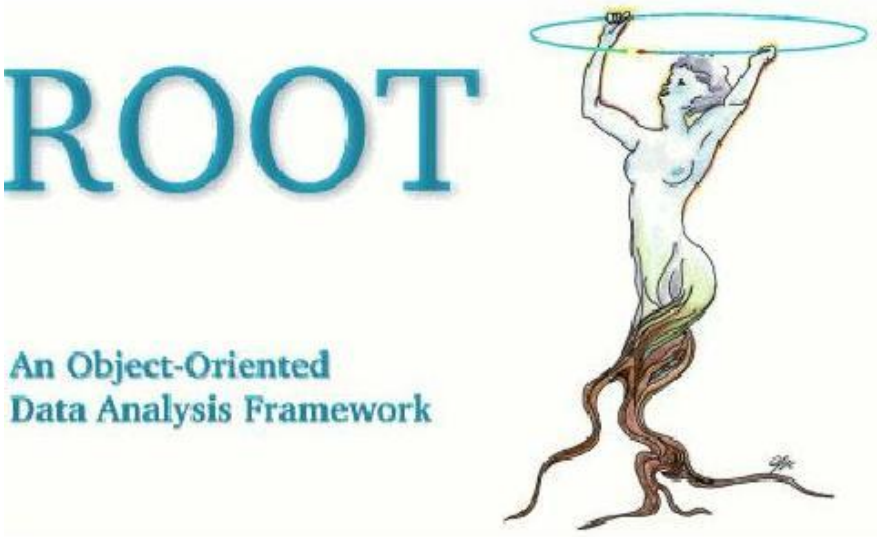
Manuale di CROOT
(interfaccia semplificata a ROOT)

(Laboratorio di Calcolo A/B a.a. 2002-2003)



Indice

Capitolo 1

Introduzione

ROOT è un pachetto completo di programmi per l'analisi dati (comprende, tra l'altro, funzioni di grafica 2D e 3D, di gestione di istogrammi e di minimizzazione) e viene distribuito per la maggior parte dei sistemi operativi oggi in uso.

In questo breve manuale si descrive la libreria CROOT (pronuncia ci-root), un'interfaccia semplificata a ROOT creata per le esigenze dei corsi di Laboratorio di Calcolo del primo anno. Cenni sull'utilizzo diretto delle librerie di ROOT verranno fornite nell'Appendice B.

ROOT fornisce, in maniera naturale, un'ampia interfaccia interattiva che permette di modificare gli attributi dei grafici realizzati (colori, forma dei punti, ecc...). L'utilizzo di quest'interfaccia, basata su pannelli di comando autoesplicativi (e quindi di facile utilizzo), non sarà trattato in questo manuale.

La documentazione completa su ROOT è disponibile sul sito *http://root.cern.ch* mentre le istruzioni per installare la libreria CROOT sono disponibili sul sito del corso di Laboratorio di Calcolo *http://www.fisica.unige.it/labc*

1.1 Utilizzare le librerie grafiche

Per utilizzare le librerie CROOT occorre includere, nel file sorgente, l'header file opportuno:

```
#include <croot.h>
```

Nella fase di compilazione, in ambiente Linux, occorre includere le librerie attraverso la macro `crootlib`

```
> g++ sorgente.cpp -o eseguibile `crootlib`
```

Capitolo 2

Manuale di CROOT

2.1 Introduzione al manuale

Le funzioni di CROOT vengono presentate fornendo, per ciascuna, il prototipo ed una breve descrizione del modo di funzionamento. Le stringhe di caratteri devono essere passate alle funzioni come stringhe C (tipo `char *`).

2.2 Inizializzazione e controllo della finestra grafica

`void R_Init(void)`

Apre ed inizializza la finestra grafica

`void R_End(void)`

Chiude la finestra grafica

`void R_Divide(int nx,int ny)`

Divide la finestra in `nx` zone lungo x e `ny` zone lungo y (le zone o sottodivisioni sono dette **pad**).

Esempio: l'istruzione

```
R_Divide(2,3);
```

divide la finestra in 6 pad uguali.

La pad corrente dopo una chiamata a `R_Divide` è la prima. Ogni volta che si disegna un nuovo oggetto (senza sovrapposizione) la pad corrente viene aggiornata alla successiva. Esaurite tutte le pad della finestra, la finestra viene ripulita (si gira pagina) e si riparte dalla prima pad. Le opzioni che permettono di intervenire tra un cambio pagina e l'altro sono regolate dalla funzione `R_SetWait`.

`void R_cd(int n)`

Definisce la pad `n` (vedi `R_Divide`) come pad corrente.

`void R_SetWait(short int iopt)`

Imposta il comportamento del programma al cambio pagina.

`iopt=0` la pagina viene girata automaticamente (senza attendere);

`iopt=1` (default) alla fine di ogni pagina il programma attende che sia inserito da terminale un carattere (`'q'` o `'n'`) per continuare

`iopt=2` alla fine di ogni pagina viene attivato il pannello di comandi.

`void R_SetUpdate(short int iopt)`

Gestisce l'aggioramento dei grafici in finestra.

`iopt=0` l'aggiornamento è a cura dell'utente (via `R_Update`);

`iopt=1` (default) l'aggiornamento è automaticamente eseguit ogni volta che si modifica un grafico esistente o se ne crea uno nuovo.

`void R_Update(void)`

Forza l'aggiornamento della finestra grafica con gli ultimi oggetti disegnati

`void R_Interact(void)`

Ferma l'esecuzione del programma e attiva il pannello di comandi.

`void R_PrintPicture(const char *file)`

Salva l'immagine disegnata sulla pagina corrente sul file il cui nome è dato dalla stringa `file`. Formati possibili per il file: `.ps` (consigliato per la stampa), `.eps`, `.gif`.

2.3 Grafica 2D

`void R_DrawFrame(double xmin, double xmax, double ymin, double ymax)`

Disegna, nella pad corrente, un sistema di assi x-y con estremi `xmin`, `xmax` sull'asse x e `ymin`, `ymax` sull'asse y.

`void R_ResizeFrame(double xmin, double xmax, double ymin, double ymax)`

Ridefinisce il sistema di assi x-y nella pad corrente.

`void R_Graph(int n, double *x, double *y, const char *opt)`

Disegna `n` punti con coordinate date dai vettori `x` e `y`. La stringa `opt` può selezionare le seguenti opzioni:

`P` ogni punto è disegnato con il marker corrente;

`L` i punti sono uniti con una spezzata;

`C` i punti sono uniti con una curva continua;

`A` gli assi sono disegnati (ed i loro estremi automaticamente determinati); se l'opzione `A` non è specificata il grafico viene disegnato sugli assi (o sul grafico) pre-esistenti (opzione da usare in alternativa a `R_DrawFrame`).

```
void R_GraphErrors(int n, double *x, double *y, double *ex, double *ey,
     const char *opt)
```

Disegna `n` punti con coordinate date dai vettori `x` e `y` e rappresenta i loro errori, `ex` e `ey`, con barre. Le opzioni sono comuni a `R_Graph`.

```
void R_PlotFunc1D(double fun(double), double xmin, double xmax,
     const char *opt)
```

Disegna la funzione `fun` tra gli estremi `xmin` e `xmax`. Le opzioni sono comuni a `R_Graph` (per default `*opt="C"`).

```
void R_PlotFuncPar1D(double fun(double x, double *par), int npar,
     double *par, double xmin, double xmax, const char *opt)
```

Disegna la funzione `fun` tra gli estremi `xmin` e `xmax` con i valori degli `npar` parametri assegnati dal vettore `par`. Le opzioni sono comuni a `R_Graph` (per default `*opt="C"`).

```
void R_Text(double x, double y, const char *opt)
```

Scrive, alle coordinate `x` e `y`, la stringa `opt`.
La stringa `opt` può contenere chiavi:

`^` : `x^{2}` da x^2;

`_` : `x_{2}` da x_2;

`#` : permette di scrivere lettere greche

```
#alpha #beta #gamma #delta #epsilon #zeta #eta #theta #iota
#kappa #lambda #mu #nu #xi #omicron #pi #varpi #rho #sigma
#tau #upsilon #phi #varphi #chi #psi #omega #Gamma #Delta
#Theta #Lambda #Xi #Pi #Sigma #Upsilon #Phi #Psi #Omega
```

o simboli matematici: tra cui `#sqrt{m}` (che da $\sqrt{m}$) e gli altri simboli in tabella


```
void R_Line(double x1, double y1, double x2, double y2)
```

Disegna la linea congiungente due punti di coordinate `(x1,y1)` e `(x2,y2)`.

```
void R_AxisTitles(const char *cx, const char *cy)
```

Scrive come titolo sull'asse `x` la stringa `cx` e sull'asse `y` la stringa `cy`.

2.4 Grafica 3D

```
void R_Draw3DFrame(double xmin, double xmax, double ymin, double ymax,
     double zmin, double zmax, const char *opt)
```

Definisce un sistema di assi 3D con estremi `xmin-xmax`, `ymin-ymax` e `zmin-zmax`. Se `*opt="N"` gli assi non sono disegnati.

```
void R_3DGraph(int n, double *x, double *y, double *z, const char *opt)
```

Grafica `n` punti (in uno spazio 3D) con coordinate date dai vettori `x`, `y`, `z`. Le opzioni definite da `opt` sono le stesse di `R_Graph`: manca solo l'opzione `A`.

```
void R_3DAxisTitles(const char *cx, const char *cy,const char *cz)
```

Scrive come titolo sull'asse `x` la stringa `cx`, sull'asse `y` la stringa `cy` e sull'asse `z` la stringa `cz`,

2.5 Grafici modificabili

Le funzioni:

```
void R_GraphObj(const char *name, int n, double *x, double *y,
     const char *opt)
```

```
void R_GraphErrorsObj(const char *name, int n, double *x, double *y,
     double *ex, double *ey, const char *opt)
```

```
void R_3DGraphObj(const char *name, int n, double *x, double *y,
     double *z, const char *opt)
```

sono analoghe a quelle senza il postfisso `Obj`.
La differenza principale è che queste funzioni creano grafici che sono modificabili nel seguito. Essi sono identificati dalla stringa `name` e, una volta creati con una chiamata alla funzione, possono essere aggiornati (modifica delle coordinate, riduzione/aumento dei punti) con una successiva chiamata alla stessa funzione con l'opzione `*opt="M"`.
Queste funzioni sono di particolare utilità se si vogliono rappresentare punti/linee in movimento.

```
void R_Delete(const char *name)
```

Cancella dalla memoria l'oggetto `name` (grafico modificabile o istogramma).

2.6 Istogrammi

Un istrogramma è un diagramma che rappresenta la distribuzione di frequenza di un certo tipo di evento in funzione di una o più variabili caratteristiche. Lo spazio delle variabili è suddiviso in intervalli regolari (canali o celle) compresi tra un valore minimo ed un valore massimo; l'istogramma è costruito disegnando, per ogni canale (o cella), un rettangolo (o un parallelepipedo in 2D) di base pari alla larghezza del canale e di altezza pari al numero di eventi che hanno variabile caratteristica contenuta in quella cella.
Nella gestione degli istogrammi si individuano tre fasi: creazione, riempimento, disegno.

```
void R_CreateHist1D(const char *hist, const char *test, int ncx,
     float lowx, float upx)
```

Crea un'istrogramma unidimensionale identificato con la stringa `hist` con titolo `test` di `ncx` canali con estremo inferiore `lowx` ed estremo superiore `upx`.

```
void R_CreateHist2D(const char *hist, const char *test, int ncx,
     float lowx, float upx, int ncy, float lowy, float upy)
```

Crea un'istrogramma bidimensionale identificato con la stringa `hist` con titolo `test` di `ncx(ncy)` canali lungo x(y) con estremi inferiore `lowx (lowy)` ed estremi superiori `upx (upy)`.

```
void R_FillHist1D(const char *hist, float x, float w)
```

Riempe l'istogramma `hist` con il valore `x` con peso `w`.

```
void R_FillHist2D(const char *hist, float x, float y, float w)
```

Riempe l'istogramma `hist` con la coppia di valori `x` r `y` con peso `w`.

```
void R_PlotHist(const char *hist, const char *opt)
```

Disegna l'istogramm `hist` con l'opzione `opt`. Opzioni generali:

`AXIS` : Disegna solo gli assi

`HIST` : Disegna solo il contorno dell'istrogramma (e non gli eventuali errori)

`SAME` : Sovrappone al precedente disegno sulla stessa pad

`CYL` : Usa coordinate Cilindriche

`POL` : Usa coordinate Polari

`SPH` : Usa coordinate Sferiche

`LEGO` : Disegna parallelepipedi per rappresentare il contenuto dei canali

`SURF` : Rappresenta il contenuto dell'istogramma con una superficie

A queste opzioni si aggiunge l'opzione `M` (Modifica) che aggiorna un'istograma già disegnato.
Opzioni per istogrammi 1D:

`C` : Disegna una curva continua che unisce le altezze dei canali dell'istogramma

`E` : Disegna le barre di errore

`E1` : Aggiunge alle barre di errore linee perpendicolari pari alla larghezza del canale

`L` : Disegna una linea che unisce le altezze dei canali dell'istogramma

`P` : Rappresenta ogni canale con il marker corrente

`*H` : Rappresenta ogni canale con un asterisco (*)

Opzioni per istogrammi 2D:

`ARR` : Disegna frecce che indicano il gradiente tra celle adiacenti

`BOX` : Disegna un quadrato per ogni cella con superficie proporzionale al suo contenuto

`COL` : Disegna un quadrato per ogni cella con una gradazione di colore variabile con il suo contenuto

`COLZ` : Come `COL`. In aggiunta stampa la tabella di colori

`CONT` : Disegna curve di livello

`CONT1` : Disegna curve di livello con tipi diversi di linea per ogni livello

`CONT2` : Disegna curve di livello con lo stesso tipo di linea per tutti i livelli

`CONT4` : Disegna curve di livello riempiendo l'area tra un livello e l'altro con colori diversi

`SCAT` : Disegna uno "scatter-plot" (disegno con puntini a densità variabile)

`TEXT` : Disegna, per ogni cella, il suo contenuto in forma testuale

```
void R_ResetHist(const char *hist)
```

Azzera il contenuto dell'istogramma `hist`.

```
double R_GetBinContentHist1D(const char *hist, int bin)
```

Ritorna il contenuto del canale `bin`.

```
double R_GetBinContentHist2D(const char *hist, int binx, int biny)
```

Ritorna il contenuto del canale (o cella) identificato dagli indici `binx` e `biny`.

```
double R_GetEntries(const char *hist)
```

Ritorna il numero di valori (entrate) immessi nell'istogramma.

```
int R_GetBinNumber(const char *hist, double x, const char *axis)
```

Ritorna il numero del canale dell'asse `axis`(=`"x"` o `"y"`) dell'istrogramma `hist` in cui è contenuto il valore `x`.

```
double R_GetBinCenter(const char *hist, int ibin, const char *axis)
```

Ritorna il centro del canale `ibin` dell'asse `axis`(=`"x"` o `"y"`) dell'istogramma `hist`.

```
int R_GetNBins(const char *hist, const char *axis)
```

Ritorna il numero di canali dell'istogramma `hist` lungo l'asse `axis`(=`"x"` o `"y"`).

```
void R_SetRangeHist(const char *hist, int imin, int imax, const char *axis)
```

Seleziona il numero di canali dell'istogramma `hist` lungo l'asse `axis`(=`"x"` o `"y"`).

```
void R_SetOptStat(int mode)
```

Definisce il tipo di informazioni mostrate nel pannello statistico dell'istogramma.
Il parametro `mode` può essere `iourmen` (default = 0):

- `n` = 1 : stampa il nome dell'istogramma
- `e` = 1 : stampa il numero di entries
- `m` = 1 : stampa il valore medio
- `r` = 1 : stampa l'RMS
- `u` = 1 : stampa il numero delle entries al di sotto dell'estremo inferiore
- `o` = 1 : stampa il numero delle entries al di sopra dell'estremo inferiore
- `i` = 1 : stampa il valore della somma dei canali

```
double R_GetMean(const char *hist, int axis)
```

Ritorna la media dell'istogramma `hist` lungo l'asse `axis`(=1,2,3).

```
double R_GetRMS(const char *hist, int axis)
```

Ritorna l'RMS (Root Mean Square = deviazione standard) dell'istogramma `hist` lungo l'asse `axis`(=1,2,3).

```
double R_GetIntegral(const char *hist, const char *opt)
```

Ritorna la somma dei contenuti dei canali dell'istrogramma, se `*opt="width"` moltiplicata per la larghezza (o area) del canale.

2.7 MINUIT

MINUIT è un package per la ricerca di minimi di funzioni multiparametriche.
La principale applicazione riguarda la stima, tramite il metodo dei minimi quadrati, del miglior valore di uno o più parametri e dei loro errori.
L'utente fornisce a MINUIT la funzione di χ^2 che calcola la somma degli scarti quadratici tra i dati sperimentali ed i valori aspettati (calcolati sulla base di una funzione teorica multiparametrica) pesati con gli errori sperimentali. Gli unici parametri liberi della funzione χ^2 sono i parametri della funzione teorica. MINUIT minimizza il χ^2 rispetto a tali parametri o, in altre parole, trova i valori dei parametri che danno il minor valore di χ^2.

```
void R_Minuit(double (*fun)(int np, double *xval), const char *mode,
     int npar, char **cpar, double *par, double *step, double *min,
     double *max, double *outpar, double *err, double *chiq)
```

Minimizza la funzione di χ^2 (`fcn`) fornita dall'utente e da la miglior stima dei parametri e dei loro errori.

Parametri di ingresso:

`fcn` : funzione di χ^2 fornita dall'utente:

```
double fcn(int npar, double *xpar){
  ...
  double chi2 = ...
  return chi2;
}
```

dove i parametri `npar` e `xpar` rappresentano rispettivamente il numero di parametri ed il vettore contenenti i valori dei parametri (forniti a `fcn` da MINUIT durante le varie fasi della minimizzazione).

`mode` : Stringa che specifica l'opzione desiderata

- `" "` Minimizzazione default
- `"M"` Minimizzazione interattiva

`np` : Numero di parametri (dimensione di `cpar`, `par`, `step`, `min`, `max`, `outpar` e `err`)

`cpar` : Vettore di stringhe contenenti il nome dei parametri

`par` : Vettore con il valore iniziale dei parametri

`step` : Vettore con i passi iniziali di variazione dei parametri

`min` : Vettore con i limiti inferiori dei parametri

`max` : Vettore con i limiti superiori dei parametri

Parametri di uscita:

`outpar` : Vettore con i valori ottenuti per i parametri

`err` : Vettore con i valori ottenuti per gli errori sui parametri

`chiq` : Valore del χ^2 nel punto di minimo

Osservazioni:

se `min=max=0.0` i parametri sono lasciati liberi di variare tra $-\infty$ e $+\infty$.

```
void R_MinuitLimit(double (*fun)(int np, double *xval), int npar,
     double *par, double *step, double *min, double *max,
     double *outpar, double *err, double *chiq)
```

Funzionalità identiche a `R_Minuit` eccetto:

- il modo di minimizzazione "default" è automaticamente selezionato
- il nome del parametri è fissato (*p0, p1, p2, ...*).

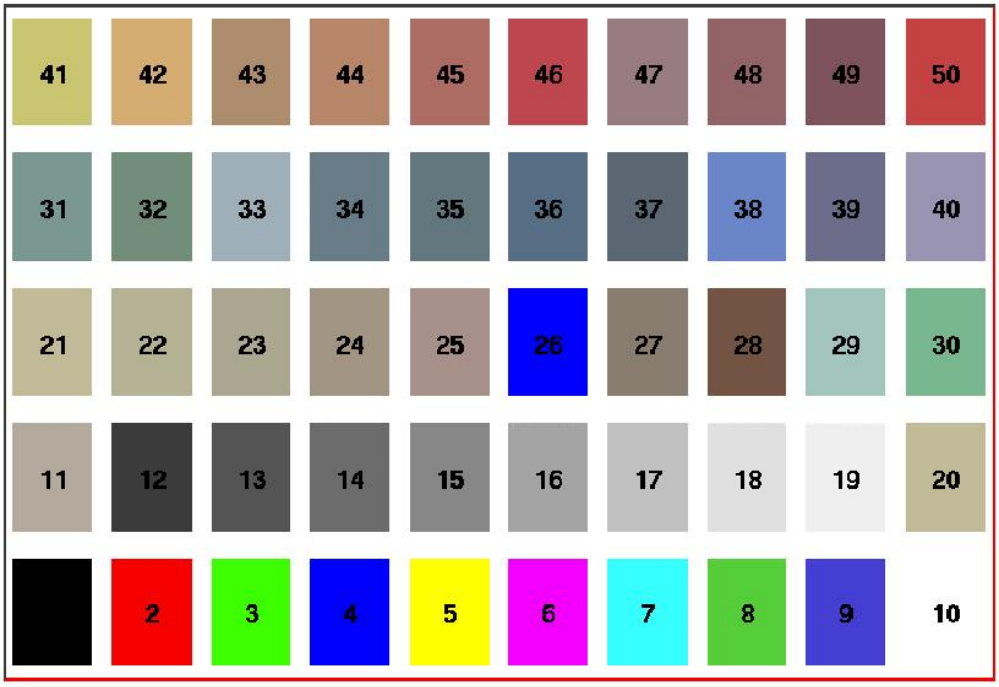


Tabella 2.1: Tavola degli indici di colore: 1=nero, 2=rosso, 3=verde, 4=blu, 5=giallo, 6=magenta, 7=azzurro, 10→19 sfumature di grigio, 20→29 sfumature di marrone, 30→39 sfumature di blu, 40→49 sfumature di rosso.

```
void R_MinuitSimpl(double (*fun)(int np, double *xval), int npar,
     double *par, double *outpar, double *err, double *chiq)
```

Funzionalità identiche a `R_MinuitSimpl` eccetto:

- i parametri non sono limitati.

2.8 Configurazioni generali ed opzioni

Molte delle opzioni qui descritte possono essere selezionate tramite il pannello di controllo (attivabile con `R_SetWait(2)`). È tuttavia utile, nel caso di operazioni ripetute, poter definire le opzioni desiderate nel programma, una volta per tutte.
Tutte le funzioni del tipo `R_SetXXXX` attivano opzioni che restano valide dalla chiamata alla funzione in poi.
I colori saranno nel seguito identificati con la variabile intera `icol` secondo lo schema in Tavola 2.8, il tipo di linea con `ilin` (1 continua, 2 tratteggiata, 3 punteggiata, 4 tratteggiata-punteggiata), il tipo simbolo associato ad un punto con `imark` secondo lo schema in Tavola 2.8

```
void R_SetOptLogx(short int iopt)
```

Imposta la scala logaritmica (lineare) per l'asse x se `iopt`=1(0 default).

```
void R_SetOptLogy(short int iopt)
```

Imposta la scala logaritmica (lineare) per l'asse y se `iopt`=1(0 default).

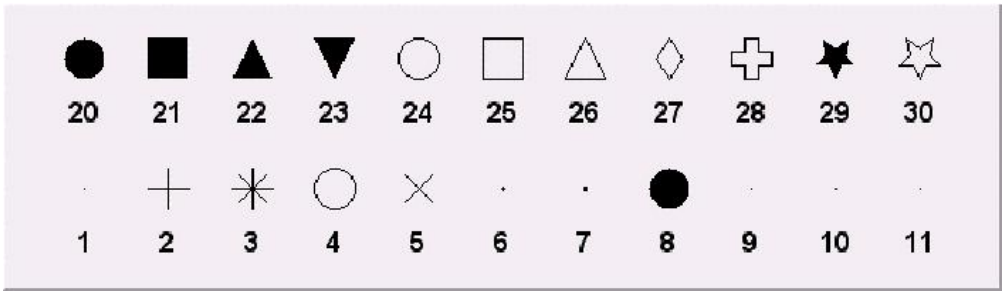


Tabella 2.2: Tavola dei simboli

`void R_SetOptLogz(short int iopt)`

Imposta la scala logaritmica (lineare) per l'asse z se `iopt`=1(0 default).

`void R_SetGrid(short int iopt)`

Predispone (se `iopt`=1) il disegno di una griglia di linee ortogonali nel piano x-y. È utile per visualizzare facilmente il valore delle coordinate di un punto.

`void R_SetMarkerStyle(short int imark)`

Definisce il simbolo (marker) associato al disegno dei punti secondo la Tavola 2.8.

`void R_SetMarkerColor(short int icol)`

Definisce il colore del marker.

`void R_SetMarkerSize(float iopt)`

Definisce la dimensione del marker.

`void R_SetLineStyle(short int ilin)`

Definisce il tipo di linea.

`void R_SetLineColor(short int icol)`

Definisce il colore della linea.

`void R_SetLineWidth(short int iopt)`

Definisce lo spessore della linea.

`void R_SetTitleOffset(float off, const char *opt)`

Imposta a `off` la distanza tra il nome dell'asse e l'asse `opt` (`*opt`="x" o "y").

`void R_SetHistLineStyle(short int iopt)`

Definisce il tipo di linea da utilizzare negli istrogrammi.

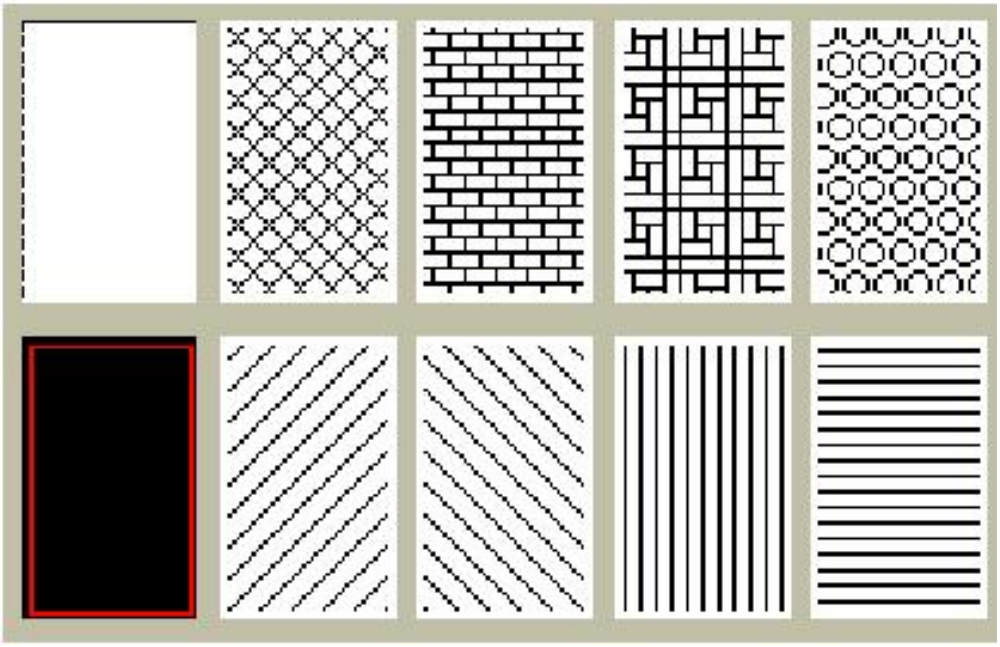


Tabella 2.3: Tavola degli stili per istogrammi

```
void R_SetHistLineColor(short int iopt)
```

Definisce il colore per il bordo dell'istrogramma.

```
void R_SetHistFillStyle(short int iopt)
```

Definisce lo "stile" che si deve utilizzare per l'istogramma secondo la convenzione:

0	vuoto (default);
1001	pieno;
2001	tratteggiato;
3001+`codice`	dove `codice` è definito in Tavola 2.8;
4000	finestra trasparente;

da `4001` a 4100 si passa da una finestra 100% trasparente a 100% opaca.

```
void R_SetHistFillColor(short int iopt)
```

Definisce il colore da utilizzare per riempire gli istogrammi.

```
void R_SetHistLineWidth(short int iopt)
```

Definisce lo spessore del bordo degli istogrammi.

Appendice A

Uso di MINUIT

Le funzioni `R_Minuit`, `R_MinuitLimit` e `R_MinuitSimpl` sono tutte interfacce alla libreria CERN chiamata MINUIT che è utilizzata, principalmente, per la soluzione numerica del metodo dei minimi quadrati in analisi statistiche di dati sperimentali. In questa appendice si riporta un'estratto molto parziale del manuale di MINUIT, limitato alle necessità di analisi relative al corso di Laboratorio di Calcolo.
`FCN` indicherà nel seguito la generica funzione da minimizzare.

A.0.1 Basic concepts of MINUIT

The transformation for parameters with limits.

For variable parameters with limits, MINUIT uses the following transformation:

$$P_{\text{int}} = \arcsin\left(2\frac{P_{\text{ext}} - a}{b-a} - 1\right) \qquad\qquad P_{\text{ext}} = a + \frac{b-a}{2}\left(\sin P_{\text{int}} + 1\right)$$

so that the internal value P_{int} can take on any value, while the external value P_{ext} can take on values only between the lower limit a and the upper limit b. Since the transformation is necessarily non-linear, it would transform a nice linear problem into a nasty non-linear one, which is the reason why limits should be avoided if not necessary. In addition, the transformation does require some computer time, so it slows down the computation a little bit, and more importantly, it introduces additional numerical inaccuracy into the problem in addition to what is introduced in the numerical calculation of the `FCN` value. The effects of non-linearity and numerical roundoff both become more important as the external value gets closer to one of the limits (expressed as the distance to nearest limit divided by distance between limits). The user must therefore be aware of the fact that, for example, if he puts limits of $(0, 10^{10})$ on a parameter, then the values 0.0 and 1.0 will be indistinguishable to the accuracy of most machines.
The transformation also affects the parameter error matrix, of course, so MINUIT does a transformation of the error matrix (and the "parabolic" parameter errors) when there are parameter limits. Users should however realize that the transformation is only a linear approximation, and that it cannot give a meaningful result if one or more parameters is very close to a limit, where $\partial P_{\text{ext}}/\partial P_{\text{int}} \approx 0$. Therefore, it is recommended that:

- Limits on variable parameters should be used only when needed in order to prevent the parameter from taking on unphysical values.
- When a satisfactory minimum has been found using limits, the limits should then be removed if possible, in order to perform or re-perform the error analysis without limits.

How to get the right answer from MINUIT.

MINUIT offers the user a choice of several minimization algorithms. The `MIGRAD` (Other algorithms are available with Interactive MINUIT, as described on Page 20) algorithm is in general the best minimizer for nearly all functions. It is a variable-metric method with inexact line search, a stable metric updating scheme, and checks for positive-definiteness. Its main weakness is that it depends heavily on knowledge of the first derivatives, and fails miserably if they are very inaccurate.
If parameter limits are needed, in spite of the side effects, then the user should be aware of the following techniques to alleviate problems caused by limits:

Getting the right minimum with limits.

If MIGRAD converges normally to a point where no parameter is near one of its limits, then the existence of limits has probably not prevented MINUIT from finding the right minimum. On the other hand, if one or more parameters is near its limit at the minimum, this may be because the true minimum is indeed at a limit, or it may be because the minimizer has become "blocked" at a limit. This may normally happen only if the parameter is so close to a limit (internal value at an odd multiple of $\pm\frac{\pi}{2}$ that MINUIT prints a warning to this effect when it prints the parameter values.
The minimizer can become blocked at a limit, because at a limit the derivative seen by the minimizer $\partial F/\partial P_{\rm int}$ is zero no matter what the real derivative $\partial F/\partial P_{\rm ext}$ is.

$$\frac{\partial F}{\partial P_{\rm int}} = \frac{\partial F}{\partial P_{\rm ext}}\frac{\partial P_{\rm ext}}{\partial P_{\rm int}} = \frac{\partial F}{\partial P_{\rm ext}} = 0$$

Getting the right parameter errors with limits.

In the best case, where the minimum is far from any limits, MINUIT will correctly transform the error matrix, and the parameter errors it reports should be accurate and very close to those you would have got without limits. In other cases (which should be more common, since otherwise you wouldn't need limits), the very meaning of parameter errors becomes problematic. Mathematically, since the limit is an absolute constraint on the parameter, a parameter at its limit has no error, at least in one direction. The error matrix, which can assign only symmetric errors, then becomes essentially meaningless.

Interpretation of Parameter Errors:

There are two kinds of problems that can arise: the **reliability** of MINUIT's error estimates, and their **statistical interpretation**, assuming they are accurate.

Statistical interpretation:

For discussuion of basic concepts, such as the meaning of the elements of the error matrix, or setting of exact confidence levels.

Reliability of MINUIT error estimates.

MINUIT always carries around its own current estimates of the parameter errors, which it will print out on request, no matter how accurate they are at any given point in the execution. For example, at initialization, these estimates are just the starting step sizes as specified by the user. After a `MIGRAD` or `HESSE` step, the errors are usually quite accurate, unless there has been a problem. MINUIT, when it prints out error values, also gives some indication of how reliable it thinks they

are. For example, those marked `CURRENT GUESS ERROR` are only working values not to be believed, and `APPROXIMATE ERROR` means that they have been calculated but there is reason to believe that they may not be accurate.
If no mitigating adjective is given, then at least MINUIT believes the errors are accurate, although there is always a small chance that MINUIT has been fooled. Some visible signs that MINUIT may have been fooled are:

- Warning messages produced during the minimization or error analysis.
- Failure to find new minimum.
- Value of `EDM` too big (estimated Distance to Minimum).
- Correlation coefficients exactly equal to zero, unless some parameters are known to be uncorrelated with the others.
- Correlation coefficients very close to one (greater than 0.99). This indicates both an exceptionally difficult problem, and one which has been badly parameterized so that individual errors are not very meaningful because they are so highly correlated.
- Parameter at limit. This condition, signalled by a MINUIT warning message, may make both the function minimum and parameter errors unreliable. See the discussion above "*Getting the right parameter errors with limits*".

The best way to be absolutely sure of the errors, is to use "independent" calculations and compare them, or compare the calculated errors with a picture of the function. Theoretically, the covariance matrix for a "physical" function must be positive-definite at the minimum, although it may not be so for all points far away from the minimum, even for a well-determined physical problem. Therefore, if MIGRAD reports that it has found a non-positive-definite covariance matrix, this may be a sign of one or more of the following:

A non-physical region: On its way to the minimum, MIGRAD may have traversed a region which has unphysical behaviour, which is of course not a serious problem as long as it recovers and leaves such a region.

An underdetermined problem: If the matrix is not positive-definite even at the minimum, this may mean that the solution is not well-defined, for example that there are more unknowns than there are data points, or that the parameterization of the fit contains a linear dependence. If this is the case, then MINUIT (or any other program) cannot solve your problem uniquely, and the error matrix will necessarily be largely meaningless, so the user must remove the underdeterminedness by reformulating the parameterization. MINUIT cannot do this itself.

Numerical inaccuracies: It is possible that the apparent lack of positive-definiteness is in fact only due to excessive roundoff errors in numerical calculations in the user function or not enough precision. This is unlikely in general, but becomes more likely if the number of free parameters is very large, or if the parameters are badly scaled (not all of the same order of magnitude), and correlations are also large. In any case, whether the non-positive-definiteness is real or only numerical is largely irrelevant, since in both cases the error matrix will be unreliable and the minimum suspicious.

An ill-posed problem: For questions of parameter dependence, see the discussion above on positive-definiteness.
Possible other mathematical problems are the following:

Excessive numerical roundoff: Be especially careful of exponential and factorial functions which get big very quickly and lose accuracy.

Starting too far from the solution: The function may have unphysical local minima, especially at infinity in some variables.

MINUIT interactive mode

The routine `HMINUIT`, with the `M` option, moves to interactive mode and gives control to the MINUIT program. In this case, the user may enter MINUIT control statements directly.

Overview of available MINUIT commands

CLEar

Resets all parameter names and values to undefined. Must normally be followed by a PARAMETER command or equivalent, in order to define parameter values.

CONtour par1 par2 [devs] [ngrid]

Instructs MINUIT to trace contour lines of the user function with respect to the two parameters whose external numbers are **par1** and **par2**. Other variable parameters of the function, if any, will have their values fixed at the current values during the contour tracing. The optional parameter **[devs]** (default value 2.) gives the number of standard deviations in each parameter which should lie entirely within the plotting area. Optional parameter **[ngrid]** (default value 25 unless page size is too small) determines the resolution of the plot, i.e. the number of rows and columns of the grid at which the function will be evaluated.

EXIT

End of Interactive MINUIT. Control is returned to the calling routine.

FIX parno

Causes parameter **parno** to be removed from the list of variable parameters, and its value will remain constant (at the current value) during subsequent minimizations, etc., until another command changes its value or its status.

HELP [SET] [SHOw]

Causes MINUIT to list the available commands. The list of SET and SHOw commands must be requested separately.

HESse [maxcalls]

Instructs MINUIT to calculate, by finite differences, the Hessian or error matrix. That is, it calculates the full matrix of second derivatives of the function with respect to the currently variable parameters, and inverts it, printing out the resulting error matrix. The optional argument **[maxcalls]** specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

IMProve [maxcalls]

If a previous minimization has converged, and the current values of the parameters therefore correspond to a local minimum of the function, this command requests a search for additional distinct local minima. The optional argument **[maxcalls]** specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

MIGrad [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, the most efficient and complete single method, recommended for general functions (see also MINImize). The minimization produces as a by-product the error matrix of the parameters, which is usually reliable unless warning messages are produced. The optional argument **[maxcalls]** specifies the (approximate) maximum number of function calls after which the calculation will be stopped even if it has not yet converged. The optional argument **[tolerance]** specifies required tolerance on the function value at the minimum. The default tolerance is `0.1`. Minimization will stop when the estimated vertical distance to the minimum (`EDM`) is less than `0.001*[tolerance]*UP` (see `SET ERR`).

MINImize [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, as does the MIGrad command, but switches to the SIMplex method if Migrad fails to converge. Arguments are as for MIGrad.

MINOs [maxcalls] [parno] [parno]...

Causes a Minos error analysis to be performed on the parameters whose numbers **[parno]** are specified. If none are specified, Minos errors are calculated for all variable parameters. Minos errors may be expensive to calculate, but are very reliable since they take account of non-linearities in the problem as well as parameter correlations, and are in general asymmetric. The optional argument **[maxcalls]** specifies the (approximate) maximum number of function calls *per parameter requested,* after which the calculation will be stopped for that parameter.

RELease parno

If **parno** is the number of a previously variable parameter which has been fixed by a command: **FIX parno**, then that parameter will return to variable status. Otherwise a warning message is printed and the command is ignored. Note that this command operates only on parameters which were at one time variable and have been FIXed. It cannot make constant parameters variable; that must be done by redefining the parameter with a PARAMETER command.

REStore [code]

If no **[code]** is specified, this command restores all previously FIXed parameters to variable status. If **[code]=1**, then only the last parameter FIXed is restored to variable status.

SCAn [parno] [numpts] [from] [to]

Scans the value of the user function by varying parameter number **[parno]**, leaving all other parameters fixed at the current value. If **[parno]** is not specified, all variable parameters are scanned in sequence. The number of points **[numpts]** in the scan is 40 by default, and cannot exceed 100. The range of the scan is by default 2 standard deviations on each side of the current best value, but can be specified as from **[from]** to **[to]**. After each scan, if a new minimum is

found, the best parameter values are retained as start values for future scans or minimizations. The curve resulting from each scan is plotted on the output unit in order to show the approximate behaviour of the function. This command is not intended for minimization, but is sometimes useful for debugging the user function or finding a reasonable starting point.

SEEk [maxcalls] [devs]

Causes a Monte Carlo minimization of the function, by choosing random values of the variable parameters, chosen uniformly over a hypercube centered at the current best value. The region size is by default 3 standard deviations on each side, but can be changed by specifying the value of **[devs]**.

SET ERRordef up

Sets the value of **up** (default value= 1.), defining parameter errors. MINUIT defines parameter errors as the change in parameter value required to change the function value by **up**. Normally, for chisquared fits **up=1**, and for negative log likelihood, **up=0.5**.

SET LIMits [parno] [lolim] [uplim]

Allows the user to change the limits on one or all parameters. If no arguments are specified, all limits are removed from all parameters. If **[parno]** alone is specified, limits are removed from parameter **[parno]**. If all arguments are specified, then parameter **[parno]** will be bounded between **[lolim]** and **[uplim]**. Limits can be specified in either order, MINUIT will take the smaller as **[lolim]** and the larger as **[uplim]**. However, if **[lolim]** is equal to **[uplim]**, an error condition results.

SET PARameter parno value

Sets the value of parameter **parno** to **value**. The parameter in question may be variable, fixed, or constant, but must be defined.

SET PRIntout level

Sets the print level, determining how much output MINUIT will produce. The allowed values and their meanings are displayed after a **SHOw PRInt** command. Possible values for **level** are:

-1 No output except from SHOW commands

0 Minimum output (no starting values or intermediate results)

1 Default value, normal output

2 Additional output giving intermediate results.

3 Maximum output, showing progress of minimizations.

SET STRategy level

Sets the strategy to be used in calculating first and second derivatives and in certain minimization methods. In general, low values of **level** mean fewer function calls and high values mean more reliable minimization. Currently allowed values are 0, 1 (default), and 2.

SHOw XXXX

All **SET XXXX** commands have a corresponding **SHOw XXXX** command. In addition, the SHOw commands listed starting here have no corresponding SET command for obvious reasons. The full list of SHOw commands is printed in response to the command **HELP SHOw**.

SHOw CORrelations

Calculates and prints the parameter correlations from the error matrix.

SHOw COVariance

Prints the (external) covariance (error) matrix.

SIMplex [maxcalls] [tolerance]

Performs a function minimization using the simplex method of Nelder and Mead. Minimization terminates either when the function has been called (approximately) **[maxcalls]** times, or when the estimated vertical distance to minimum (`EDM`) is less than **[tolerance]**. The default value of **[tolerance]** is `0.1*UP` (see `SET ERR`).

Appendice B

Uso di ROOT

Questa appendice vuole dare alcuni accenni all'utilizzo diretto delle librerie ROOT in un programma C++: ci preme sottolineare che questa parte non fa parte del programma e serve solo a soddisfare vostre eventuali curiosità aprendo la strada all'utilizzo diretto di un package complesso.

Dopo una breve (e molto parziale) descrizione di due classi base di ROOT: TCanvas e TGraph se ne esemplificherà l' utilizzo in un caso semplice.

B.1 Un breve tour in ROOT

ROOT è un package completo per la rappresentazione dei dati e l'analisi statistica che abbiamo finora utilizzato, indirettamente, come libreria. ROOT è fornito però anche di un'interfaccia grafica, di cui daremo qualche cenno, che ne permette l'utilizzo come strumento a sé.
Per una descrizione completa di ROOT si rimanda al manuale (disponibile al sito *http://root.cern.h*).

B.2 Alcune classi della libreria ROOT

Per ogni classe di ROOT l'include file si chiama con lo stesso nome della classe (ad es. TCanvas.h è l'header file di TCanvas)

B.2.1 TCanvas

TCanvas è la classe che gestisce la finestra grafica (Canvas="tela")

```
TCanvas(const char *name, const char *title, int form)
```

Crea una nuova finestra grafica di dimensioni predefinite. Se `form < 0` la barra del menu non è mostrata.

`form=1` 700x500 (dimensioni in pixel della finestra) a 10,10 (posizione in in pixel dell'estremo superiore sinistro)

`form=2` 500x500 a 20,20

`form=3` 500x500 a 30,30

`form=4` 500x500 a 40,40

`form=5` 500x500 a 50,50

```
void Divide(int nx, int ny)
```

La canvas corrente è divisa in `nx` e `ny` divisioni uguali (pads).

B.2.2 TPad

La pad è in generale un'entità grafica che contiene oggetti grafici (la canvas è un tipo particolare di pad: per questo motivo tutti i metodi di TPad sono anche metodi per TCanvas).
Ogni pad è associata alla lista di puntatori degli oggetti grafici che contiene. Il "disegno" di un oggetto (associato al metodo `Draw` comune a molte classi) non è altro che l'aggiunta del suo puntatore a tale lista (non implica, da solo, un disegno "reale" sulla finestra grafica).
L'oggetto globale `gPad` (membro di TPad) rappresenta il puntatore alla pad (o canvas) corrente.
Non descriviamo qui il creatore di TPad poiché per le applicazioni più semplici è sufficiente poter eseguire operazioni sulla la pad corrente.

```
void cd(int subpadnumber)
```

Permette di accedere alla pad numero `subpadnumber` e modifica `gPad` in modo che punti ad essa.

```
void Update()
```

Aggiorna la pad (o canvas) disegnando (o ri-disegnando) realmente ciò che è etichettato come modificato (ogni volta che un oggetto viene disegnato in una pad automaticamente questa viene considerata modificata)

```
void Modified()
```

Forza a considerare modificata la pad (o canvas) su cui agisce.

B.2.3 TGraph

TGraph è la classe che descrive un oggetto grafico formato da N punti le cui coordinate sono espresse da due vettori: X e Y.

```
TGraph(int n, const double* x, const double * y)
```

Crea l'oggetto grafico

```
void Draw(char *opt)
```

Appende il disegno del grafico alla pad (o canvas) corrente. La stringa `opt` seleziona le seguenti opzioni:

- `P` ogni punto è disegnato con il marker corrente;
- `L` i punti sono uniti con una spezzata;
- `C` i punti sono uniti con una curva continua;
- `A` gli assi sono disegnati (ed i loro estremi automaticamente determinati).

`void SetPoint(int i, double x, double y)`

Assegna le coordinate `x` e `y` al punto `i` del grafico. Se il punto `i` non esiste viene creato.

`void SetMarkerStyle(short int iopt)`

Assegna `iopt` come tipo di marker

`void SetMarkerStyle(short int icol)`

Assegna `icol` come colore del marker

`void SetMarkerSize(float opt)`

Assegna `opt` come dimensione del marker

B.3 Utilizzare ROOT come libreria

Per utilizzare le classi di ROOT occorre includere gli opportuni header file e compilare il file sorgente con il comando (dato su una sola riga)

```
g++ -g -O -Wall -fPIC -D_REENTRANT -I${ROOTSYS}/include
-L ${ROOTSYS}/lib -lCore -lCint -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript
-lMatrix -lMinuit -lPhysics -lm -ldl -lpthread -rdynamic
-o eseguibile sorgente.cpp
```

(è utile ridefinire il comando utilizzando le variabili di ambiente o utilizzare un Makefile)

B.3.1 Un semplice esempio

Il sorgente seguente mostra come realizzare ed aggiornare il grafico di n punti

```
#include "TGraph.h"
#include "TCanvas.h"
#include <iostream>

int main(){

  // Definisco una finestra grafica
  TCanvas *c1 = new TCanvas("tela","Titolo della tela",1);
  // La divido in 4 pads
  c1->Divide(2,2);
  // Mi posiziono nella seconda pad
  c1->cd(2);

  int n=5;
  double x[5]={0.0,0.4,1.2,2.0,2.5};
  double y[5]={1.0,0.6,0.2,3.0,0.5};

  // Creo un oggetto grafico
  TGraph *gr = new TGraph(n,x,y);
  gr->SetMarkerSize(0.5);
```

```
  gr->SetMarkerStyle(20);
  // Appendo l'oggetto alla lista di oggetti della pad
  gr->Draw("AP");
  // Disegno
  c1->Update();

  char c;
  cout <<" Inserisci un carattere qualsiasi per continuare: " << endl;
  cin >> c;

  // Aggiungo un punto
  gr->SetPoint(6,-1.0,4.0);
  // Dichiaro modificata la pad corrente (altrimenti non viene ridisegnata)
  gPad->Modified();
  // Ri-disegno
  c1->Update();

  cout <<" Inserisci un carattere qualsiasi per uscire: " << endl;
  cin >> c;

  delete c1;
  delete gr;

}
```

B.4 Utilizzare ROOT in modo interattivo

Per accedere a ROOT in modo interattivo si da il comando

```
> root
```

sul vostro schermo compare quindi un cursore con il numero incrementale `n` delle operazioni da voi eseguite

```
root [n]
```

Per eseguire le stesse operazioni contenute nel programma dell'esempio C++ si possono inserire, una ad una, le istruzioni (la sintassi dei comandi è C++); è tuttavia più pratico creare un file di comandi, detto "macro", (usualmente con estensione .C) ed eseguirlo con il comando

```
root [n] .x nomefile.C
```

Possiamo ottenere una macro con le stesse funzionalità del programma precedente eliminando dal sorgente gli header files e `main()`.
Ecco, infine, il comando per chiudere una sessione interattiva di ROOT:

```
root [n] .q
```