY Hamburg'da H1 işbirliğindeki dört büyük veri setini kullanır. Fizik alanları daha küçük veri setleri kullanılarak meydana getirilemeyen çalışmalar bu örnekle üretildi. ROOT Ağacında saklanılmış veriyi analiz etmek için birkaç yol vardır.

- **TTree::**`Draw` kullanmak:

  Bu küçük işler için çok uygun ve randımanlıdır. Bir **TTree::**`Draw` çağırmak bazen bir histogram üretir. Histogram otomatik olarak meydana getirilir. Seçme ifadesi komut satırında belirtilebilir.

- **TTreeViewer** kullanmak:

  Bu aynı işlevsellikle **TTree::**`Draw`'a grafiksel bir arayüzdür.

- **TTree::**`MakeClass` ile meydana getirilmiş kod kullanmak:

  Bu durumda, kullanıcı analiz sınıfının bir örneğini oluşturur. Olay döngüsü üzerinden kontrolüne sahiptir ve histogramların limitsiz numarasını meydana getirebilir.

- **TTree::**`MakeSelector` ile meydana getirilmiş kod kullanmak:

  TTree::`MakeClass` ile meydana getirilmiş koda benzer, kullanıcı kompleks analiz yapabilir. Bununla birlikte, olay döngüsünü kontrol edemez. Olay

döngüsü kullanıcı tarafından çağırılan **`TTree::`**`Process` ile kontrol edilir. Bu çözüm aşağıdaki kodla örneklenir.

```
h42->MakeSelector("h1analysis");
```

ROOT veri setlerinin her biri "h42" olarak isimlendirilmiş bir ROOT ağacı içerir. `h1analysis.h`'daki sınıf tanımı ROOT tarafından otomatik olarak meydana getirilir. Bu iki dosya üretir: `h1analysis.h` ve `h1analysis.C`. `h1analysis.C` dosyasının iskeleti kişiselleştirmekte için yapılır. `h1analysis.h` sınıfı ROOT'un **`TSelector`** sınıfından elde edilir. Aşağıdaki `h1analysis`'in üye fonksiyonları **`TTree::`**`Process` metodu ile çağrılır.

- `Begin` : Bu fonksiyon ağacın başlamasının üzerine her zaman bir döngü çağırır. Bu histogramınızı oluşturmak için uygun bir yapıdır.
- `Notify()` : Bu fonksiyon bağdaki yeni bir ağaca ilk girişte çağrılır.
- `ProcessCut` : Bu fonksiyon, gerçekten eğer giriş analiz edilmeliyse, bir flamaya geri dönmek için her bir girişin başlangıcında çağrılır.
- `ProcessFill` : Bu fonksiyon Select ile kabul edilmiş tüm girişler için giriş döngüsünde çağrılır.
- `Terminate` : Bu fonksiyon bir **`TTree`**'de döngünün sonunda çağrılır. Bu histogramınızı çizmek ve uygun hale getirmek için uygun bir yapıdır.

Bu programı kullanmak için, aşağıdaki oturumu deneyelim. İlk olarak, komut başına gerçek ve CPU zamanını göstermekte zamanlayıcısı çevrilir.

```
root [] gROOT->Time();
```

Adım A : Dört H1 veri dosyasıyla bir **TChain** oluşturlur. Bağ aşağıda uygulanmış bu kısa `h1chain.C` indisi tarafından oluşturulur. `$H1` H1 veri dizinine işaret eden bir sistem semboldür.

```
{
TChain chain("h42");
chain.Add("$H1/dstarmb.root");
      //21330730 bytes, 21920 events
chain.Add("$H1/dstarpla.root");
      //71464503 bytes, 73243 events
chain.Add("$H1/dstarplb.root");
      //83827959 bytes, 85597 events
chain.Add("$H1/dstarp2.root");
      //100675243 bytes, 103053 events
}
```

Komut satırından üst indise gidilir.

```
root [] .x h1chain.C
```

Adım B : Şimdi dört veri dizini içeren bir dizin oluşturuldu. **TChain TTree**'nin soyundan geldiğinden beri, bağdaki tüm olaylarda döngü için **TChain::**`Process` metodu çağrılır. **TChain::**`Process` metodunun parametresi oluşturulmuş **TSelector** sınıfını (`h1analysis.C`) içeren dosyanın ismidir.

```
root [] chain.Process(h1analysis.C)
```

Adım C : Aynı B adımı gibi, ama eklemede seçilmiş girişlerle olay listesi doldurulur. Olay listesi **TSelector::**`Terminate` metodu tarafından "`elist.root`" dosyasına kaydedilir. Seçilmiş olayların listesini görmek için, `elist->Print("all")` yapılabilir. Seçilen fonksiyon dosyalardaki bağda 283813 olayın dışında 7525 olay seçmektedir.

```
root [] chain.Process("h1analysis.C","filllist")
```

Adım D : İşlem yalnızca olay listesindeki girişlerdir. Olay listesi adım C tarafından oluşturulmuş `elist.root`'daki dosyadan okunur.

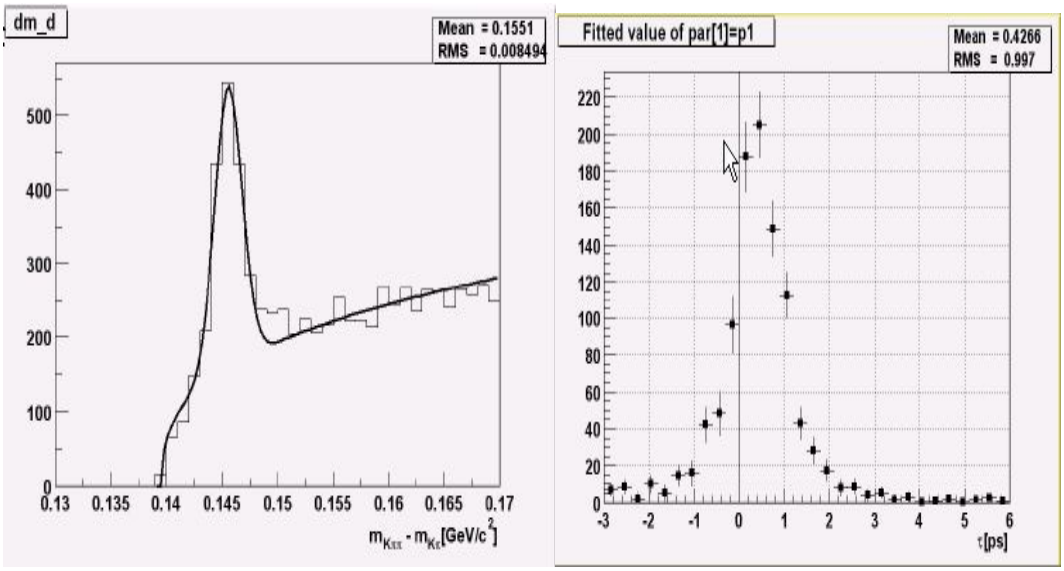
```
root [] chain.Process("h1analysis.C","uselist")
```

Adım E : Yukarıdaki adım yorumcuyla uygulanılır. "`h1analysis.C+`" ya da "`h1analysis.C++`" tarafından "`h1analysis.C`"'nin yeniden yapılandırılmasıyla ACLİC (özel fizik kütüphanesi) kullanarak B, C, D adımlarını tekrarlanabilir.

Adım F : Eğer yeni bir oturum başlarken yorumcu hızı ve ACLİC hızı arasındaki farkları görmek istenirse,

```
root [] chain.Process("h1analysis.C+","uselist")
```

uygulandığı zaman, adım 1'deki gibi bağ oluştur.
Dört farklı metotla, B, C, D ve E, uygulanmış komutları Şekil 1'de gösterilen iki çizimi üretir.



Şekil 1: Örnek analiz sonuçları

15. GİRDİ (İnput) / ÇIKTI (Output)

15.1 ROOT Dosyalarının Fiziksel Düzenlemesi

ROOT dosyası UNIX dosya dizinine benzer. Seviyelerin sınırsız sayıda düzenlenmiş dizinlerini ve nesnelerini içerebilir. Hatta makineden bağımsız formatta saklanır. Bir ROOT dosyasının fiziksel düzenlemesini görmek için, ilk biri oluşturulur. Aşağıdaki örnekte bir ROOT dosyası ve 15 histogram oluşturulur, bir Gaussian dağılımından 1000 girişle her bir histogram doldurulur ve oluşturulan dosyaya yazılırlar.

```
{
   char name[10], title[20];
   //bir histogramlar düzeni oluşturmak
   TObjArray Hlist();
   //histograma işaretçi oluşturmak
   TH1F* h;
   //15 histogram yapmak ve doldurmak
   //ve nesne düzenine onları eklemek
   for (Int_t i = 0; i < 15; i++) {
       sprintf(name,"h%d",i);
       sprintf(title,"histo nr:%d",i);
               h = new TH1F(name,title,100,-4,4);
       Hlist.Add(h);
       h->FillRandom("gaus",1000);
   }
   //bir dosya açmak ve dosaya düzeni yazmak
   TFile f("demo.root","recreate");
   Hlist->Write(),
   f.Close
}
```

TFile ROOT dosyasını tanımlama sınıfıdır. Örneğin son satırında dosya kapandı. İçeriğine göz atmak için dosya tekrar açılmış olmalı, oluşturulan TBrowser nesnesiyle ROOT nesne penceresindeki içerikler görülebilir.

```
root [] TFile f("demo.root")
root [] TBrowser browser;
```

TFile nesnesi oluşturulduktan sonra, fiziksel düzenlemeye bakmak için **TFile::**`Map()` metodu çağrılabilir. `Map()` çıktısı tarihi/zamanı, kaydın başlama adresini, kayıttaki byteların sayısını, kaydın sınıf ismini ve sıkıştırma faktörünü görüntüler.

```
root[] f.Map()
20010404/09347    At:64     N=84      TFile
20010404/09347    At:148    N=380     TH1F        CX = 2.49
20010404/09347    At:528    N=377     TH1F        CX = 2.51
20010404/09347    At:905    N=378     TH1F        CX = 2.50
20010404/09347    At:1283   N=376     TH1F        CX = 2.52
20010404/09347    At:1659   N=374     TH1F        CX = 2.53
20010404/09347    At:2033   N=390     TH1F        CX = 2.43
20010404/09347    At:2423   N=380     TH1F        CX = 2.49
20010404/09347    At:2803   N=380     TH1F        CX = 2.49
20010404/09347    At:3183   N=385     TH1F        CX = 2.46
.........
```

148 byteda ilk bir tanesinin başlamasıyla onbeş histogram görülür. 64 byte başlayan **`TFile`** giriş dosyasıdır. İlk 64 byte dosya başlığı tarafından alınır. Aşağıdaki tabloda dosya başlığı bilgileri gösterilmektedir.

Tablo 1 : Dosya başlık bilgileri

Byte	Değer İsmi	Tanımlama
1 → 4	"`root`"	ROOT dosya tanımlayıcısı
5 → 8	`fVersion`	Dosya formatı versiyonu
9 → 12	`fBEGIN`	İlk veri kaydına işaret eder
13 → 16	`fEND`	EOF'da ilk serbest kelimeye işaret eder
17 → 20	`fSeekFree`	FREE veri kaydına işaret eder
21 → 24	`fNbytesFree`	FREE veri kaydındaki byte sayısı
25 → 28	`nfree`	Serbest veri kaydının sayısı
29 → 32	`fNbytesName`	Oluş zamanında **TName'**deki byet sayısı
33 → 33	`fUnits`	Dosya işaretçileri için byte sayısı
34 → 37	`fCompress`	Zip sıkıştırma seviyesi

Dosya başlığının ilk dört baytı ROOT dosyası gibi bir dosya tanımlayan "`root`" dizini içerir. Bu tanımlayıcıdan dolayı, ROOT ".`root`" uzantısına bağlı değildir. `nfree` ve değer serbest kayıtların sayısıdır. ROOT dosyası maksimum 2 gigabyte büyüklüğe sahiptir. `FNByteFree` ile beraber bu değişken kayıtlar ve baytların kısımlarında serbest uzayın parçasını saklar. Bu hesap silinmiş kayıtları da içerir.

15.2 Histogram Kayıtları

Histogram kayıtları değişken uzunluklu kayıtlardaki 15 histogramdan oluşur.

```
20010404/09347    At:148    N=380      TH1F       CX = 2.49
20010404/09347    At:528    N=377      TH1F       CX = 2.51
...
```

Her bir kaydın ilk dört baytı bu kayıttaki byte sayısının bir tam sayı değeridir. Başlıktaki baytın geri kalanı dosyada belirsiz bir şekilde veri bloğu tanımlamak için tüm bilgileri içerir. Bu nesne verisi tarafından takip edilir. Aşağıdaki tablo her bir kişisel kayıttaki değerleri açıklar.

Tablo 2 : Kayıt tanımlama değerleri

Byte	Değer İsmi	Tanımlama
1 → 4	`Nbytes`	Sıkıştırılmış nesne uzunluğu
5 → 6	`Version`	**`TKey`** versiyon tanımlayıcısı
7 → 10	`ObjLen`	Sıkıştırılmamış nesne uzunluğu
11 → 14	`Datime`	Nesne dosyaya yazıldığındaki tarih ve zaman
15 → 16	`KeyLen`	Anahtar yapı uzunluğu
17 → 18	`Cycle`	Anahtar devri
19 → 22	`SeekKey`	Kendi kendine kaydı işaret eder
23 → 26	`SeekPdir`	Dizin başlığını işaret eder
27 → …	`lname`	Sınıf ismindeki byte sayısı
28 → …	`ClassName`	Nesnenin sınıf ismi
… → …	`lname`	Nesne ismindeki byte sayısı
… → …	`Name`	Nesnedeki isimle `lName` byte
… → …	`lTitle`	Nesne başlığındaki byte sayısı
… → …	`Title`	Nesnenin başlığı
… → …	`DATA`	Nesneye birleşmiş veri baytı

15.3 Sınıf Tanımlama Listesi (StreamerInfo List)

Histogram kayıtları sınıf tanımlamalarının `StreamerInfo` listesi tarafından takip edilir. Liste dosyaya yazılmış olan her bir sınıfın tanımlamasını içerir.

```
20010404/09347    At:5854   N=2390     StreamerInfo CX = 3.41
```

`demo.root`'ta, sınıf tanımlama listesi

- **TH1F**
- **TH1F** kalıntı ağacındaki tüm sınıflar
- Nesne veri üyelerinin tüm sınıfları
- Nesne veri üyelerinin kalıntı ağacındaki tüm sınıflar

için tanımlamayı içerir. Bu tanımlama **TStreamerInfo** sınıfı tarafından tamamlanır ve çoğu kez basitçe `StreamerInfo` olarak gönderilir. `Demo.root` örneği yalnızca **TH1F** nesnelerini içerir. Aşağıdaki `StreamerInfo` çıktı örneği **TH1F** tanımlamak için gerekli tüm sınıfların `StreamerInfo`'larını içerir.

```
root[] f.ShowStreamerInfo()
StremerInfo for class: TH1F, version=1
  BASE      TH1           offset=0  type=0   1-Dim histogram base class   BASE
         TArrayF          offset=0  type=0   Array of floats

StreamerInfo for class TH1, vesion=3
  BASE      TNamed        offset=0  type=67   The basis for named object BASE
         TAttline         offset=0  type=0    Line attributes
  BASE      TAttFill      offset=0  type=0    Fill area attributes
  BASE      TAttMarker    offset=0  type=0    Marker attributes
  Int_t     fNcells       offset=0  type=3    number bins(1D),cells(2D)+...
  TAxis     fXaxis        offset=0  type=61   X axis descriptor
  TAxis     fYaxis        offset=0  type=61   Y axis descriptor
  TAxis     fZaxis        offset=0  type=61   Z axis descriptor
  Short_t   fBarOffset    offset=0  type=2    (1000*offset) for barcharts...
  Short_t   fBarWidth     offset=0  type=2    (1000*width)for bar charts...
  Stat_t fEntries         offset=0  type=8    Number of entries
  Stat_t fTsumw           offset=0  type=8    Total sum of weights
  Stat_t fTsumw2          offset=0  type=8    Total sum of squares of weights
  Stat_t fTsumwx          offset=0  type=8    Total sum of weights*X
  Stat_t fTsumwx2         offset=0  type=8    Total sum of weights*X*X
  Double_t  fMaximum      offset=0  type=8    Maximum value for plotting
  Double_t  fMinimum      offset=0  type=8    Minimum value for plotting
  Double_t  fNormFactor   offset=0  type=8    Normalizetion factor
  TArrayD   fContour      offset=0  type=62   Array to display contour levels
  TArrayD   fSumw2        offset=0  type=62   Array ofsum of squares ofweight
  TString   fOption       offset=0  type=65   histogram options
  TList* fFunction        offset=0  type=63   ->Pointer to list of funtions
                                                    (fits and user)
StreamerInfo for class: TNamed, version=1
...
StreamerInfo for class: TAttLine, version=1
...
StreamerInfo for class: TAttFill, version=1
...
```

ROOT çoklu versiyonlara sahip olmak için bir sınıfa izin verir ve her versiyon `StremerInfo` şeklindeki kendi tanımana sahiptir.

15.4 Mantıksal ROOT Dosyası : TFile ve Tkey

Yalnızca ardışık girişleri desteklemek mümkün değildir bundan dolayı ROOT rasgele ve direk girişleri sağlar. Böyle yapmak için, **`TFile`**, aslında dosyadaki nesneler için bir kanıt olan, **`Tkey`** listesi saklar. **`Tkey`** sınıfı dosyadaki nesnelerin kayıt başlıklarını tanımlar. Dosyada belirli bir dosya bulmak için **`TFile::`**`Get()` metodu kullanılır.

```
root[] TFile f("demo.root")
root[] f.GetListOfKeys()->Print()
TKey Name = h0, Title = histo nr:0, Cycle = 1
TKey Name = h1, Title = histo nr:1, Cycle = 1
TKey Name = h2, Title = histo nr:2, Cycle = 1
TKey Name = h3, Title = histo nr:3, Cycle = 1
TKey Name = h4, Title = histo nr:4, Cycle = 1
TKey Name = h5, Title = histo nr:5, Cycle = 1
TKey Name = h6, Title = histo nr:6, Cycle = 1
TKey Name = h7, Title = histo nr:7, Cycle = 1
TKey Name = h8, Title = histo nr:8, Cycle = 1
TKey Name = h9, Title = histo nr:9, Cycle = 1
TKey Name = h10, Title = histo nr:10, Cycle = 1
TKey Name = h11, Title = histo nr:11, Cycle = 1
TKey Name = h12, Title = histo nr:12, Cycle = 1
TKey Name = h13, Title = histo nr:13, Cycle = 1
TKey Name = h14, Title = histo nr:14, Cycle = 1
root[] TH1F *h9 = (TH1F*)f.Get("h9");
```

`TFile::``Get()` "h9" isimli **`Tkey`** nesnesini bulur. Anahtarlar **Tkey'**lerin **Tlist'**inde elde edilebildiğinde anahtar listesi üzerinde tekrarlanabilir.

```
{
 TFile f("demo.root");
 TIter next(f.GetListOfKeys());
 TKey *key;
 while ((key=(TKey*)next())){
       print("key:%s points to an object of class: %s at %d\n",
       key->GetName(),
       key->GetClassName(),key->GetSeekKey());
}
}
```

Anahtar listesine eklemede, **TFile** başka iki dosya daha saklar: **`TFile::`**`fFree` dosyadaki boşluğu serbest hale dönüştürmek için kullanılmış serbest blokların **Tlist**'idir. ROOT en iyi serbest bloğu bulmayı dener. Eğer serbest blok saklanmış olan yeni nesnenin büyüklüğü ile eşleşirse, nesne serbest bloğa yazılır ve bu serbest blok

listeden silinir. Eğer değilse, nesneden daha büyük ilk serbest blok kullanılır. **`TFile::`** `fListHead` hafızadaki nesnelerin ayrılmış listesini içerir.

16. HAFIZADAKİ NESNELER VE DİSKTEKİ NESNELER

`TFile::``ls()` metodu diskteki ("-d") nesneleri yada hafızadaki ("-m") nesneleri listeleyen bir seçeneğe sahiptir. Hiçbir seçme verilmezse, ilki hafızadaki nesneler, diğerleri de diskteki nesnelerdir.

```
root[] TFile *f = new TFile("hsimple.root");
root[] gDirectory->ls("-m")
TFile**         hsimple.root
 TFile*         hsimple.root
```

Hatırlanacağı gibi **`gDirectory`** güncel dizindir ve burada "f."ya karşılık gelir. Sonraki komut güncel dizindeki diskteki nesneleri listeler.

```
root[] gDirectory->ls("-d")
TFile**         hsimple.root
 TFile*         hsimple.root
  KEY: TH1F     hpx;1       This is the px distribuiton
  KEY: TH2F     hpxpy;1     py vs px
  KEY: TProfile hprof;1     Profile of pz vesus px
  KEY: TNtuple  ntuple;1    Demo ntuple
```

Diskten hafızaya bir nesne getirmek için, "Get" kullanılmalıdır. Örnek olarak, hprof çizmek için dosyadan okunur ve hafızada bir nesne oluşturulur. Burada profil histogram çizilir ve içerikler listelenir.

```
root[] hprof->Draw()
<TCanvas::makeDefCanvas>: created default TCanvas with name c1
root[] f->ls()
TFile** hsimple.root
TFile* hsimple.root
OBJ:TProfile hprof Profile of pz versus px : 0
KEY: TH1F hpx;1 This is the px distribuiton
KEY: TH2F hpxpy;1 py vs px
KEY: TProfile hprof;1 Profile of pz vesus px
KEY: TNtuple ntuple;1 Demo ntuple
```

OBJ ile başlayan yeni satır, bu dizine hafızadan eklenen hprof isimli, **Tprofile** sınıfı bir nesne anlamına gelir. Hafızadaki bu yeni hprof disktekinden bağımsızdır. İlk ifade ile TCanvas c1 isimli bir tuval oluşturur. Bununla birlikte yeniden oluşturulan tuval güncel dizinin içeriklerinde listelenmez. Bunun nedeni tuvali (canvas) güncel dizine eklenmez, çünkü varsayım olarak yalnızca histogramlar ve ağaçlar güncel dizinin nesne listesine eklenir. Oluşturulan tuval tuvaller listesine eklenir.

Liste g`ROOT→ GetListCanvases()→ ls()` komutu tarafından elde edilir.

```
root[] gROOT->GetListCanvases()->ls()
Canvas Name=c1 Title=c1
Option=TCanvas fXlowNDC=0 fYlowNDC=0 fWNDC=1 fHNDC=1
Name= c1 Title= c1
Option=TFrame X1=-4.000000 Y1=0.000000 X2=4.000000 Y2=19.384882
 OBJ: TProfile hprof Profile of pz versus px : 0
 TPaveText X1=-4.900000 Y1=20.475282 X2=-0.950000 Y2=21.686837 title
 TPAveStats X1=2.800000 Y1=17.446395 X2=4.800000  Y2=21.323371 stats
```

Aynı örnekle ilerlenirse ve yeni bir histogram çizilirse OBJ girişinin birden fazla olduğu görülür.

```
root[] hpx->Draw()
root[] f->ls()
TFile** hsimple.root
TFile* hsimple.root
OBJ:TProfile hprof Profile of pz versus px : 0
OBJ:TH1F hpx this distribution: 0
KEY: TH1F hpx;1 This is the px distribuiton
KEY: TH2F hpxpy;1 py vs px
KEY: TProfile hprof;1 Profile of pz vesus px
KEY: TNtuple ntuple;1 Demo ntuple
```

TFile::`ls()` hafızdaki nesnelerin listesi ve diskteki nesnelerin listesi üzerindeki döngüdür. Her iki durumda, her bir nesnenin `ls()` metodu çağrılır. `Ls()` metodunun yürütmesi nesnenin sınıfına özgüdür, bu nesnelerin tamamı `Tobject` neslindendir ve **Tobject::**`ls()` yürütmesinden sonra oluşur.

16.1 Diske Histogram Kaydetmek

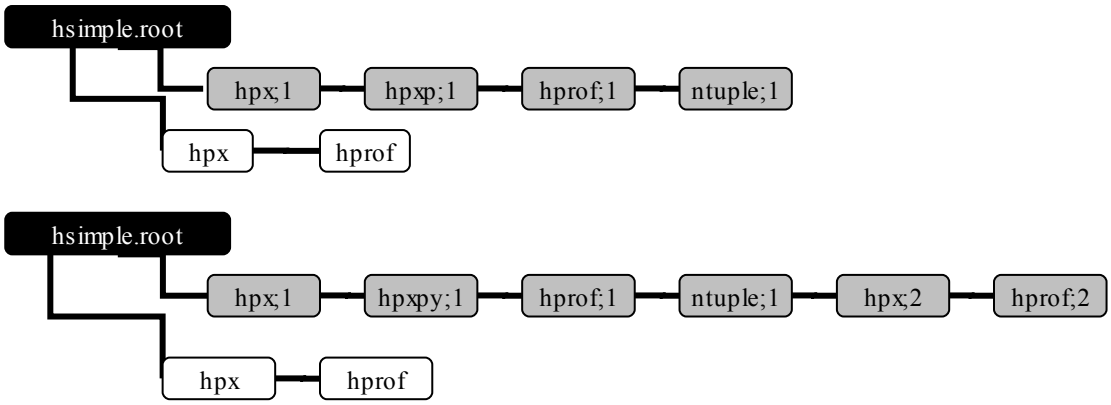
Hafızadaki (OBJ) nesneler disk (KEY) üzerindeki nesnelere özdeştir. Aynı örnek üzerinden devam edilirse, hafızadaki hpx'e doldurma eklemek için

```
root[] hpx->Fill(0)
```

kullanılır. Bu durumda, hafızadaki hpx diskteki hpx histogramından farklı olur. Nesnenin yalnızca bir versiyonu hafızada olabilir, bununla birlikte nesnenin çoklu versiyonları diskte saklanabilir. **TFile::**`Write()` metodu diske güncel dizindeki nesneleri listesini yazar ve hpx ve hprof'un yeni versiyonlarını ekler.

```
root[] f->Write()
root[] f->ls()
TFile** hsimple.root
TFile* hsimple.root
OBJ:TProfile hprof Profile of pz versus px : 0
OBJ:TH1F hpx this distribution: 0
KEY: TH1F hpx;2 This is the px distribuiton
KEY: TH1F hpx;1 This is the px distribuiton
KEY: TH2F hpxpy;1 py vs px
KEY: TProfile hprof;2 Profile of pz vesus px
KEY: TProfile hprof;1 Profile of pz vesus px
KEY: TNtuple ntuple;1 Demo ntuple
```

TFile::`Write()` metodu dosyaya güncel dizindeki nesnelerin giriş listelerini yazar. Yukarıdaki örnekte görüldüğü gibi `hpx;2` ve `hprof;2` dosyaya eklenen iki yeni anahtardır. Hafızadakine benzemeyen, bir dosya aynı isimli çoklu nesneleri saklama yeteneğine sahiptir. Bununla birlikte dosya aynı isimli nesnelerin devir numaralarını, yani noktalı virgülden sonraki numaralarını, diske farklı kaydeder. Şekil 2'de histogramın diske kaydedilmeden önce ve kaydedildikten sonra dosya üzerindeki değişiklikler şematik olarak gösterilmektedir.



Şekil 2: Kayıttan önceki ve sonraki durumlar. Şekildeki gri gruplar diskteki nesneleri beyaz gruplar hafızadaki nesneleri ve siyah gruplarda dizinleri göstermektedir.

Eğer oluşturulan birkaç farklı histogramdan yalnızca biri kaydedilmek istenirse, örneğin hpx seçeneği, bu kez sadece o histogram için **`TH1::`**`Write` metodu kullanılır.

```
root[] hpx->Write()
```

Eğer kaydedilen histogram farklı bir isimle kaydedilecekse `Write("yeniisim")` uygulaması yapılmalıdır. Diske yazılan nesne üzerinde değişiklik yapılmışsa ve bu değişiklikler daha önce diske kaydedilen nesne üzerine tekrar yazılacaksa

```
root[] hpx->Write("",TObject::kOverwrite)
```

seçeneği kullanılabilir. Bununla birlikte disk üzerine nesneleri ve koleksiyonları kaydetmek içinde `Write()` seçeneği kullanılır. Disk üzerine nesne kaydetmek için **`TDirectory::`**`Write()` ve koleksiyon kaydetmek için **`TCollection::`**`Write()` metodları kullanılmalıdır.

17. AĞAÇLAR(TREES)

Girdi/çıktı bölümünde, nesnelerin ROOT dosyalarına nasıl kaydedilebildiği gösterildi. Aynı sınıf nesnelerin büyük nicelikleri kaydedilmek istendiği takdirde, ROOT **TTree** ve **Tntuple** sınıflarını kullanır. **TTree** sınıfı disk uzayını azaltmak ve giriş hızını arttırmak için kullanılır. **Tntuple**, yalnızca yayılan nokta numaraları tutmak için sınırlanan, bir **TTree**'dir. **TTree** kullanıldığında, yaprak (**leaf**) verili dal (**branch**) tamponlar oluşur ve tamponlar, dolduğunda, dosyaya yazılır.

TTree her bir nesnedeki başlığı azaltır, ama sınıf isimlerini kapsar. Sıkıştırma kullanılarak, her bir aynı sınıf nesnenin sınıf isimleri sıkıştırılabilir. **TTree** ve sıkıştırma kullanılarak başlık orijinal 60 byte göre yaklaşık 4 byte azaltılır. Bununla birlikte, sıkıştırma kapanırsa, bu büyük kayıtlar görülmez. **TTree** veri girişlerini kullanmakta da kullanılır. Bir ağaç (**tree**) bir dal hiyerarşisi kullanır ve her bir dal birbirini etkilemeden herhangi bir diğer daldan okunabilir.

Ağaç ve dal kombinasyonu kullanılarak elde edilen sonuçlar içerisinden istenilen sonuçlar alınabilir. Örnek olarak, P_x ve P_y'yi olayın veri üyeleri olduğunu farz edilirse, her olay ve histogram sonucu için $P_x^2 + P_y^2$ hesaplamak istenirse **TTree** kullanılmadan milyonlarca sonuç elde edilebilir. Ama P_x'i kapsayan bir dal ve P_y'yi kapsayan başka bir dal ile bir ağaç kullanılırsa, yalnızca P_x ve P_y dalları okunarak tüm P_x ve P_y değerleri okunabilir. Basit bir **TTree** örneği olarak CERN'de personeller hakkındaki istatistikleri içeren ASCII dosyasındaki **TTree**'yi aşağıdaki gibidir.

```
{
// example of macro to read data from an ascii file and
//create a root file with an histogram and a TTree
  gROOT->Reset();

//the structure to hold the variables fort he branch
  struct staff_t {
                Int_t cat;
                Int_t division;
                Int_t flag;
                Int_t age;
                Int_t service;
                Int_t children;
                Int_t grade;
                Int_t step;
                Int_t nation,
                Int_t hrweek;
                Int_t cost;
};
staff_t staff;
//open the ASCII file
  FİLE *fp = fopen("staff.dat","r");
  Char line(81);
//create a new ROOT file
  TFile *f = new TFile("staff.root","RECREATE");
//create a TTree
  TTree *tree = new TTree("tree","staff data from ascii file");
//create one branch with all information from the structure
  tree->Branch("staff",&staff.cat,"cat/I:division:flag:age:
                service:children:grade:step:nation:hrweek:
          cost");
//fill the tree from the values in ASCII file
  while (fgets(&line,80,fp)) {
      sscanf(&line[0],"%d%d%d%d",
             &staff.cat,&staff.division,&staff.flag,&staff.age);
      sscanf(&line[13],"%d%d%d%d",&staff.service,
                     &staff.children,&staff.grade,&staff.step);
      Sscanf(&line[24],"%d%d%d%d",&staff.nation,
                     &staff.hrweek,&staff.cost);
 tree->Fill();
}
//check what the tree looks like
 tree->Print();
 fclose(fp);
 f->Write();
}
```

Bu uygulama bir ASCII dosyasını açar, bir ROOT dosyası ve bir TTree oluşturur. Ayrıca burada **TTree::**`Branch()` metodu ile bir dal oluşturulur. `Branch()` metodunun ilk parametresi dalın ismidir, ikinci parametresi okunan ilk yapraktaki adrestir. Dal bir kez tanımlandığı zaman, indis `staff_t` yapısındaki ASCII dosyasından veri okunur ve ağacı doldurur. Son satırlarda ise sırasıyla ASCII dosyası kapatılır ve ROOT dosyası diske kaydedilen ağaca (tree) yazılır.

Ağacın bir girişine erişmek için en kolay yol **TTree::**`Show()` metodunu kullanmaktır. Örnek olarak, `staff.root` ağacıdaki onuncu girişe bakmak için:

```
root[] TFile f("staff.root")
root[] tree->Show(10)
======> EVENT 10
cat             = 361
division        = 9
flag            = 15
age             = 51
service         = 29
children        = 0
grade           = 7
step            = 13
nation          = 7
hrweek          = 40
cost            = 7599
```

17.1 Sıkıştırma ve Performans

ROOT iyi bilinen gzip algoritmasına dayanan sıkıştırma algoritmasını kullanır. Sıkıştırmanın dokuz seviyesini destekler. Sıkıştırma seviyesi **TFile::** `SetCopmressionLevel` metodu ile ayarlanabilir. Yerine getirme için seviyelerden birinin seçimi nesne gibi disk uzayına kaydedilen dosyaları okumak ya da yazmak için alınan zaman arasındaki uyuşmadır. Sıkıştırma olmadığını belirtmek için seviye sıfıra ayarlanır. Sıkıştırma programın performansı üzerinde büyük bir etkiye sahip olabilir. Sıkıştırma faktörü, disk uzayına kayıtlı, veri türleri ile değişir. Değer yalnızca bir kez yazıldığı için aynı değer dizisi ile bir tampon sıkıştırılır. Örnek olarak, bir parçacık daima aynı olan bir pion kütlesine (2.5×10^{-28} kg) ve ya pozitif ya da negatif olan bir pion yüküne sahip olsun. 1000 pion için kütle yalnızca bir kez ve yük yalnızca iki kez yazılır. Veri seyrekleştiği zaman, yani birkaç sıfır olduğunda, sıkıştırma faktörü yükselir.

Tablo 3 : Örnek sıkıştırma seviyeleri

Sıkıştırma seviyesi	Byte	Yazma Zamanı (saniye)	Okuma Zamanı (Saniye)
0	1,004,998	4.77	0.07
1	438,366	6.67	0.05
5	429.871	7.03	0.06
9	426	8.47	0.05

Bir nesnenin sıkıştırılmama zamanı sıkıştırma zamanına göre küçüktür ve seçilmiş sıkıştırma seviyesinden bağımsızdır. Sıkıştırma seviyesi herhangi bir zamanda değiştirilebilir ama yeni sıkışma seviyesi yalnızca yeniden yazılan nesneye uygulanır. Bu nedenle, ROOT dosyası farklı sıkıştırma seviyeli nesneleri içerebilir.

17.2 Dallar (Branches)

Dallar için sınıflar TBranch olarak adlandırılır. Dalların organizasyonu önceden tahmin edilen kullanım için veriyi optimize etmekte tasarlayıcıya izin verir. İki değişen birbirinden bağımsız ise, değişkenler birlikte kullanılamaz. Bu durumda değişkenler ayrılmış dallarda yapılandırılır. Bununla birlikte değişkenler benzerlerse, bir noktanın koordinatları gibi, koordinatların her ikisiyle tek bir dal oluşturmak daha verimlidir. TBranch üzerindeki bir değişken yaprak (leaf) olarak adlandırılır. Bir dal nesne girişi, basit bir değişken listesi, dosya içeriği, TList içeriği ya da nesne dizisi olabilir. Bütün bu dallar için farkı **`TTree::`**`Branch` metodları vardır.

İlk örnekteki (`staff.root`) veri gibi basit bir değişken listesi, tamsayı yada float gibi, kaydedilmek istenirse aşağıdaki **`TTree::`**`Branch()` kullanılır.

```
tree->Branch("Ev_Branch",&event,"temp/F:ntrack/I:nseg:nvtex:flag/i");
```

İlk parametre dalın ismi, ikinci parametre ilk değişkenin okunduğu adrestir. Yukarıdaki koddaki "event" bir float, bir tasmayı ve üç işaretlenmemiş tamsayılı bir yapıdır. Yaprak

ismi yapının dışındaki değişkenleri seçmek için kullanılmaz, ama yalnızca yaprak için isim kullanılır. Bu üçüncü parametrede tanımlanmış sıradaki bir yapıda olması gerektiği anlamına gelir. Üçüncü parametre yaprak listesini tanımlaya dizidir. Her bir yaprak bir isme ve "/" ile ayrılmış bir türe sahiptir. Her bir yaprak sonraki yapraktan "**:**" ile ayrılır.

<Değişken>/<Tür>:<Değişken>:<Değişken>/<Tür>

Belirlenen her bir tür için farklı semboller kullanılır:

- C : 0 karakter tarafından sonlandırılmış karakter dizisi
- B : 8 bit işaretlenmiş tamsayı
- b : 8 bit işaretlenmemiş tamsayı
- S : 16 bit işaretlenmiş tamsayı
- s : 16 bit işaretlenmemiş tamsayı
- I : 32 bit işaretlenmiş tamsayı
- i : 32 bit işaretlenmemiş tamsayı
- F : 32 bit floating noktası
- D : 64 bit floating noktası

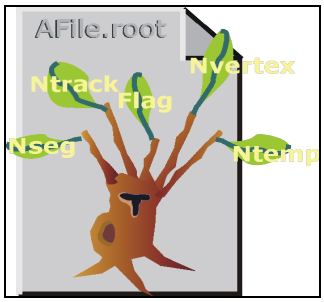
Tür ne kadar çok boşluk bölüneceğine karar vermekte byte hesabı için kullanılır. Varsayım olarak, değişken tür tanımlayıcı sembolünde belirtilmiş byte sayısı ile kopyalanır. Bununla birlikte, tür iki karakterini içerirse, sayı çıktı tamponuna değişken kopyalandığında kullanılmış olan byte numarasını belirtir. Aşağıdaki satırda tanımlanmış `ntrack` 32 bit tamsayı yerine 16 bit tamsayı olarak yazılmış olur.

"ntrack/I2"

17.3 Yarılma Seviyesini Ayarlamak

Bir dalı yarmak nesnedeki her veri üyesi için bir alt-dal oluşturmak anlamına gelir. Ayrılma seviyesi yarılmayı etkisiz kılmak için 0'a ayarlanabilir ya da yarılma derinliğini gösteren 1 ile 99 arasındaki bir sayıya ayarlanabilir. Eğer yarılma seviyesi 0'a ayarlanırsa, bütün nesneler bir dala yazılır. Yarılma seviyesi 1 olduğunda nesnenin veri üyesi bir dal tayin edilir, yarılma 2 olduğunda dal sayısı 2. Eğer yarılma seviyesi 99 olarak ayarlanırsa nesne maksimum seviyede yarılmaya uğrar.

Yarılma yok



Yarılma var

Bir dal yarılarak hemen birkaç dal üretebilir. Her bir dal hafızada kendine özgü bir tampona sahiptir. Eğer 50'den daha az dal varsa, tampon büyüklüğü 32000 bytedir, 100'den az dal varsa 16000 civarında ve 500'den daha fazla dal varsa 4000 byte tampon büyüklüğü oluşur. Bu sayılar 32MB'dan 256MB'a sıralanan hafıza büyüklüklü bilgisayarlar için tayin edilir. Eğer hafıza fazla ise, daha geniş tampon büyüklükleri belirlenir.

Dal yarılması okunma için hızlıdır ama yazma için biraz daha yavaştır. Okumak hızlıdır çünkü aynı türdeki değişkenler ardışık olarak saklanır ve türün her defa okunmuş olması gerekmez. Tampon numarasının daha büyük olması nedeniyle yazmak daha yavaştır.

17.3.1 Yarılma Kuralları

Bir dal yarılığında, farklı türlerdeki değişkenler başka şekilde ele alınır. Buna göre bir dal ayrıldığında uygulanan kurallar aşağıdaki gibidir:

- Eğer veri üyesi temel tür ise, **`TBranchElement`** sınıfının bir dalı olur.

- Veri üyesi temel türlerin düzeninde olabilir. Bu durumda, düzen için bir tek dal oluşturulur.
- Veri üyesi temel türlerin düzeni için bir işaretçi olabilir. Uzunluk değişebilir ve sınıf tanımındaki veri üyesinin yorum alanında belirtilmeli.
- İşaretçi veri üyesi, **`TClonesArray`** dışında, yarılmaz. **`TClonesArray,`** yarılma ikiden daha büyük yarılma seviyesindeyse, yarılır. Yarılma seviyesi 1 olduğunda, **`TClonesArray`** yarılmaz.
- Veri üyesi bir nesne işaretçisi ise, özel bir dal oluşturulur. Dal, dal tamponuna nesneyi ayırmak için Streamer fonksiyon sınıfı tarafından doldurulur.
- Veri üyesi bir nesneyse, bu nesnenin veri üyeleri yarılma seviyesine göre dallara yarılır. (yarılma seviyesi >2)
- Temel sınıflar nesne yarıldığında yarılırlar.
- Kuramsal temel sınıflar hiç yarılmazlar.
- STL konteynırları bazı uç durumlar dışında desteklenirler, aşağıdaki örnekler desteklenmezler:

```
//STL vector of vectors of TAxis*
vector<vector<TAxis*>> fVectAxis;

//STL map of string/vector
map<string,vector<int>> fMapString;

//STL deque of pair
deque<pair<float,float>> fDequePair;
```

- C-yapısındaki veri üyelerinin yarılma modu desteklenmez
- Yarılmayan bir nesne inceleme için yavaş olabilir
- Yarılmayan STL konteynerlar incelemede ulaşılabilir değildirler.

Nesne ile bir oluşturulursa, genellikle tüm veri üyeleri yarılmış olur. Ama yarılmadan bir veri üyesi muaf tutulabilir. Bu veri üyesinin yorum alanında belirtilir.

```
class Event : public TObject {
private:
     EventHeader     fEvtHdr;   //sayfa başlığı bölünmez
```

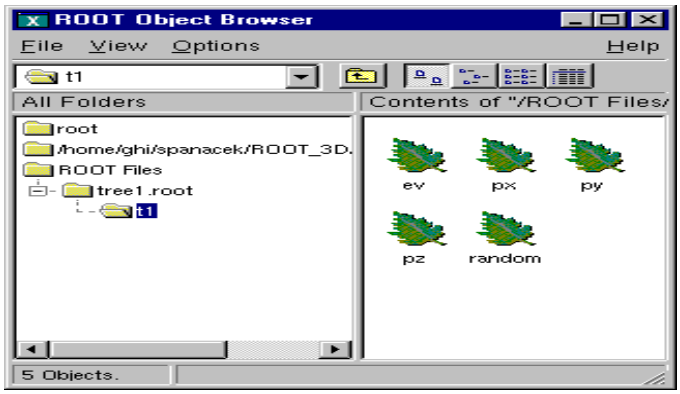
17.4 ÖRNEK 1 : Basit Değişkenli Bir Ağaç Oluşturmak

Bu örnek birkaç basit değişkenle (tamsayı ve floating noktası) bir ağacın nasıl yazıldığını, okunduğu ve gözlemlendiğini gösterir. Aşağıdaki satırlar ağaç (`tree1w`) yazan fonksiyonlardır. İlk olarak değişkenler tanımlanır (`px, py, pz, random` ve `ev`), daha sonra her değişken için çağırılan **`TTree::`**`Branch` metodu ile ağaca değişkenlerin her biri için bir dal eklenir.

```
Void tree1w()
{
   //create a Tree file tree.root
   //create the file, the Tree and a few branches
   TFile f("tree1.root","RECREATE");
   TTree t1("t1","a simple Tree with simple variables");
   Float_t px, py, pz;
   Double_t random;
   Int_t ev;
   t1.Branch("px",&px,"px/F");
   t1.Branch("py",&py,"py/F");
   t1.Branch("pz",&pz,"pz/F");
   t1.Branch("ev",&ev,"ev/I");
//fill the tree
   for (Int_t i=0; i<10000; i++) {
     gRandom->Rannor(px,py);
     pz = px*px + py*py;
     random = gRandom->Rndm();
     ev = i;
     t1.Fill();
   }

   //Save the Tree header; the file will be automatically closed
   //When going out of the function scope
   t1.Write();
}
```

Oluşturulan ağacı ve dalları görüntülemek için ROOT nesne penceresinde ROOT dosyalarına bakılır. Şekil 3'de oluşturulan ağacın nesne penceresi gösterilmektedir.



Şekil 3: ROOT nesne penceresi. Oluşturulan ve diske kaydedilen dosalar kendi alt dosyaları ile bu dizin altındadır.

Pencere açıldıktan sonra panelin sağ tarafında px, py, pz, ev ve random dalları görülür. Bunlar birer yaprak şeklindedir çünkü oluşturulan dallar sadece bir yapraktan oluşmaktadır. Penceredeki yapraklar üzerine iki kez tıklandığı zaman seçilen yaprağa ait histogram görüntülenir.

18. HADRONİK İLERİ KALORİMETRE (HF)

CMS (Compact Muon Selenoid) değişik detektör sistemlerinin soğansı bir yapıda birleştirilmesinden oluşmuş bir detektör sistemidir. Hadronik ileri kalorimetrede CMS'in bir alt detektörüdür. HF görülmemiş parçacık akılarını denemek üzere tasarlanmıştır. Detektörün geri kalanı için yalnızca 100 GeV'e kıyasla, bir proton-proton etkileşmesi başına 760 GeV iki ileri kalorimetreye bırakılır. HF ileri jetlerin elde edilmesini ve dikine kayıp enerjinin (E_T^{miss}) daha hassas bir şekilde ölçülmesini sağlamaktadır. Bu sayede başta Higss parçacığının keşfi olmak üzere birçok önemli fizik konusuna katkı sağlamaktadır. Kayıp dikine enerjinin ölçülmesi $H \to ZZ \to llv\nu$ ve $H \to WW \to lvjj$ kanallarında $m \approx 500\,\text{GeV}$ kütleli Standart Model (SM) Higgs araştırması için önemlidir(8).

Higgs bozonunun $H \to WW \to lvjj$ ve $H \to WW \to lljj$ kanalarında araştırılması yüksek ışıklılık (luminosity) gerektirir ($L \approx 10^{34}$ $cm^{-2}sn^{-1}$). Higgs bozonunun bu kütlelerde WW veya ZZ bozonlarının füzyonu ile üretilmesi ileri iki jetin varlığıyla karakterize edilir. Bu jetler oldukça yüksek enerjili olup momentumlarının dik bileşeni yaklaşık olarak W bozonunun kütlesi mertebesindedir. Ayrıca ileri jetlerin elde edilmesi Higgs'in 80-140 GeV kütlesi aralığında araştırılması için de oldukça kullanışlıdır.

İleri kalorimetreler $3 < \eta < 5$ aralığındaki sahte-sürati (pseudorapidity) kapsar. Eğer Cerenkov ışıması sonucu oluşan ışıma açısı θ olarak alınırsa ($\cos\theta = 1/n\beta$ burada ki n ortamın kırılma indisi, β ışığın kuvars içerisindeki hızı) sahte-sürat

$$\eta = -\ln\tan(\theta/2)$$

olarak ifade edilebilir.

10^{34} $cm^{-2}\,sn^{-1}$'lik bir ışıklılıkta LHC'nin işlemesiyle, ve toplam 100 mb'lik bir pp tesir kesiti varsayarak (demet geçişi başına 25 pp çarpışması), geçiş başına IP'deki ortalama parçacık çeşitliliği 5700 civarındadır (rms=1200). Bu, 280 parçacık/geçiş/sürat birimine karşılık, $2.3\text{x}10^{11}$ sn^{-1}'lik bir orana uyar. $4.5 < \eta < 5$ bölgesi $6.0\text{x}10^{6}$ $cm^{-2}\,sn^{-1}$ civarında bir akı ile $9.6\text{x}10^{9}$ Hz civarından bir parçacık oranı dener ve emilmiş doz

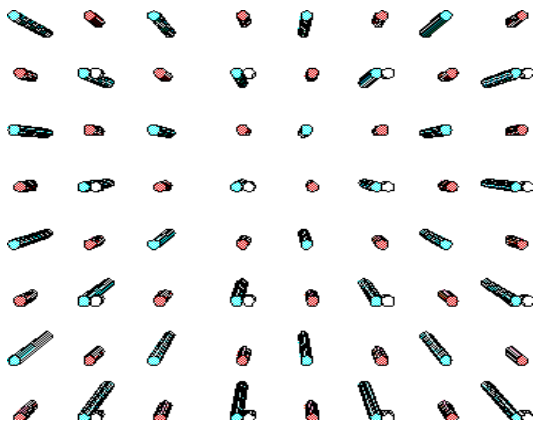
yaklaşık 100 Mrad/yıl'a ulaşır. Bu nedenle, detektör olağanüstü yüksek radyasyon alanında daha uzun ömürlü olmalıdır(9).

HF üzerindeki parçacık olayları kalorimetre soğurucusundaki büyük nötrino akışına ve soğurucu malzemenin aktivasyonuna öncülük eden duşlardan oluşur. Bu nedenle HF tarafından kullanılan algılama tekniği aktif hale getirilmiş radyoçekirdeklerin bozunmalarındaki nötrinolara ve düşük enerjili parçacıklara duyarlı değildir. Bu aktivasyonun bir sonucu olarak, HF'in içindeki parçalar (LHC çalışmaya başladıktan 60 gün sonra) 10 mSv/h civarında bir radyasyon kaynağı olur.

HF modüller yerleştirilmiş kuvars fiberli çelik bloklarla yapılır. Kuvarstaki ışığın hızından daha büyük bir hızla kuvars bir fiberden geçen yüklü parçacık Cerenkov etkisinden dolayı foton yayar. Fiberin bir ucuna yerleştirilmiş detektör tarafından gözlemlenen ışık miktarı parçacığın hızına, olay açısına ve parçacık yörüngesi ile fiber merkezi arasındaki mesafeye bağlıdır. Aynı zamanda fiberin çekirdeğine ve kaplamanın kırılma indisine, fiberin spektral iletim aralığına ve ışık detektörünün spektral kuantum verimliliğine de bağlıdır. HF'in kalibrasyonu üç bağımsız donanım sistemine dayanır: Lazer Gain Monitör, Işık yayan diyotlar (LED) ve hareketli radyoaktif kaynak. Bu sistemler %3'ü aşmadığı kesin olmayan ayrıntılı bir enerji ölçüsündeki sonuçlar için gerekli kalibrasyonu verir.

18.1 HF'in Yapısı ve Çalışma Prensipleri

HF demir soğurucu içerisine yerleştirilen kuvars liflerden oluşmaktadır. HF içerisinde farklı uzunlukta iki modül vardır. Modüllerden her biri etkileşim noktasından z=11.1m uzaklıktadır. Kullanılan lifler gelen proton-proton demetine paralel olacak şekilde yerleştirilir. Uzun olan fiber 1.65 m uzunluktadır ve kalorimetrenin elektromanyetik bölümünü oluşturur. Elektromanyetik kısım foton ve elektron gibi elektromanyetik etkileşen parçacıkların enerjilerini ölçmek için kullanılır. Kısa olan fiber ise 1.43 m uzunluğunda olup detektörün yüzeyinden 22 cm uzaklıkta olacak şekilde yerleştirilir ve hadronik bölümü oluşturur. Şekil 4'te detektörün kesiti gösterilmektedir.



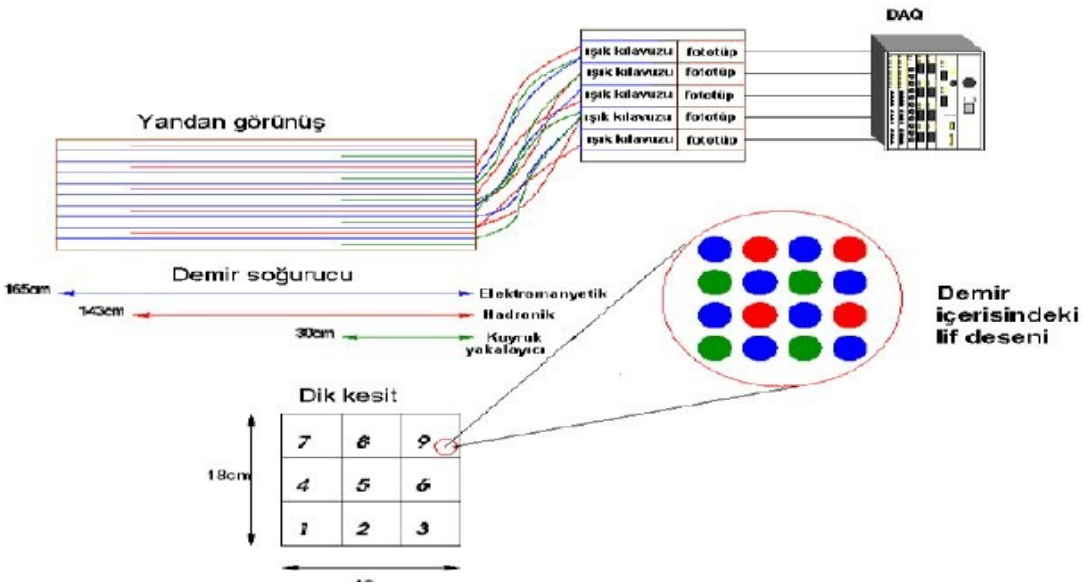
Şekil 4 : Üç farklı uzunluktaki fiber 2 mm kare geometride bakır soğuruculara yerleştirilir. EM fiberler soğurucunun giriş uzunluğundadır (1.65 m) ve şematik olarak TC (Tail Catcher) fiberlerden sonra gösterilirler.(TC fiberler kalorimetrenin arkasından soğurucuya 30 cm eklenir). HAD fiberler şekilde kısa olarak görünen fiberlerdir ve detektörün yüzeyinden 22 cm uzaklıktadır.

Hadronik bölüm elektromanyetik bölüm ile birlikte hadronların enerjilerini ölçmeye ve elektromanyetik etkileşen parçacıkları hadronlardan ayırmaya yarar(10). Şekil 5'te fiberlerin fototüplere montajı gösterilmektedir.

Şekil 5 : Kuvars fiberli HF takozların montajı

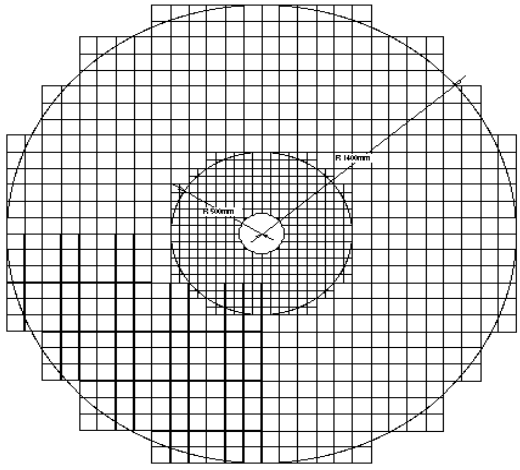
Farklı uzunluktaki fiberler farklı fototüplere takılırlar. Elektromanyetik (EM) fiberlerin bağlı olduğu fototüplerin oluşturduğu gruba QIE EM (QIE : Charge integrator and encoder), hadronik (HAD) fiberlerin bağlı olduğu fototüplerin oluşturduğu gruba QIE HAD adı verilir. Fototüpler ışığı bir elektrik sinyaline dönüştüren aygıtlardır, fotokatottan (fotoelektrik etki kullanılarak, fotonlar elektronlara çevrilirler), bir dizi

diyottan ve sonuçlanma akımını toplayan bir anottan oluşurlar. Şekil 6'da test amaçlı kullanılan bir düzeneğin prototipi görülmektedir.



Şekil 6 : Test edilmek üzere yapılmış prototipin şematik görünümü

HF detektörünün içine yerleştirilmiş kıvılcım saçan plakalara kule (tower) adı verilir. Kule yapısının kare geometride olması tercih edilir; $|\eta| > 4$ için, kuleler 5cm x 5cm ve $|\eta| < 4$ için, kuleler 10cm x 10cm şeklindedir. Bu kuleler HF'in ön cephesinde yer alırlar. Bu kuleler aşağıdaki şekilde HF detektörlere yerleştirilir.

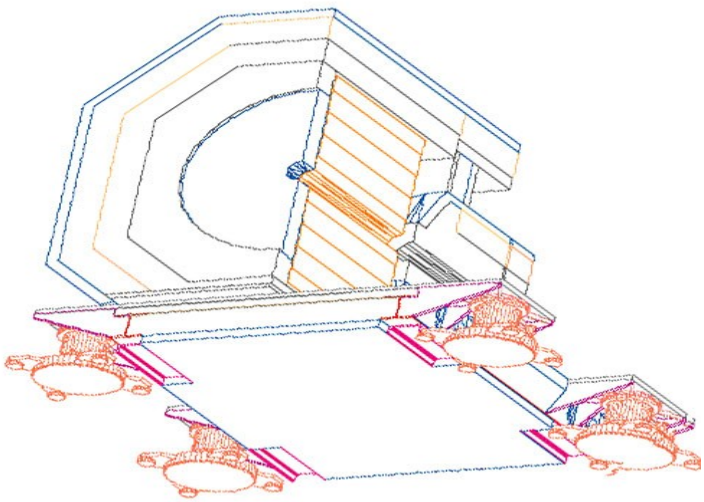


Şekil 7 : Tanımlandığı gibi kule yapıların HF'in ön cephesindeki yerleşimi gösteriliyor. Daha küçük olan (5cmx5cm) kuleler ışının iletim bölgesine daha yakındır ve iri kuleler $\eta<4$ bölgesine yerleştirilmiştir. İçerideki yarıçap 12.5 cm'dir

Oluşturulan her bir HF detektörünün içerisine toplam 18 tane soğurucu takoz (wedge) yerleştirilir. Bu takozların oturtulduğu taban plakaları fototüp kutuları ve zırhlardan oluşmuşlardır. Her bir takoz 20^0'lik bir yer kaplar. Lifler ışık kılavuzlarına gönderilecek şekilde buket halinde bu takozların içinden geçerler. Fototüpler doğrudan bu ışık kılavuzlarının üzerine oturtulmuştur.

Detektörün çalışma prensibi Cerenkov ışımasına dayanır. Gelen parçacıklar soğurucuyla etkileşerek ikincil parçacıklar oluştururlar. Oluşan bu parçacıklar yeterli enerjiye sahip oldukları sürece yeniden etkileşerek yeni parçacıklar oluşmasını sağlarlar. Bu mekanizmayla birçok parçacık oluşmasına duş (shower) denilir. Oluşan etkileşmenin türüne göre duş elektromanyetik veya hadronik duş olmak üzere iki şekilde gerçekleşir.

Genellikle her hadronik duşun birde elektromanyetik bileşeni vardır. Etkileşmeler oluşurken parçacıkların büyük bir kısmı kuvars liflerin içinden geçerler. HF içerisinde Cerenkov mekanizması kullanılır. Cerenkov mekanizmasıyla elde edilen duşlar diğer tekniklerle elde edilen duşlara göre oldukça dardır. Örnek olarak, dE/dx prensibiyle çalışan kalorimetrelere göre, duşun dikine genişliği en az 3 kat daha dardır. Ayrıca Cerenkov kalorimetrelerde hadronik duşların çoğunlukla elektromanyetik bileşenleri daha baskındır. Bu kalorimetrenin boyunun daha kısa olmasını sağlar. Yani Cerenkov kalorimetrelerde 8λ'lık bir uzunluk yeterli iken dE/dx kalorimetrelerde 12λ'lık bir uzunluk gerekmektedir. Bu özellikler, bu detektörlerin aynı işi daha küçük boyutlarda yapmasını sağlar. Şekil 8'de tamamlanmış bir HF detektörünün alttan kesiti gösterilmektedir.



Şekil 8 : HF detektörün alt yandan görüntüsü. Şekilde HF detektörünün bir kesiti ve alt koruyucu tamponun pozisyonu gösteriliyor. Alt tamponun hareketi dört köşesine yerleştirilmiş hava yastıkları tarafından sağlanmaktadır.

Cerenkov ışımasının diğer özellikleri ise yüksüz ve relativistik olmayan parçacıklara duyarsız olmasıdır. Bu sayede ortamda bol miktarda bulunan düşük enerjili (MeV) nötron fonuna ve soğurucunun nükleer aktivasyonu sonucu yaydığı birçok radyoaktif ürüne duyarsız kalır.

18.2 Data Analizleri ve Sonuçları

18.2.1 Enerji Ayarı

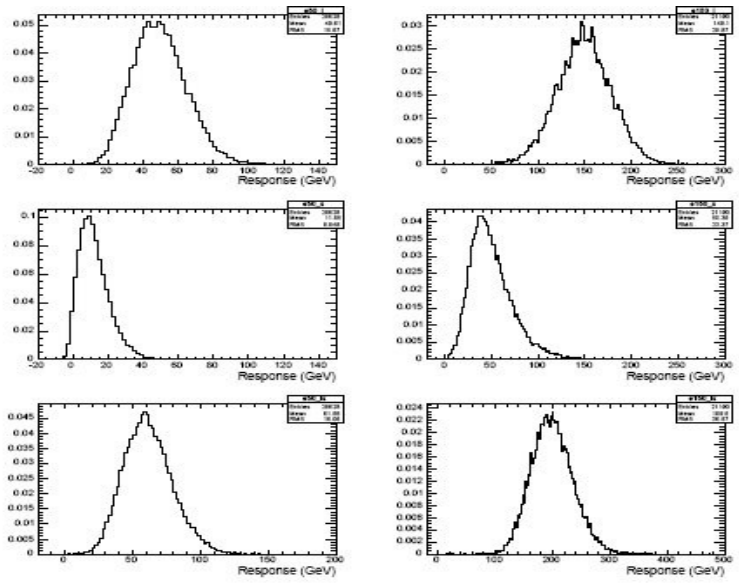
HF takozlarının enerji ayarı uzun (L) ve kısa (S) fiber kısımlarının her ikisi için 100 GeV'luk elektronlar kullanılarak tamamlanır. Altı takoz dikey ve yatay bir şekilde 2.5 cm'lik adımlarla taranır. Takozların yanından enine sızıntıya duyarlı olmamak için, parçacıkların çarpışma noktasının, analizin durduğu esnada, kenarlardan en az 2 cm içeride olması sağlanılır. Tüm i kulesine çarpan elektronlar için, j kulesinde ortalama sinyal olarak A_{ij} hesaplanır. L kısım i kulesi için, 100 GeV ışın enerjisine eşit elektronlar için tam takoz sinyali gereklidir(11).

$$A_{ij} w_j = 100GeV$$

Burada w_j j kulesi için bilinmeyen kalibrasyon katsayısıdır. Her bir takozda, 24 L ve 24 S kanalı vardır(i = 1,2,….24). Bu 24 lineer denklem çözülerek, tüm L türündeki kuleler için kalibrasyon sabitleri belirlenir. S kulelerinin kalibrasyonunda, toplam S sinyali 30 GeV'a ayarlanılarak, aynı yöntem kullanılır.

18.2.2 Enerji Çözünürlüğü

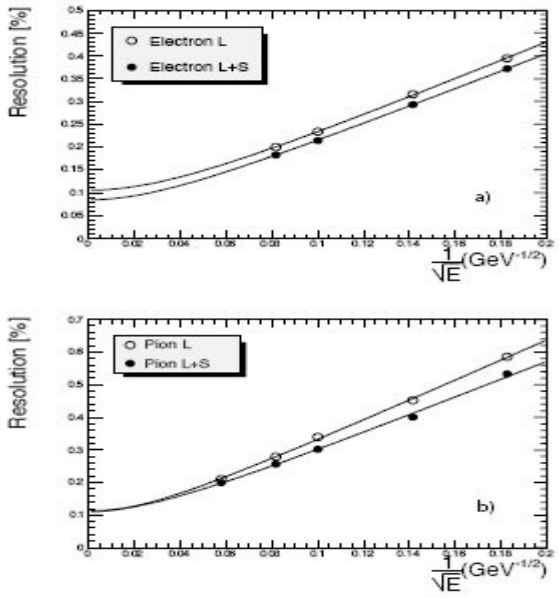
50 GeV ve 150 GeV'lik elektronlar için L, S ve toplam L+S tepki fonksiyonları aşağıdaki Şekil 9'da gösteriliyor. Detektör, tüm uzun fiber kulelerinden toplanan elektron sinyalini ışın enerjisine eşitleyecek şekilde ayarlanır. 50 GeV'lik elektronlar için kısa kısımdaki ortam sinyali ışın enerjisinin yaklaşık %24, yani 11.95 GeV, kadardır. Benzer şekilde, uzun fiber kısımları 150 GeV'lik elektronlar için 149.1 GeV verir.



Şekil 9 : Uzun (en üst), kısa (orta) ve birleşmiş L+S (en alt) sinyallerdeki 50 GeV'lik (sol taraf) ve 150 GeV'lik (sağ taraf) elektronlar için tepki fonksiyonları yukarıda gösteriliyor. Tüm dağılımlar integralleri 1'e eşit olacak şekilde normalize edilmiştir.

HF kalorimetrelerin elektromanyetik enerji çözünürlüğü fotoelektron istatistiği ile kontrol edilir. 50 GeV'de, elektromanyetik enerji çözünürlüğünün ~%29 'u tahmini

kısımdan dolayıdır. Sabit kısım (~%10) yapısal benzersizlikler tarafından idare edilir. Fiberden fibere mesafe Moliére yarıçapının önemli bir parçası gösterildiği sürece, kalorimetre tepkisi parçacıkların çarpma noktasına bağlı olur. İki fiber arasındaki yarı yol çarpma noktası için daha küçüktür ve çarpma noktası fibere yaklaşıyormuş gibi artar. Şekil 10'da enerji çözünürlüğünün fiberler üzerindeki sonuçları gösterilmektedir.



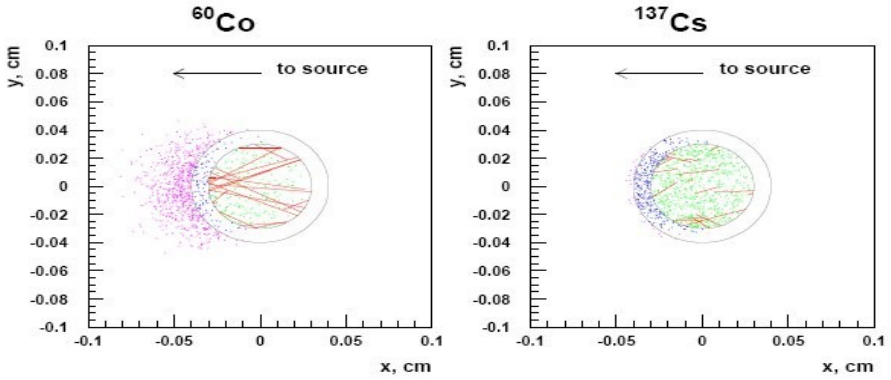
Şekil 10 : Elektromanyetik enerji çözünürlüğü (a) fotoelektron istatistiği tarafından idare edilir ve böylece $a/\sqrt{E} \oplus b$ olarak parametre edilir. Enerji çözünürlüğü, tahmini kısımlar için %28 ve sabit kısımlar (b) için %11 dir. Uzun fiberin sonuçları kullanılır.

18.2.3 Radyasyon Dayanıklılığı

Dedektörlerin çalıştığı ileri bölgeler ($\eta > 3$ ve $\theta < 5.7$) radyasyon bölgelerinin oldukça yüksek olduğu bölgelerdir. Bir p-p çarpışmasında iki HF modülünde depolanacak enerji miktarı 760 GeV'dir. Bu bölgelerde ortalama enerjileri 14 MeV olan nötron oranları 10^8 Hz/cm^2'dir. Yüklü hadronların oranı da duşun maksimum olduğu yerlerde 10^{13} Hz/ cm^2'den 10^{16} Hz/cm^2'lere kadar değişmektedir. LHC'nin 10 yıl çalışması durumunda $\eta = 5$ bölgesinde beklenen doz 10 Grad'dır. Bu nedenle bu bölgelerde çalışacak dedektörlerde kullanılacak olan malzemeler radyasyona dayanıklı malzemeler olmalıdır. HF'te bu yüzden aktif eleman olarak radyasyona en dayanıklı maddelerden biri olarak bilinen kuvarstan yapılmış lifler kullanılmaktadır.

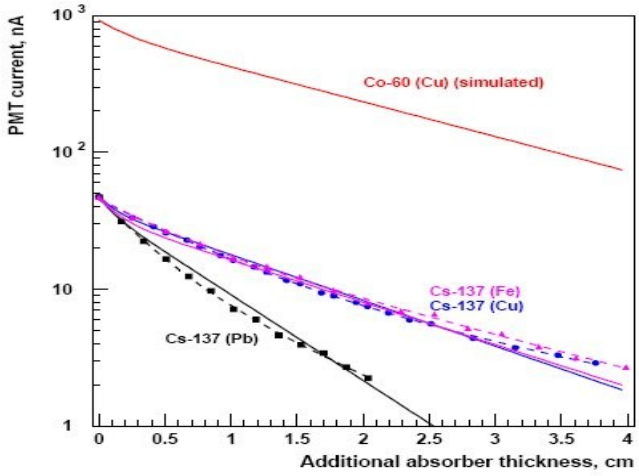
Liflerin radyasyona dayanıklı olmaları HF için çok önemlidir. Doz arttıkça liflerinin iletiminin nasıl değiştiği oldukça önemlidir. Bu nedenle, deney süresince fiberlerin 500 MeV'lik elektron demeti ile karşılaşmaları göz önünde tutularak ışık iletimlerinin artan elektron sayısı ile nasıl değiştiği kontrol edilmelidir. Yapılan çalışmalarda foto tüplerin kuantum yeterlilikleri en yüksek oldukları bölgelerde fiberin ışık iletimin zayıflamasının diğer bölgelere nazaran daha az olduğu görülmüştür. 100 Mrad'luk doz için zayıflama 1.51 ± 0.15 dB/m olarak ölçülmüştür(1).

Radyoaktif kaynak kullanımı daha yüksek enerjili kalorimetreler için temel enerji kalibrasyonu, izleme ve teşhisler için test edilmektedir. Deneysel parçacık fiziğindeki LHC'de (Large Hadron Calorimeter) CMS deneylerindeki HF'lerde aktif parça olarak silis-çekirdekli ve polimer kaplamalı fiberler kullanılır. İleri kalorimetreler yüksek dozlu radyasyondan (~100 MRad/yıl) korunmaya ihtiyaç duyduğundan beri bu tercih genelde kuvars fiberlerin radyasyon direncine dayanır. Böyle bir kalorimetrede, Cerenkov eşiğinin (elektronlar için E>178 keV) üzerindeki yüklü duş parçacıkları Cerenkov ışığı üretirken sinyal üretilir, aslında kalorimetreler yapıldığında genellikle duşun elektromanyetik bileşenine duyarlıdır(12). Aktif fotonlar tarafından Compton elektronlarının üretildiği boşluk noktalarının benzeri olan noktalardaki farklı gama spektrumları nedeniyle kobalt (Co) ve sezyum (Cs) kaynakları arasındaki farklar şekil 11'de gösterilmektedir. Katı çizgiler Cerenkov fotonlarının üretildiği izleri gösterir.



Şekil 11 : Co^{60} ve Cs^{137} kaynakları için sinyal üretim simülasyonunun grafiksel örneği. Noktalar sıfırdan farklı sinyal üretimi için Compton elektronunun nakavt pozisyonlarının boşluk dağılımını gösterir. Katı çizgiler Cerenkov ışığının üretildiği izleri işaret eder.

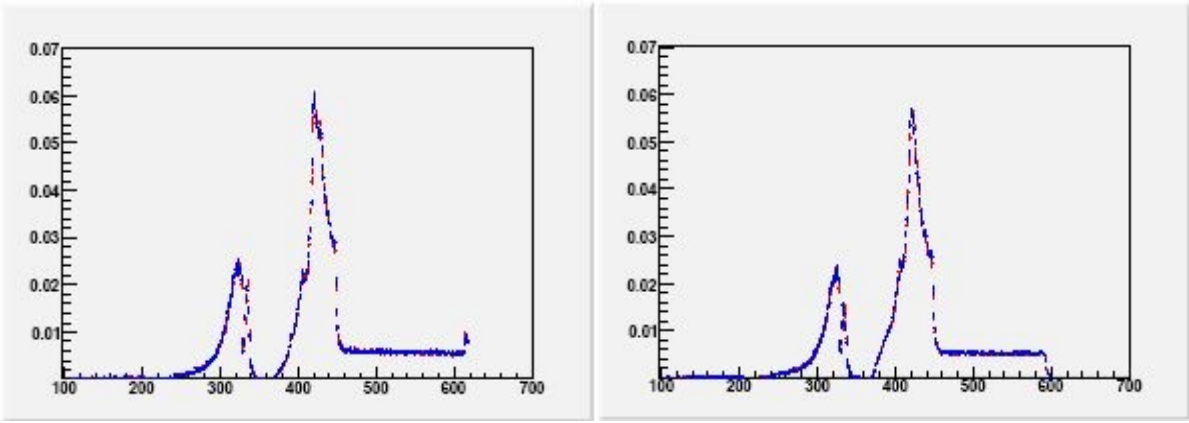
Bozunmaların %94.4'nü 661 keV'lik gamalar oluşturur. Bozunmanın geri kalan %5.6'sı 524 keV'lik elektronlardan elde edilen sonuçlardır. Co^{60} sonuçları sadece simülasyonla elde edilmiştir. Kaynak kalibrasyonu için Cs^{137} kullanılmamasının nedeni Cerenkov ışımasının zayıf olmasıdır. Ölçümlerin ve simülasyonların sonuçları şekil 12'de gösterilmektedir. Tüm madde ve tabakalar için simülasyon ve veriler arasında uyum vardır(13).



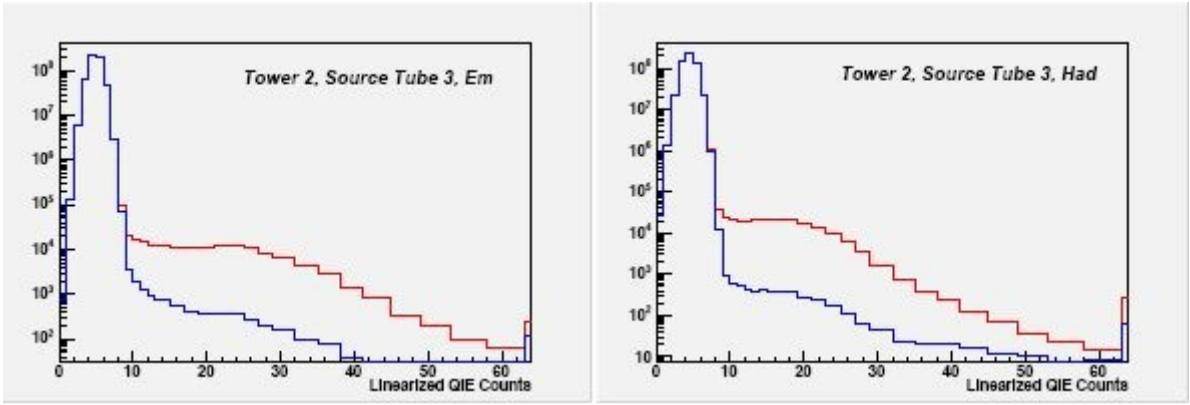
Şekil 12 : Noktalar ölçümleri, katı çizgiler ise Monte Carlo simülasyonunu gösterir. Kesik çizgiler veri noktalarına uygun olan parametrelerdir. Veri noktaları demir, kurşun ve pirinç soğurucularla 21 mCi Cs^{137} ile alınır.

19. HF KALİBRASYON METOTLARI

PMT'den gelen sinyaller QIE rakamlayıcısı tarafından front end elektroniği ile rakamlanılır. QIE lineer olmayan dört farklı aralıkta yük ölçebilir. Her bir aralık 32 kutu (bin) içerir bunlardan 15 kutuluk bolümdeki yükler 1 ile ağırlıklı(Weighted) ölçülür, 7 kutuluk 2 ile, 4 kutulu 3 ile, 3 kutulu ise 5 ile ağırlıklı ölçülür. Rakamsal yükün değeri dört aralığının her biri için farklıdır. Kaynak kalibrasyonu için yalnızca ilk QIE aralığı 2.6 fC'lik rakamsal yükle kullanılır. Veri her 25 ns'lik zaman diliminde rakamlanılır sonra 20 Hz'lik frekansla readout ve histogramlarda otomatik olarak işlenir. Bu kaynak kalibrasyon testinde 5 mCi Co^{60} kaynağı kullanılmaktadır. Aşağıda Şekil 13 ve 14'de kaynak kalibrasyonunda alınan örnek histogram sonuçları gösterilmektedir.



Şekil 13 : L ve S hücreleri için kaynağın pozisyonun kalorimetrenin tepkisi ile olan ilişkisi



Şekil 14 : L ve S hücreleri için tipik QIE dağılımı; kırmızı (üstteki) çizgi - kaynak soğurucunun içinde; mavi (alttaki) çizgi – kaynak takozun dışında.

Kaynak ilk olarak takozun dışında korunmuş bir garaja yerleştirilir. Sonra, özel bir kaynak sürücüsü kullanılarak, takozun arka tarafından kaynak tüpe yerleştirilir(ışın yönüne ters). Şekil 13 kaynaktan garaja uzaklığın bir fonksiyonu olarak bazı QIE eşiği üzerinden hesaplanmış yükü gösterir (sol ve sağ kısımlar uygun bir şekilde L ve S readoutlardan toplanan sinyali gösterir). Kaynak fiber yığınlara ve PMT'ye yakın bölge boyunca geçtiğinde dağılımda iki tepe (peak) görünür. Tepelerin sağ tarafındaki sabit tepki takozun içindeki hareket eden kaynaktan elde edilir. L readout için, kaynak takozdan ayrıldığında sinyal süratle artar, ve sonra keskin bir şekilde aşağı düşer. Kaynaktan dolayı olan bu artış, takozun dışından gelen L fiberlerin uçlarında aydınlanma başlatır. S readout için sinyalin kaybolduğu nokta L fiberlerin readout'una göre 20cm yer değiştirir. Bu L ve S fiberler arasındaki uzunluk farktan ortaya çıkar.

Şekil 14 doğrusallaşan QIE sayımlarının süresinde L ve S readout hücreleri için tipik QIE dağılımını gösterir. Kırmızı (üsteki) çizgi kaynağın soğurucunun içinde olduğu duruma uyar, mavi (alttaki) çizgi kaynağın takozun dışında olduğu durum içindir. Her iki dağılım aynı zaman müddeti için elde edilir ve bu sebeple girişlerdeki aynı sayıya normalize edilir. Kaynak koordinatlarının farklı aralıkları, farklı hücreler için takoz içindeki kaynak için dağılımları çizilir. Her bir hücre için kaynak koordinatlardaki aralık Şekil 13'de gösterilene benzer ayrı ayrı bölgeler kullanılarak belirlenir.

Kaynak verileri toplanırken PMT ler iki farklı voltaj da (-1350 V ve -1150 V) çalıştırılmıştır. Kaynak ve test ışını verisinin birleştirilen analizi kaynakla yapılan kalibrasyonun mutlak normalizasyonunu yapmak için kullanılır. -1150 V için 2004 yılında alınmış hiçbir test ışını datası bulunmaktadır(sadece wedge 2.13 için kaynak verisi bulunmaktadır) dolayısıyla 2006 kaynak testi verileri ancak -1350 V ile alınan datalar ile karşılaştırılabilinir. Kaynak verilerini kullanarak HF kalibrasyonu için birkaç farklı metodu göz önünde tutulmuştur. Bu metotların tek farkı yüklerin verilerden nasıl elde edilmesi ile ilgilidir.

19.1 "Mean Charge" Metodu

Bu metotta, yük(her bir saniyedeki için)

$$Q_i^{(\sin yal/background)}(QIE/s) = \frac{40MHz}{\sum_{j=1}^{32} n_j} \cdot \sum_{j=1}^{32} q_j \times n_j \qquad (1)$$

formülü kullanılarak sinyal ve background histogramları için ayrı ayrı hesaplanılır. Burada i kaynak tüp numarası, q_j QIE yükünün değeri ve n_j histogramın j'inci aralığındaki girişlerin sayısıdır. Ortalama yük değerleri sinyal ve background için hesaplanmış ortalama yük değerleri birbirinden çıkartılarak hesaplanır:

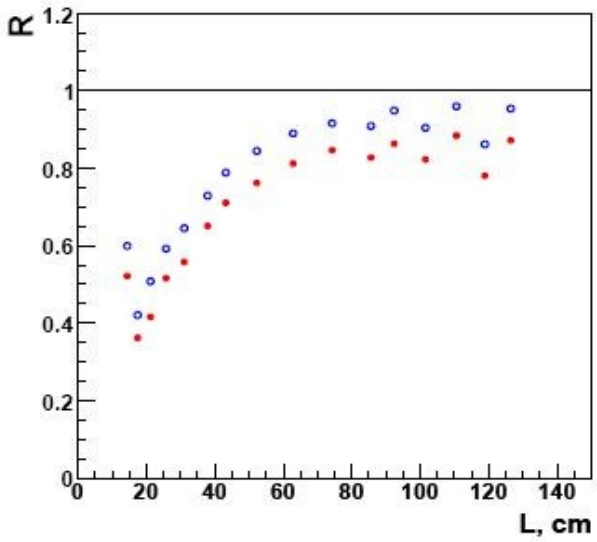
$$Q_i(QIE/s) = Q_i^{\sin yal} - Q_i^{background} \qquad (2)$$

Q_i yükü

$$E(GeV/s) = Q_i(QIE/s).\frac{g_i(GeV/QIE)}{R_i} \qquad (3)$$

yoluyla kaynak tarafından bırakılan enerji ile ilgilidir, burada $E(GeV/s)$ kaynaktan bırakılan enerji, $g_i(GeV/QIE)$ kalibrasyon katsayısı ve R_i geometriksel ölçü faktörüdür.

MC ile hesaplanan R_i geometriksel ölçü faktörü uzunluğa bağlı olarak Şekil 15 de gösterilmiştir. Sinyalin parçası sonsuz büyüklükteki hücreye nispeten i'inci hücredeki fiber tarafından toplanır. R_i kaynak tüpten hücre sınırlarına olan mesafeye ve readout fiberlere göre kaynak tüpün pozisyonuna bağlıdır.



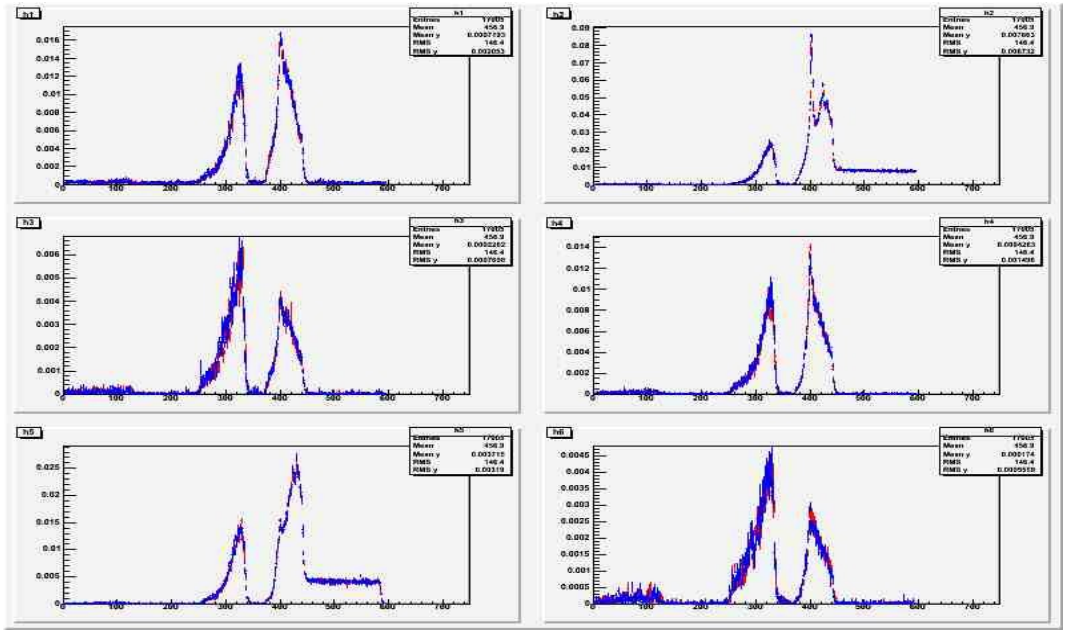
Şekil 15 : Işın merkezinden itibaren uzaklığın bir fonksiyonu olarak geometriksel ölçü faktörünün MC hesaplaması. İçi boş semboller kaynak tüp S oluğundayken L yığınındaki, kaynak tüp L oluğundayken S yığınındaki sinyale uyar. İçi dolu semboller kaynak tüp fiberle oluk aynı türden olduğundaki sinyale uyar.

g_i eşitlik(3)'den

$$g_i(GeV/QIE) = \frac{U(mCi)\varepsilon(GeV)R_i}{Q_i(QIE/s)} \qquad (4)$$

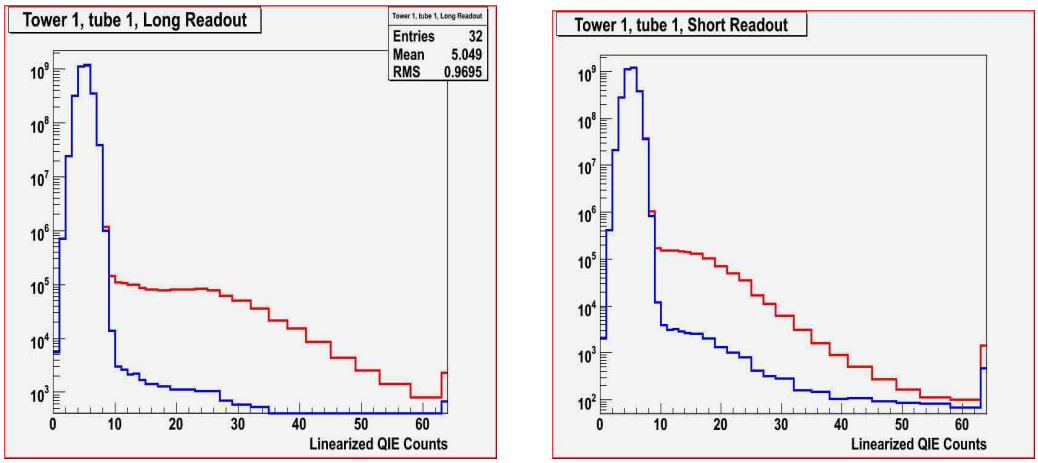
olarak ifade edilebilinir, burada $U(mCi)\varepsilon(GeV)$, E(GeV)'in yerine değiştirildi, $U(mCi)$ kaynağın aktivitesi ve $\varepsilon(GeV)$ 1mCi'lik parçalanma sayısı için sonsuz büyüklükteki kule tarafından toplanacak ortalama sinyaldir.

Her bir kalorimetre hücresi için tek kalibrasyon katsayısı C_i^{kaynak} türetilir. Eğer hücre yalnızca bir kaynak tüp içerirse, uygun g değerine eşittir ve iki kaynak tüp varsa, bu tüplerin ortalama g değerine eşittir. C_i'yi hesaplamak için, yalnızca eşitlik(4)'deki bilinmeyen ε parametresi belirlenir. Bilinmeyen ε parametresi kaynak ve ışın verileri için kullanılan takozlara göre belirlenir. Şekil 16 da "Mean Charge" metot'u ile elde edilen sonuçlar gösterilmektedir.



Şekil 16 : "Mean Charge" metoduna göre takoz 1.03 tower 1 için L ve S hücreleri için kaynağın pozisyonun bir fonksiyonu olarak kalorimetrenin tepkisi.

"Mean Charge" metodu ile takoz 1.03 (1.03 kısaltması 1. kısımdaki 3 takoz anlamına gelmektedir) için elde edilen kalibrasyon katsayıları: S readout için 0.195809 ve L readout için 0.335099 olur. Bu metodun keskinliği esas denge tarafından sınırlanır. Çünkü sinyal ve background histogramlar çoğunlukla, temel pozisyondaki küçük dalgalanmanın hesaplanan ortalama yükü kuvvetli bir şekilde etkileyebildiği temel peak girişlerden meydana gelir. S hücresi için kalibrasyon keskinliği L hücresinden azdır.



Şekil 17 : Takoz 1.03 tower 1 için EM (soldaki) ve HAD (sağdaki) QIE dağılımları

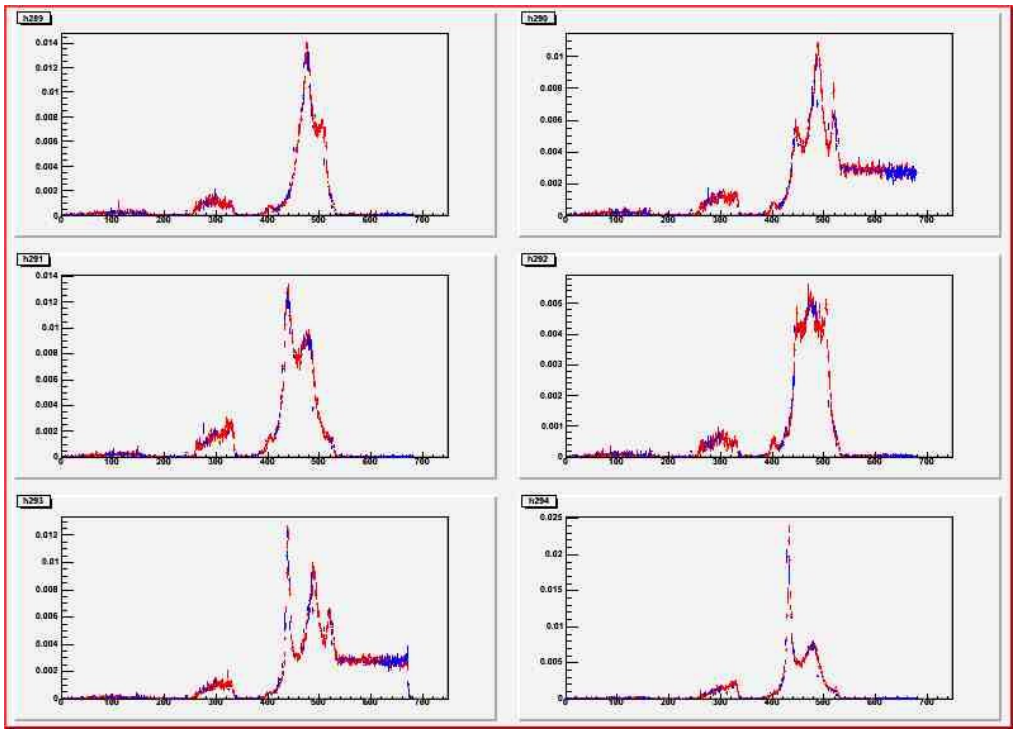
19.2 "Fixed QIE Bin Range" Metodu

"Mean Charge" metodunun keskinliği açık bir şekilde temel dengeye bağlıdır. Bu bağlılıktan kurtulmak için tüm hücreleri kapsayan sabit birkaç QIE kutu aralığındaki yük hesaplanmalıdır. Belirlenen bu kutu aralığı 10-30 aralığı olarak alınır(14).

Yük aşağıdaki eşitlik kullanılarak sinyal ve background histogramları için ayrı ayrı hesaplanılır:

$$Q_i^{(\sin yal/background)}(QIE/s) = \frac{40MHz}{\sum_{j=1}^{32} n_j} \cdot \sum_{j=10}^{30} q_j \times n_j \qquad (6)$$

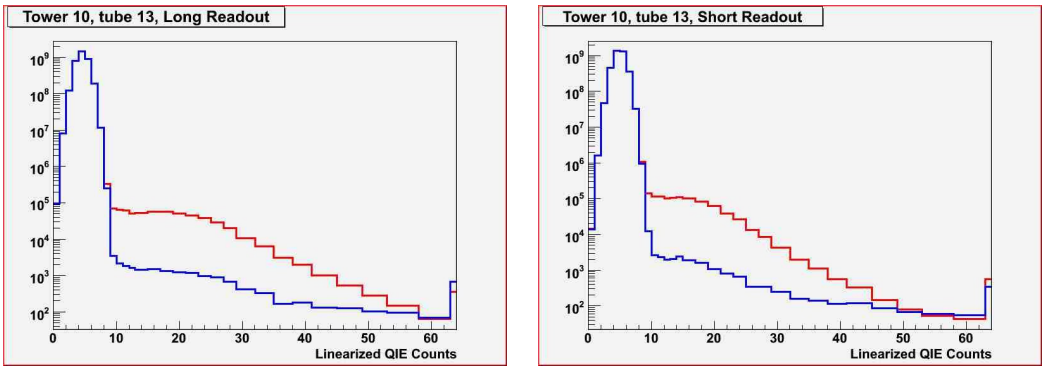
Burada i kaynak tüp numarası, q_j QIE yükünün değeri ve n_j histogramın j'inci aralığındaki girişlerin sayısıdır.



Şekil 18 : "Fixed QIE Bin Range" metoduna göre takoz 2.03 tower 10 için L ve S hücreleri için kaynağın pozisyonun bir fonksiyonu olarak kalorimetre tepkisi.

Sinyal ve background histogramlar için yükün hesaplanmasından sonra, "Mean Charge" metodunda tanımlanan kalibrasyon prosedürü eşitlik(2)'den yola çıkılarak kullanılır. Bu metoda göre takoz 2.03 için elde edilen kalibrasyon katsayıları: S readout için 0.339703 ve L readout için 0.364571 bulunur.

SPE'deki (Single Photoelektron) sinyalin olası paylaşımından dolayı SPE'deki sinyal ardışık iki zaman dilimi arasında maksimum değerden sıfıra dağıtılır. Fix bin range'de hesaplanan yük toplam yükün bir parçasıdır ve toplam yüke oranı bazı dış parametrelere, örnek olarak PMT kazançları, bağlı olur. Bununla beraber, kalibrasyon katsayısı toplam yüke ters orantılı olmalı, eşitlik (4). Bu metodun keskinliğine sınırlama koyar.

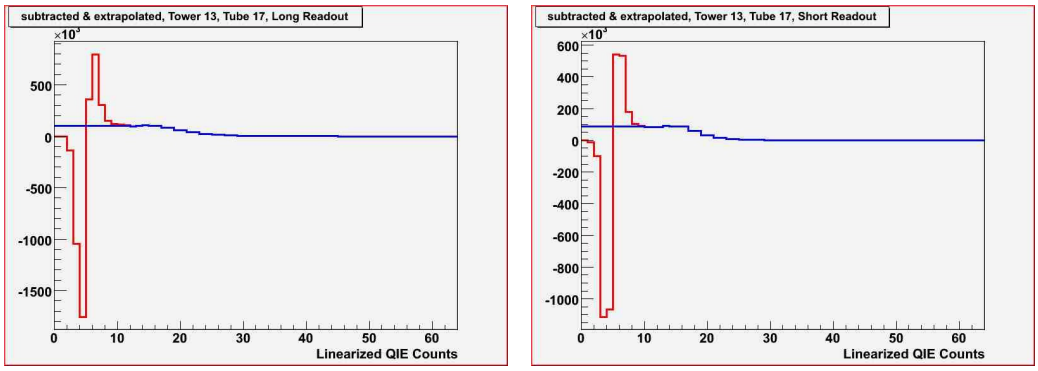


Şekil 19 : Takoz 2.03 tower 10 için EM (soldaki) ve HAD (sağdaki) QIE dağılımları

19.3 "Signal Extrapolation Under the Pedestal Peak" Metodu

Bu metodun ilk adımında, sinyal ve background histogramlar çıkarılır. Histogramın şeklinin esasen temel girişlerin çıkarılmasıyla belirlendiği küçük QIE değerlerinin durumunda değilken, sonuçlanan histogram (Şekil 20 kırmızı çizgi) büyük QIE değerlerinin bölgesinde saf kaynağın aktivitesini gösterir. Küçük QIE değerlerinin bölgesindeki büyük negatif tepe normalizasyonun sonucudur: Sinyal histogramı, backgroundun birinin önemli bir şekilde yüksek QIE bölgesinde girişlerin fazlalığını karşılamak için temel bölgedeki girişlerin daha küçük numarasını içerdiği gibi, girişlerin aynı numaralarına normalize olur. Temel peakin altındaki sinyal parçasını çıkarmak için, bir dışdeğerbiçimi uygulanmalı. Dışdeğerbiçimi birkaç derecede keyfidir çünkü temel peakin altındaki sinyalin şekli bilinemez.

Değeri çıkarılan dağılımda sabit sinyalli uygun bölgeden tayin edilir. Bu bölge her bir kalorimetre hücresi için ayrı ayrı ayarlanır. Şekil 20'de gösterilen alan için x ekseninden 12-16 arasındaki bölge seçilmiştir. Sabite yakın dışdeğerbiçimi temel değerin uygun bölgesinden kullanılır, aşağıda temel sinyal sıfıra ayarlanır (Şekil 20 mavi çizgi)(14).

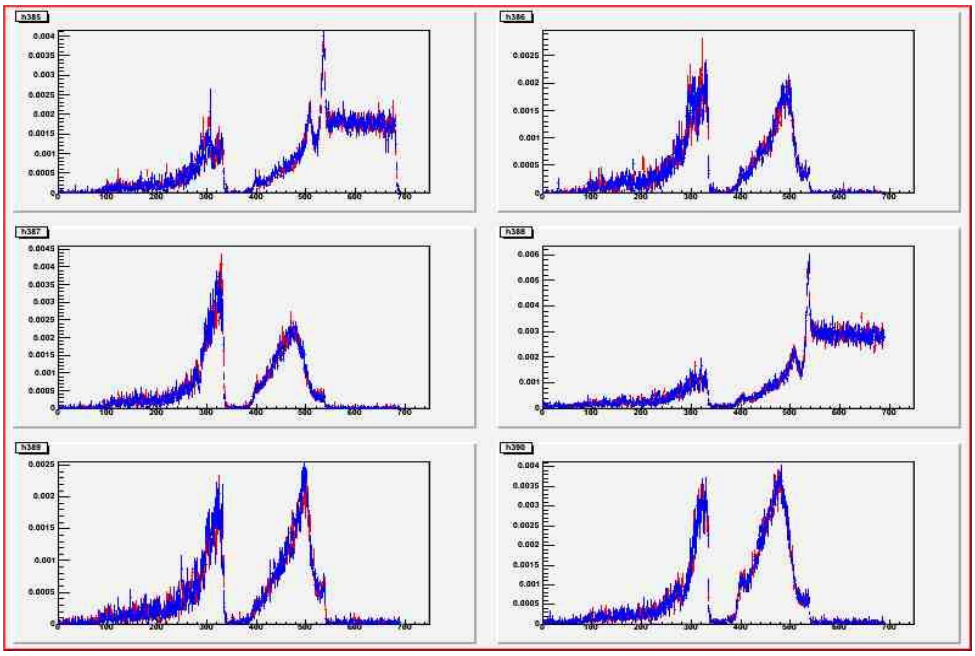


Şekil 20 : Takoz 2.01 tower 13 ile elde edilen sonuçlar. Kırmızı çizgi – kaynak hücrenin dışındayken katkını çıkarılmasından sonra QIE dağılımı. Mavi çizgi – dışdeğerbiçiminin sonucu

O zaman doğrusallaşan QIE sayımının süresindeki toplam yük her bir çıkarılan için hesaplanır ve aşağıdaki şekilde formalize edilir:

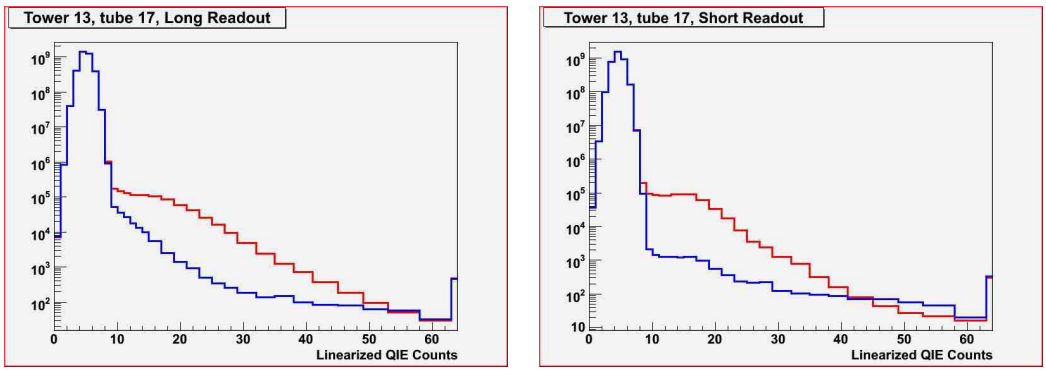
$$Q_i = \frac{40MHz}{\sum_{j=1}^{32} n_j} \cdot \sum_{j=ip+1}^{32} n_j \times (q_j - p) \qquad (7)$$

Burada n_j histogramın j'inci aralığındaki girişlerin sayısı, p temel pozisyon, ip temel p değerini içeren aralıktır. Kaynak kalibrasyonunun keskinliği geometriksel ölçü faktörü için MC (Mean Charge) hesaplarının keskinliğine bağlıdır. Kaynağın içindeki fiberlerin sayısına ve hücre içindeki kaynak tüpün göreceli pozisyonuna bağlıdır. Bu değerler farklı takozlardan aynı hücreler için biraz farklıdır. Şekil 21'de metotla elde edilen sonuçlar gösterilmektedir.



Şekil 21 : "Signal Extrapolation Under the Pedestal Peak" metoduna göre takoz 2.01 tower 13 için L ve S hücreleri için kaynağın pozisyonun bir fonksiyonu olarak kalorimetre tepkisi.

"Signal Extrapolation Under the Pedestal Peak" metoduna göre S readout için kalibrasyon katsayısı 0.353074 ve L readout için ise 0.487847 olarak hesaplandi. Bu metotta ilk iki metodun avantajları birleştirilmektedir: temel peakin altındaki sinyal yükü hesaba katılır (20.1'deki bölümde tanımlanan metottaki gibi) ve toplanan yüke büyük olasılıkla temel pozisyon tarafından etki edilmez (20.2'deki bölümde tanımlanan metottaki gibi). Bu metodun keskinliği küçük QIE değerlerinin bölgesinde dışdeğerbiçiminin belirsizliğinden dolayı sınırlanılır. Şekil 23 QIE dağılımlarını göstermektedir.



Şekil 22 : Takoz 2.01 tüp 13 için EM (soldaki) ve HAD (sağdaki) QIE dağılımlar

20. SONUÇLAR

5 mCi Co^{60} kaynak kullanılarak elde edilen kalibrasyon sonuçları gösterilmektedir. Elde edilen tepki-kaynak pozisyonu histogramlarında kalorimetre tepkisinin sıfırdan farklı olduğu noktalar göz önünde tutulmalıdır. Aslında üç farklı metot olmasına rağmen aynı noktalardan elde edilen kalibrasyon sonuçları aynıdır. Yöntemler sonucunda bulunan kalibrasyon katsayıları kalibrasyon keskinliği olarak adlandırılan $C_i^{kaynak} / C_i^{ışın}$ değerin belirlenmesi için kullanılır. Her bir metot için elde edilen sinyal ve background histogramlar, kalibrasyon katsayılarının EM readout için daha büyük olduğunu göstermektedir. Bu tezde yapılan çalışmalar ve verilen sonuçlar bir on çalışmanın sonucu olup HF ile ilgili detaylı kalibrasyon çalışmaları daha sonra yapılacaktır.

KAYNAKLAR

1. CMS NOTE 2001/020, "Radiation Hardness Studies of High OH^- Content Quartz Fibres Irradiated with 500 MeV Electrons"

2. Ömer AYFER, "Kim Korkar Linux'ten", Pusula Yayınevi, Ekim-2004

3. Kaan ASLAN, "UNİX/LİNUX Sistem Programlama Ders Notları", http://www.csystem.org/file_archive/unix.pdf

4. http://www.belgeler.org/arsiv/archive.html

5. Kaan ASLAN,"C ve C++ Programlama Ders Notları"http://www.csystem.org/file_archive/c.pdf

6. Necati ERGİN, "Visual C++ 6.0", http://www.csystem.org/file_archive/cplusplus2_06.pdf

7. **"**ROOT an Object-Oriented Data Analysis Framework User Guide 3.05"

8. http://atlas.web.cern.ch/Atlas/Welcome.html

9. P. Gorodetzky, Rad. Phys. And Chem., "Forward Calorimeter Layout and Design", http://www.iop.org/EJ/abstract/0954-3899/30/12/N01

10. İsa DUMANOĞLU, "CERN'deki CMS Deneyinin Bir Alt Detektörü Olan En İleri Kalorimetre", http://kutuphane.taek.gov.tr/internet_tarama/dosyalar/cd/4115/bildiri.html

11. CMS NOTE 2006/044, "Design, Performance, and Calibration of CMS Forward Calorimeter Wedges"

12. CERN, "The Hadron Calorimeter Project, Technical Design Report", 1999

13. CMS Internal Note 2004/002, "Response of a Quartz Fibre Calorimeter to Radioactive Sources"

14. CMS Internal Note 2006/035, "HF Source Calibration"