Guida di sopravvivenza a root

Nicolò Salimbeni

2021

Contents

Chapter 1

INTRODUZIONE

Questo PDF non vuole essere sostituto o una traduzione dei manuali ufficiali di root. Tengo a sottoloneare che NON sono un professore ma uno studente. Ho solamente perso una quantità notevole di ore di sonno per imparare tutte queste cose da solo vagando nel web. Ad un certo punto le cose erano tante e ho iniziato a scriverle, quando mi sono reso conto che potevano essere utili anche a qualcun'altro ho sistemato la formattazione ed è venuto fuori questo. Potrebbero esserci cose sbagliate o inaccurate, chiunque se ne accorga può scriverlo a questa mail: `nicolosalimbeni@gmail.com`, provvederò a sistemare l'errore e aggiornare la versione del PDF.

Prova di un codice qualunque di root con referenza 4.3:

```
TCanvas* c = new TCanvas("c", "c", 1000, 450, 1300, 650);
c->SetGrid();
c->SetBottomMargin(0.12591304);

TGraphErrors* TC_plot = new TGraphErrors(freq_ROOT, A_ROOT
  , err_freq_fake_ROOT, err_A_ROOT);
TC_plot->SetTitle("Funzione di Trasferimento Condensatore
  (T_{C}); f (Hz); A=V_{out}/V_{in}");
TC_plot->SetMarkerStyle(20);
TC_plot->SetMarkerSize(0.5);
TC_plot->GetXaxis()->SetLabelSize(0.047);
TC_plot->GetYaxis()->SetLabelSize(0.047);
TC_plot->GetXaxis()->SetTitleSize(0.052);
TC_plot->GetYaxis()->SetTitleSize(0.052);
TC_plot->GetYaxis()->SetTitleOffset(1);
TC_plot->GetXaxis()->SetMaxDigits(3);
TC_plot->Draw("AP");
```

Codice 1.1: Codice

1.1 Richiami di C++ TODO

1.2 Basi per capire cosa sta succedendo

1.3 Consigli per la lettura

Chapter 2

OGGETTI PRINCIPALI

2.1 TVectorD

Sono gli equivalenti dei vettori di C++. Servono in alcune funzioni di root specifiche, come ad esempio per inserire i dati nei TGraph. funzionano come array, bisogna definirli con una dimensione come segue:

```
1  TVectorD x(n);
```

Dove n è un intero per la dimensione (se n=10 le posizioni sono da 0 a 9), ed x è il nome dell'oggetto. Vanno riempiti come gli array, ma ci sono molte funzioni che lavorano su di questi (vanno richiamate con l'operatore .).

- .GetNoElements() : restituisce un int con il numero di elementi
- .GetLwb() : "Get lower bound" ritorna la prima posizione, che è essenzialmente 0
- .GetUpb() : "Get upper bound" ritorna l'ultima posizione, quindi se ho 10 elementi ritorna 9
- .ResizeTo(int n) : ridimensiona il vettore, se n=10 allora il vettore dopo aver chiamato la funzione avrà 10 elementi, da 0 a 9.

2.2 TH1

2.2.1 Costruttore

Per creare un istogramma l'oggetto da utilizzare è il TH1D.

```
1  TH1D* h = new TH1D("name","title",nbin,begin,end)
```

Codice 2.1: Costruttore più comune per il TH1D

- "name" è il nome con cui verà chiamato all'interno della funzioni di root, per convenzione lo si chiama come l'oggetto, in questo caso quindi bisognerebbe scrivere name=h. In generale poichè dietro le quinte root usare il name per riferirsi a questo istogramma, non possono essere chiamati due istogrammi diversi con lo stesso nome, in tal caso il secondo sovrascrive il primo.
- "title" è il titotlo che viene visualizzato sul canvas quanto viene disegnato l'istogramma.
- "nbin" è il numero di bin (intero);
- begin e end (double) sono dove inizia e finisce l'istogramma, questo è anche il range che verrà stampato.

Questo non è l'unico costruttore possibile, si veda la documentazione per gli altri forniti.

Attenzione: per convanzione i bin contengono il loro estramo inferiore, di conseguenza se end=10 e nbin=200 allora il bin che contiene 10 sarà il 201, il quale è al di fuori dell'istogramma quindi anche al di fuori del range stampato.

2.2.2 Funzioni principali

Vediamo ora le funzioni principali dell'oggetto:

```
1  h->Fill(3); //autmenta di un conteggio il bin che contiene
     3, ritorna il numero del bin in questione;
2  h->SetBinContent(455,25) //imposta il bin 455 a 25
    conteggi, non ritorna nulla;
3  h->Reset(); //elimina il contenuto dell'istogramma;
4  h->Draw(); //stampa
5  h->Draw("same"); //stampa in un canvas che contiene
    qualcosa
6  h->FindBin(2); //ritorna il numero del bin che contiene 2
7  h->GetMaximumBin(); //ritorna il bin contenente il massimo
8  h->GetMaximum(); //ritorna i conteggi del bin piu alto
9  h->GetBinContent(n); // ritorna il conteggio del bin n
10 h->GetMean(); //media
11 h->GetMeanError(); //errore media
12 h->GetStdDev(); //deviazione standard
13 h->GetStdDevError(); //errore sulla devstd
```

Attenzione: tutte le funzioni che iniziano con Get lavorano nel range in cui

l'istogramma è definito. Se però ad esempio durante l'analisi fai uno zoom sul canvas con il mouse lavorano nel range che vedi nel canvas (quello zoomato, NON tutto l'istogramma) quindi attento al range che utilizzi quando le chiami.

Viste queste funzioni più basilari vediamone alcune più articolate:

```
h2->Add(h1,c1); // default c1=1
h5->Add(h3,h4,c1,c2); // default c1=c2=1
```

La prima aggiunge i conteggi dell'istogramma h1 all'istogramma h2 con la seguente formula: h2 = h2 + c1* h1, questo permette in modo compatto di sommare sottrarre ecc... la seconda fa la somma degli istogrammi h3 ed h4 e la inserisce nel h5 come h5=h3*c1+h4*c2. Questo comando è utilizzabile anche se il numeri di bin dei vari istogrammi non è lo stesso, in tal caso viene chiamata una funzione apposita che cerca di aggiustare come può la situazione, di conseguenza cerca di evitarlo se possibile.
Una cosa da tenere in considerazione è che root in background mette in memoria delle informazioni riguardo i bin e gli errori dei bin dell'istogramma, tutto questo non ci deve interessare in quanto non vengono utilizzati direttamente dall'utente. Qualora però si voglia fare un fit è buona norma lanciare prima dei farlo il comando che segue

```
h->Sumw2();
```

per essere sicuri che il fit abbia tutte le informazioni corrette su cui basarsi. Tutto questo è descritto adeguatamente nella documentazione ufficiale dei TH1 (qui) nella descrizione della funzione Add() dove sono specificati i casi in cui è necessario, in cui è facoltativo, e le possibili eccezioni.
Un'altra funzione utile nell'analisi dati è:

```
h->Rebin(2)
```

Modifica il numero di bin dell'istogramma lasciando tutto il resto invariato. Nell'esempio 2 significa che due bin dell'istogramma prima della modifica diventeranno un solo bin dopo la modifica. In sostanza con 2 ne avrò la metà, con 3 un terzo ecc...

```
h->Integral();
h->Integral(100,500); // ritorna i conteggi tra il bin 100
   e 500
h->Integral(100,500,"width"); // integra tra 100 e 500

```

Il primo ritorna l'integrale di tutto l'istogramma, il secondo invece integra solo la parte compresa tra i bin 100 e 500 compresi. Attenzione, solo Integral(a,b) non fa l'intrale in senso comune, non calcola l'area dell'istogramma,

ma ritorna solo in numero di conteggi comrpresi tra quei bin, per avere l'area bisogna aggiungere l'opzione width. Nota bene, negli argomenti non si specifica il range sull'asse x, ma il numero del bin. Inoltre ricorda di tenere in considerazione il fatto che i bin contengono solo il loro estremo inferiore, per evitare di integrare erroneamente gli estremi. Qualora usassi la funzione Find infatti per integrare sull'asse x da 0 a 10 bisogna usare gli estremi Find(0) e Find(10*0.9999999), poichè Find(10) corrisponde al bin che contiene 10 come estremo inferiore.
Per normalizzare un istrogamma invece si può usare la funzione

```
h->Scale(d); //d e' un double
```

in questo modo tutti i bin verranno moltiplicato per d, per capirci se si vuole normalizzare a 1 allora d=1./h->Integral() ecc...
Quando un istogramma viene stampato con Draw in automatico viene creato un box con le informazioni principali. È possibile modificarlo facendo stampare valori molto utili sia riguardo l'istogramma che riguardo eventiali fit. Il tipo di questo oggetto è TPaveStats, vedere come funziona al segunete link TODO

2.3 TGraph

2.3.1 TGraphErrors

2.4 TF1

Per definire una funzione uno dimensionale definita in un certo range può essere utilizzata la classa TF1.

```
TF1* f=new TF1("funzione","3*x+2",-10,10);
```

vediamo i parametri uno per uno

- "funzione": nome con cui dietro le quinte root chiama questo oggetto
- "3*x+2": tra le virgolette va messa la funzione da utilizzare, ne parleremo più dettagliatamente in seguito.
- -10,10 : questo è il range, si possono mettere sia interi che double, non deve essere per forza simmetrico, ed è il range che viene utilizzato poi per stampare la funzione se la si vuole plottare.

2.4.1 Vari modi per definire la funzione che si vuole usare

- **FUNZIONE INLNIE:** la funzione può essere definita direttamente dall'utente senza usare funzioni predefinite, in questo modo metodo rientra anche l'utilizzo delle funzione sin() cos() log() log(10) exp() ecc... la sintassi è la seguente:

```
TF1 *fa1 = new TF1("fa1","sin(x)/x",0,10);
fa1->Draw();
```

- **FUNZIONE INLINE CON PARAMETRI:** in root NON è possibile definire una funzione in questa forma:

```
Double_t a=3;
TF1 *fa1 = new TF1("fa1","a*sin(x)/x",0,10);
```

per utilizzare un parametro all'interno di una funzione definita dall'utente è necessario prima dire a root che la funzione contiene un generico parametro [n] e poi impostarne il valore desiderato all'eserno della definizione del TF1.

```
TF1 *fa = new TF1("fa","[0]*x*sin([1]*x)",-3,3);
fa->SetParameter(0,value_first_parameter);
fa->SetParameter(1,value_second_parameter);
```

il primo valore del comando SetParameter si riferisce al numero del parametro [0], [1] ecc... il secondo è il valore che si vuole impostare. Questo può essere specificato direttamente lì, es: SetParameter(0,3); oppure puè essere utilizzata una variabile definita esternamente es: SetParameter(0,k);
Quella utilizzata precedentemente non è necessariamente la sintassi più compatta o più leggibile per definire i parametri, possono anche essere utilizzate le seguenti funzioni:

```
TF1 *fa = new TF1("fa","[0]*x+[1]",-3,3);
fa->SetParameters(3,4);
```

la funzione SetParameters permette di impostarli tutti in una sola riga, scritta così associerà 3 a [0] e 4 a [1], se ci fossero stati altri parametri la logica sarebbe stata la stessa.
Un'altra cosa utile a rendere il codice più leggibile è rinominare i parametri:

```
TF1 *fa = new TF1("fa","[0]*x+[1]",-3,3);
fa->SetParName(0,"m");
fa->SetParName(1,"q");
```

```
4  fa->SetParameter("m",3);
5  fa->SetParameter("q",4);
```

Anche reimpostare i nome può essere fatto in modo più compatte utilizzando la stessa sintassi utilizzata per reimpostare i parametri con:

```
1  fa->SetParNames("m","q");
```

- **FUNZIONE DEFINITA ESTERNAMENTE:** La funzione se ha una definizione particolarmente lunga può essere definita esternamente come una normale funzione di C++ per poi essere utilizzate nella definizione del TF1:

```
1  Double_t myFunc(double x) { return x+sin(x); }
2  ....
3  TF1 *fa3 = new TF1("fa3","myFunc(x)",-3,5);
4  fa3->Draw();
```

Questi non sono tutti i modi per definire funzioni, molte di quelle con rilevanza fisica sono già definite dentro root nella classe "TMath", se si è interessati è consigliato leggere il manuale, in questa guida non si vuole andare così a fondo.
Gli esempio sono stati presi o basati su quelli del manuale di root a questo link, lì ci sono sia maggiori dettagli sui metodi trattati sia altri non menzionati.

2.4.2 Valutare un funzione e le sue derivate

Per valutare un TF1 chiamato f in un punto il comando è:

```
1  f->Eval(x);
```

Dove x è l'ascissa. Se invece si vuole valutare una delle derivate 1° 2° o 3° il comando è:

```
1  f->Derivative(x);
2  f->Derivative2(x);
3  f->Derivative3(x);
```

Se si vuole aumentare la precisione perchè la funzione è particolarmente strana si può aumentare con:

```
1  f->Derivative(x,0,k);
```

dove 0 è un valore di default che non va modificato e k è il parametro usato nell'algoritmo. Senza spiegre come funziona l'algoritmo che deriva basta sapere che più è piccolo più la derivata sarà precisa, il valore di default è

0.001. Se si vogliono maggiori informazioni sia su come viene calcolata la derivata sia sul significato dei parametri leggere il manuale ufficiale di root per la classe TF1 e cercare la funzione Derivative a questo link.

2.5 TFile

TO DO:
TBrowser

Permette di salvare dei file (con estensione .root) che contengono varie informazioni e oggetti. Possono contenere isotogrammi, funzioni, set di dati ecc... Questo per non dover ricaricare ogni volta tutti i dati e rifare tutta l'analisi per avere determinati valori: se costruisco una volta un istogramma e salvo tutto in un TFile la volta dopo quando mi servirà non devo ricaricare i dati, mi basterà prendere quello salvato che contiene già tutto.

2.5.1 Scrittura e creazione

Il costruttore per un TFile è il seguente:

```
1  TFile* file= new TFile("nome.root","opzione");
2  file->cd(); //opzionale
```

In questo modo nella cartella in cui si è eseguito root verrà creato il file "nome.root".
Vediamo quindi cosa scrivere al posto di "opzione":

opzione	descrizione
NEW oppure CREATE	crea il file se non esiste, altrimenti non fa nulla
RECREATE	crea il file, se esiste lo sovrascrive
UPDATE	apre un file esistente, se non lo trova lo crea
REED	apre un file in sola lattura

Table 2.1: opzioni comuni (ma non le uniche) per il costruttore di un TFILE

Ma a cosa serve il comando file->cd()? Tra le varie funzioni che si possono usare in root alcune riguardano i TFile, ma non si riferiscono ad un TFile specifico. Un esempio che vedremo subito è la funzione Write(), questa non sa in quale TFile salvare un oggetto se ce ne sono definiti più di uno, cd() dice a root che da lì in poi a meno che non sia esplicito per qualche motivo quando si lancia un comando o una funzione che si riferiscono ad un TFile si riferiranno al TFile corrente "file".

Per salvare i vari oggetti di root questi hanno una funzione chiamata Write, sarà quindi sufficiente, solo dopo aver creato il file di output, chiamare ad esempio:

```
hist->SetNameTitle("nome","titolo"); // se non e' gia' stato fatto
file->cd(); //se non e' stato gia' fatto
hist->Write();
```

Qui è chiaro a cosa serve il comando cd(), nella funzione Write() non si esplicita in alcun modo in quale TFile scrivere. Utilizzando il comando cd() da lì in poi Write scriverà nel TFile selezionato fino a nuovo ordine.
Se si desidera scrivere l'oggetto nel TFile file2 sarà sufficiente chiamare file2->cd() e poi eseguire di nuovo Write().
La funzione SetNameTitle invece serve per impostare appunto un nome e un titolo agli oggetti, questi attributi sono quelli che verranno salvati nel TFile e che ci permetteranno di riconoscere l'oggetto "hist" nel TFile in futuro, per questo è caldamente consigliato impostarli sempre (in ogni caso root ne mette di defualt per sicurezza).
Le funzioni principali per gestire nome e titolo sono le seguenti:

```
Object *obj;
obj->SetNameTitle("nome","titolo");
obj->SetName("nome");
obj->SetTitle("titolo");
```

Root in automatico associa un nome e un titolo agli oggetti definiti, ma questi sono sempre gli stessi per tutti i TH1D tutti i TF1 ecc... quindi salvare oggetti diversi dello stesso tipo con stessi nome e titolo rende molto fastidioso reuperarli, in quanto saranno difficili da distinguere. Ecco perché è buona norma impostarli prima di chiamare Write().
Infine una volta finito di salvare sul file questo va chiuso con

```
file->Close();
```

Attenzione: eseguire Close è fondamentale. Se si esce da root senza chiamare Close() nel 99% dei casi andrà tutto liscio perchè quando possibile root lo fa in automatico. Se però ad esempio si chiude il terminale su cui si sta eseguendo root Close() non viene chiamata è tutte le modifiche vengono perse irrecuperabilmente.

2.5.2 Lettura

```
TFile* file = new TFile("nome file","opzione"); //crea l'
 oggetto
file->cd();
```

Questo comando in generale crea l'oggetto file, al posto di "opzione" mettere l'opzione che si desidera tra UPDATE e REED (vedi tabella 2.5.1) che potrà poi essere utilizzato di seguito. Per sapere però cosa c'è dentro è possibile proseguire in due modi, o si usa un TBrowser (vedremo di seguito cosa è e come si usa), oppure si usa l'interprete da riga di comando e il comando ".ls". Con il TBrowser è possibile nagivare al suo interno come fosse una cartella, con il comando .ls invece a schermo verrà stampato il suo contenuto.
".ls" è un altro esempio del perché è necessario chiamare cd(), con .ls non si specifica in alcun modo di che TFile mostrare il contenuto, e a schermo verrà stampato quello del TFile corrente che va impostato in precedeza.
A questo punto sappiamo come vedere cosa c'è all'interno di un TFile, vediamo ora come recuperare il suo contenuto per utilizzarlo:

```
TH1D* h = (TH1D*)file->Get("nome oggetto");
```

Per questo comando è necessaria una conversione perchè la funzione Get ritorna un TObject*, tutti gli oggetti in root sono una derivata di questa classe, di conseguenza così va sul sicuro, tu però poiché sai il tipo effettivo dell'oggetto che stai copiando puoi direttemante convertire il puntatore al tipo corretto (in questo caso un TH1D*). Questo metodo è il più esplicito ma in realtà è possibile lasciar fare tutto a root in background, se nel TFile c'è un istogramma chiamato "h" dopo aver creato il TFile e aver eseguito il relativo cd() sarà possibile utilizzare l'istogramma semplicemente con il suo nome: sarà possibile eseguire h->Draw() h->Fill() ecc... lasciando fare in background a root tutte le operazioni di copia.

Attenzione: Questo metodo è un po' fuorviante, se ad esempio da un TFile si lascia fare tutto a root in background sarebbe naturale pensare che chiamando ad esempio h->Fill(3) nel TFile l'istogramma abbia un conteggio in più nel bin che contiene 3, ma così non è. Non si sta modificando l'oggetto contenuto nel TFile ma una sua copia con lo stesso nome. Per salvare eventuali modifiche bisogna ricordare si chiamare h->Write()

2.6 Gestione oggetti all'interno di un TFile

Il TFile non va interpretato come una "cartella" di un file manager in cui gli oggetti si possono rinominare o modificare direttamente. È più una "foto"

degli elementi che contiene. Questi elementi possono essere copiati per essere utilizzati altrove o eliminati, ma non possono essere direttamente modificati. Vediamo per passi come gestire un TFile:
Inserimento: per aggiungere un oggetto basta seguire la procedura descritta precedentemente, questo nel file sarà riconducibile ad un nome e un titolo.

Salvare più volte oggetti con lo stesso nome: Questa situazione può presentarsi in due occasioni: nel TFile c'è una versione precedente di un oggetto, questo è stato copiato, modificato, e ora va ri-salvato senza modificare nome e titolo, chiamiamo questa situazione "aggiornamento", oppure non sono stati impostati nome e titolo personalizzati per due oggetti distinti (esempio due istogrammi) e salvandoli nel TFile sembra che lo stesso oggetto sia stato salvato due volte (con nome e titolo di defaul), chiamiamo questa situazione "errore nomi". Vediamo prima come root gestisce i salvataggi nel TFile e poi valutiamo le due situazioni precedenti.
I TFile in automatico non sovrascrivono gli oggetti, qualora si salvasse più volte un oggetto con lo stesso nome (ad esempio "istogramma") chiamando .ls nel TFile risulteranno più oggetti "istogramma" con nomi: "istogramma;1" "istogramma;2" e così via. Nel TFile vengono salvate tutte le versioni di un oggetto in modo da poter recuperare le precedenti in caso di errore. Per distinguerli quando si chiama la funzione Get è sufficiente scrivere:

```
1  TH1D *h = (TH1D *)file->Get("istogramma;1");
2  //oppure
3  TH1D *h = (TH1D *)file->Get("istogramma;2");
```

La chiamata con Get("istogramma") ritornerà in automatico l'ultima versione disponibile, in questo caso la 2. Vediamo ora come gestire le due situazioni iniziali con oggetti di tipo TH1D come esempio:

- **Aggiornamento:** dopo aver copiato, e modificato un TH1D di un TFile questo va risalvato con Write(), nel TFile comparità la sua versione "istogramma;2" e tutto è filato liscio. A questo punto si può eliminare la versione precedente oppure lasciare tutto così come è.

- **errore nomi:** per sbaglio due oggetti distinti di tipo TH1D con nome di default sono stati salvati in un TFile, all'interno di questo, poichè hanno stesso nome risultano essere due versioni di uno stesso oggetto: "istogramma;1" e "istogramma;2". Non c'è modo di rinominare direttamente uno dei due per renderli distinguibili, per risolvere il problema bisogna fare quanto segue:

  ```
  1 TH1D *h = (TH1D *)file->Get("istogramma;1");
  ```

```
h->SetName("nome personalizzato");
h->Write();
file->Delete("istogramma;1");
```

in questo modo si copia l'oggetto, gli si da un nome personalizzato, lo si salva di nuovo nel file, e si elimina il corrispettivo con quello di defualt. Questa procedura va fatta almeno per uno dei due così da distinguerli, poi è consigliato farla su entrambi per maggiore chiarezza.

Nota: Il titolo non concorre a questo problema di riconoscimento, se due oggetti hanno lo stesso nome e titolo diversi comunque nel TFile verranno create le versioni 1 e 2, ma almeno all'utente a prima vista queste saranno distinguibili tramite il titolo. Se invece anche questo è lo stesso tra le versione dell'oggetto l'unico modo per distinguerle è vedere cosa c'è dentro

---

Cancellare oggetti in un TFile: per questo esiste la funzione Delete(). Vediamo qualche esempio:

```
TFile *f=new TFile("file.root","UPDATE");
.....
f->Delete("nome"); //cancella l'oggetto "nome"
f->Delete("nome;1") //cancella la versione 1 di "nome"
f->Delete("*;1") //cancella la versione 1 di tutti gli
  oggetti
f->Delete("nome;*") // cancella tutte le versioni di "nome
  "
```

Inoltre quando in un TFile sono presenti più versioni di un oggetto è necessario spcificare quale/i versioni si desidera eliminare, chiamare un Delete("nome") generico non avrà alcun effetto.

2.7 TNtuple

Questo oggetto risulta particolarmente utile per la gestione dei dati da analizzare. Serve per memorizzare in modo efficiente e pratico dei set di dati, come potrebbero essere delle cordinate spaziali (x,y,z) per un numero n (non necessariamnete noto) di eventi. Si può così evitare di dover definire 3 vettori per le varie coordinate, potendo gestire tutti i dati con un solo oggetto.
La limitazione più importante è che i TNtuple sono degli oggetti "write-once, read many times", di conseguenza un TNtuple non può essere modificato, l'unico modo per "modificarlo" è copiarlo in uno nuovo modificandolo come

si vuole. Altra sottigliezza invece è il tipo dei dati da registare, per le TNtuple i valori devono essere dei float (Float_t per la precizione), si può avere maggere precisione utilizzando i TNtupleD con dei Double_t, ma in genereale non si possono salvare elementi con un tipo arbitrario. Per questo vedremo i TTree successivamente.
Vediamo ora il costruttore di questo oggetto.

```
1  TNtuple *nt = new TNtuple("ntName","titolo","var1:var2:
    var3");
```

I primi due argomenti sono intuitivi e non necessitano di spiegazioni, per la lista delle variabili invece: non ce nè un numero massimo o minimo, nell'esempio del costruttore ce ne sono tre per caso, potevano essere due, quattro ecc... Invece di procedere in casi generali vediamo attraverso un esempio il loro utilizzo.

2.7.1 Inserimento e recupero dati

```
1  TNtuple *nt = new TNtuple("event_decay","posizione
    decadimenti","n_eve:x:y:z"); //costruttore
```

I principali metodi di inserimento sono due; evento per evento attraverso la funzione Fill

```
1  nt->Fill(0,0.3,0.1,-0.25); // inserisce un set di valori
```

Oppure leggendo un file di input

```
1  nt->ReadFile("nomefile");
```

Il file in questo caso dovrà essere nel formato:
n_eve x y z
commento
n_eve x y z
.....
La lettura ignorerà le righe che inizano per # interpretandole come commenti
È poi possibile recuperare i dati inseriti nel modo seguente:

```
1  float ex,ey,ez;
2  nt->SetBranchAddress("x",&ex);
3  nt->SetBranchAddress("y",&ey);
4  nt->SetBranchAddress("z",&ez);
5  nt->GetEntry(2);
```

Con il comando SetBranchAddress come dice il nome lega i float indicati alla variabile x, y, z della TNtuple. Utilizzando poi GetEntry(2) "carica" i valori dell'evento 2 in ex, ey, ez.

2.7.2 Stampare a schermo informazioni utili

Fino ad ora abbiamo visto comandi utili da inserire in un .cc da compilare per tirarne fuori un eseguibile, vediamo ora alcune funzioni utili da utilizzare nell'interprete da terminale dopo aver caricato una macro che contiene una TNtuple.

```
nt->Print(); // stampa le informazioni sul contenuto
nt->Scan();  // stampa tutto il contenuto
nt->Scan("x") // stampa tutti i valori di x
nt->GetEntries() //stampa il numero di eventi inseriti
```

È poi possibile accedere ai vari eventi sempre stampandoli sullo schermo in base alla "posizione" all'interno del TNtuple. Il primo inserito è lo 0, il secondo è l'1 e così via:

```
nt->Show(0); //stampa l'evento 0
```

2.7.3 Ricavare grafici da un TNtuple

Avendo al suo interno una grande quantità di dati sono state scritte delle funzioni che permettono di stampare vari grafici direttamente da un TNtuple:

```
nt->Draw("varexp","selection");  //generale
nt->Draw("x") // stampa un istogramma dei valori di x
nt->Draw("x*y") // stampa un istogramma dei valori di x*y
nt->Draw("x","x>0") // stampa un istogramma dei valori di
  x per cui x>0
nt->Draw("x:y") // stampa un grafico cartesiano (x,y)
nt->Draw("x:y","x>0 & y<0") // come sopra con le
  condizioni x>0 && y<0
```

Come si capisce dagli esempi qualora non venga immessa alcuna selezione verranno stampati tutti gli elementi. Inoltre è possibile stampare non solo le variabili ma anche funzioni di esse.
È chiaro che nello stampare i dati sia stato creato un istogramma o un TGraph temporaneo, è possibile però riempire un TGraph o un TH1 definito dall'utente in precedenza utilizzando il comando:

```
nt->Project("h","varexp","selection");
```

il primo argomento è l'oggetto che riempie: un TH1, TH2 o un TH3, gli argomenti successivi sono analoghi al precedente Draw(). Ve evidenziata però una differenza tra l'utilizzo di questa funzione da linea di comando o da una macro. Da linea di comando l'oggetto h non deve essere creato necessariamente in anticipo, verrà creato al momento in automatico e h sarà il puntatore all'itogramma in questione. In tal caso viene creato con un

nome, un titolo, un range e un binning automatici. Se invece si preferisce specificarli bisogna creare h prima di eseguire Project.
Se si usa questa funzione in una macro l'oggetto sarà comunque creato in automatico, ma nella macro non sarà possibile utilizzarlo come puntatore. Nel caricare il file contenente la macro root ritorna un errore in cui dice che h non è stato ancora definito. Se si vuole quindi usare l'istogramma h all'interno della stessa macro in cui lo si crea attraverso Project h va dichiarato prima di chiamare Project con il costruttore regolare, la funzione Project invece di crearlo e riempirlo riempirà quello già creato.
Altra nota è che in questo modo non possono essere estratti dei TGraph, per farlo bisogna usare un metodo più macchinoso:

```
nt->Draw("py:px","pz>4");
TGraph *gr = new TGraph(nt->GetSelectedRows(), nt->GetV2()
  , nt->GetV1());
```

Per non scendere nei dettagli (che non so nemmeno io) basta sapere che prima è necessario lanciare Draw(), in questo modo viene salvata la selezione, poi poi bisogna costruire il TGraph.

2.8 TTree

I TTree sono oggetti simili agli TNtuple, ma più generali, permettendo di mettere in memoria oggetti di un tipo qualunque. Come suggerisce il nome sono costruiti in analogia con un albero. L'albero generale (TTree) è l'oggetto che contiene tutte le informazioni, queste si dividono in rami (TBranch) che a loro volta possono contenere più elementi dello stesso tipo (che siano vettori, strutture o double). I singoli elementi all'interno di un TBranch sono detti leaves (foglie). Un albero puo avere più rami, uno che magari contiene dei vettori, uno che contiene strutture e così via.
Ecco perché i TTree sono la generalizzazione degli TNtuple, la logica è la stessa ma può essere applicata a oggetti di vario tipo.
Vediamo adesso il loro costruttore:

```
TTree *tree = new TTree("treename","titolo");
```

A questo punto per iniziare a riempirlo è necessario creare un branch. Per rimanere più generali possibile immaginiamo di voler mettere in memoria degli oggetti di tipo Object, che siano vector, strutture, classi personalizzate, non è rilevante ora. La sintassi per salvare oggetti nel TTree è la seguente:

```
Object ob;
TBranch *branch = tree->Branch("branchname",&ob,"titolo");
```

Il branch è stato legato all'oggetto ob, a questo punto dovrò utilizzare ob per inserire all'interno del branch le varie foglie (il tipo delle folgie è implicitamente definito da ob). Inserisco dentro ob tutto quello che mi serve per quel relativo oggetto e chiamo:

```
1  br->Fill();
```

Ora per inserire un'altra foglia dovrò ricaricare sempre dentro ob le informazioni che mi servono e richiamare Fill(). Si Creeranno così le varia foglie dentro il branch, nominate con l'indice 0, 1, 2 ecc...

2.8.1 Recupero dati salvati in un TTree

La scrittura e il recupero di un tree all'interno di un TFile sono analoghi a quelli descritti nella sezione apposita, di conseguenza non verranno trattati nuovamente. Vediamo invece più nel dettaglio come estraare i dati dal tree.

```
1  TBranch *br = tree->GetBranch("branchname"); //Seleziono
    il banch che mi interessa
2  Object ob;
3  br->SetAddress(&ob)
4  br->GetEntry(0)
```

La logica è analoga a quella per le TNtuple, all'interno di un branch ho molti oggetti (che immaginiamo essere le leaves). Poichè il tree verosimilmente avrà più branch per prima cosa devo sceglierne uno mettendo in memoria il suo puntatore. A Questo punto lego una variabili al branch e attraverso GetEntry(0) carico la prima leaf nell'oggetto ob, se avessi usato GetEntry(1) avrei caricato la seconda caricata ecc...

Chapter 3

PLOT GRAFICI TODO

Cose come i range, i label, le dimenzioni ecc...

Chapter 4

FIT

4.1 Introduzione generale

In questa sezione vediamo come fare dei fit, la logica è la stessa per tutti gli oggetti che hanno una funzione Fit, di seguito si farà riferimento ad un istogramma generico h come oggetto da fittare, ma questo potrebbe essere benissimo anche un TGraph o un TGraphErrors ad esempio (questo non è del tutto vero se si vuole approfondire l'argomento, ma le opzioni base, che sono le sole riportate, sono comuni a tutti gli oggetti che ci interessano).
La funzione generale per eseguire un fit è nella fomra:

```
h->Fit (TF1 * f1,
Option_t *    option = "",
Option_t *    goption = "",
Double_t    xmin = 0,
Double_t    xmax = 0
)
```

Dove f1 è la funzione con cui eseguire il fit, option è una stringa che specifica le opzioni del fit, goption una stringa con le opzioni grafiche, mentre xmin e xmax sono il reange entro cui viene eseguito il fit. Vediamo pezzo per pezzo le opzioni più rilevanti.
La funzione per eseguire il fit base dell'oggetto h attraverso il TF1 f è:

```
  h->Fitt(f);
```

Se ad esempio si ha intenzione di impostare un range senza fornire alcuna option e goption la sintassi deve essere le seguente:

```
  h->Fitt(f,"","",xmin,xmax);
```

Da cui è chiaro come fare se si vuole lasicare una o più opzioni di default.
L'algoritmo di defaul per trovare i parametri ricercati è la minimizzazione del χ^2. Il range predefinito è quello di definizione dell'istogramma. Ultima

cosa a cui presare attenzione è che in questo modo verrà memorizzato solo l'ultimo fit eseguito rimuovendo i precedenti, se quindi si volessero fare più fit su uno stesso grafico sarà necessario cambiare le opzioni manualmente (vedi paragrafo successivo).
Inoltre la TF1 fornita per eseguire il fit deve avere dei parametri nella forma [0] [1] ecc... come mostrato nel paragrafo 2.4.1 alla voce "funzione inline con parametri"

4.2 Fit options

La sintassi per inserire opzioni aggiuntive nel fit è la seguente:

```
h->Fit(f,"opzione")
```

Nell'esempio si sottointende che si vuole fittare un generico oggetto h, che sia un istogramma, un grafico ecc... con una generica funzione f. Vediamo le opzioni che possono tornare utili più spesso:

- "R" : prende il range di definizione del TF1, fa il fit solo per i punti in quell'intervallo, e stampa il grafico in quell'intervallo

  ```
  h->Fit(f,"R");
  ```

- "+" : se sono stati fatti fit precedenti, solitamente viene stampato solo l'ultimo, se invece metti il + allora questo fit non sovrascive i precedenti e nel plot lì troverai tutti

  ```
  h->Fit(f,"+");
  ```

- "0" : se vuoi fare il fit, ma vuoi evitare di stamparlo sul plot

  ```
  h->Fit(f,"0");
  ```

- "S" : nel ritornare i risultati (vedi gestione risultati) nel TFitResult vengono salvate tutti le possibili informazioni, tra cui la matrice di covarianza, alla quale normalmente non sarebbe possibile accedere

  ```
  h->Fit(f,"S");
  ```

Se si volesse usare più di una opzione è sufficiente metterle in fila come segue:

```
h->Fit(f,"RS+");
```

4.2.1 Inserimento parametri e accortezze varie

L'algoritmo di minimizzazione non è infallibile, specilamente per funzioni particolarmente strane. Ciò significa che se lo si esegue senza dargli alcuna indicazione è molto probabile che dia risultati sbagliati. Per evitare il problema quando possibile è bene fornire dei dati iniziali da cui farlo partire. Per fare ciò si può usare la funzione SetParameter() (o simili), che è stata mostrata nel paragrafo 2.4.1. I dati iniziali non hanno nulla a che fare con C++ o root ma vanno determinati in base alla situazione fisica che si sta studiando.

Attenzione: Spesso anche se il fit è errato il χ^2 da valori accettabili, e i parametri trovati possono differire anche solo di pochi ordini di grandezza da quelli corretti, quindi per funzioni strane prestare particolre attenzione ai dati iniziali forniti all'algoritmo

---

Per dare ulteriori informazioni su cui basarsi a root è possibile utilizzare le seguenti funzioni:

```
1  f->SetParameter(0,5); //valore iniziale parametro 0 a 5
2  f->SetParLimits(0,-3,3); //parametro 0 tra -3 3
3  f->FixParameter(0,4); //fissa il parametro 0 a 4
```

4.2.2 Gestione dei risultati

La funzione Fit oltre a fare il fit ritorna anche un oggetto specifico per la gestione dei risultati di quest'ultimo (TFitResultPtr, che è essenzialmente un puntatore ad un oggetto di tipo TFitResult). Il modo più rapido quindi per recuperare tutte le informazioni rilevanti è il seguente:

```
1  TFitResultPtr r = h->Fit(myFunc,"S");
2  TMatrixDSym cov = r->GetCovarianceMatrix();  //  matrice
    di covarianza
3  Double_t chi2 = r->Chi2(); // chi2
4  Double_t par0 = r->Parameter(0); // ritorna il parametro 0
5  Double_t err0 = r->ParError(0); // ritorna l'errore del
    parametro 0
6  r->Print("V");     // stampa tutte le informazioni
    possibili
7  r->Write();        // salva i risultati in un file
```

L'oggetto r potrà poi essere salvato in un TFile, i fit a livello computazionale possono essere molto onerosi, in questo modo se in futuro si dovessero recuperare i risultati di un fit invece di rieseguirlo sarà suffciente leggerli all'interno del TFitResult salvato nel TFile.

4.3 Tips

Fare il fit in un range, poi stampare f in un altro range: la cosa più comoda è prima fare come segue:

```
h->Fit(f,"0","",xmin,xmax);
f->SetRange(xinf,xsup);
f->Draw();
```

in questo modo si calcola f, si sceglie il range desiderato per il plot e poi la si stampa.
mostrare i risultati di un fit direttamente sul grafico: inserire nel codice della macro:

```
gStyle->SetOptFit(1);
```