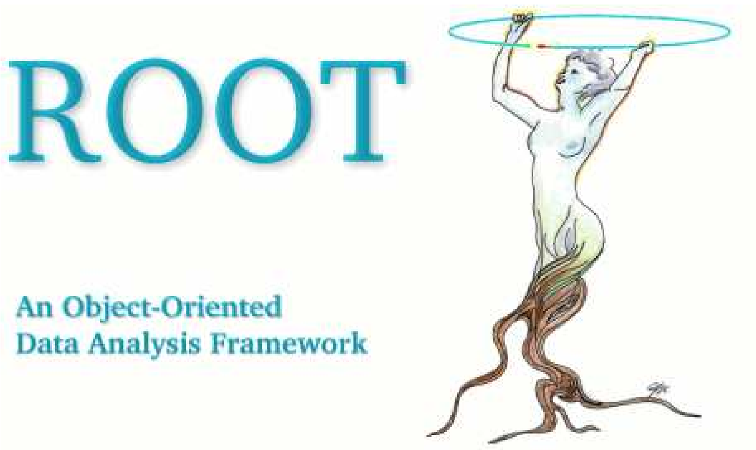
Manuale d' uso di un' interfaccia C a ROOT



Chapter 1: INTRODUZIONE

ROOT è un pacchetto completo di programmi per l'analisi dati (comprende, tra l'altro, funzioni di grafica 2D e 3D, gestione di istogrammi e minimizzazione) e viene distribuito per la maggior parte dei sistemi operativi oggi in uso.
Per utilizzarlo occorre conoscere il C++ e la programmazione ad oggetti.

In questo breve manuale si descrive un' interfaccia C a ROOT ideata per i corsi di Laboratorio di Calcolo del primo anno.
Lo scopo di questa interfaccia è quello di fornire una libreria C che permetta di accedere ad alcune delle funzioni di base di ROOT (selezionate secondo le esigenze di calcolo dei corsi dei primi anni del Corso di Laurea in Fisica).
Questa libreria non vuole essere in nessun senso un sostituto di ROOT ma solo una introduzione ad esso che non richiede la conoscenza del C++.

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 comandi autoesplicativi (e quindi di facile utilizzo), non sarà trattato in questo manuale.

Informazioni generali su ROOT sono disponibili su web all' indirizzo http://root.cern.ch.

1.1 Utilizzare le librerie grafiche

1.1.1 Linux

Per utilizzare nei propri programmi le librerie trattate in questo manuale occorre includere, nel file sorgente, l' header file con i loro prototipi:

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

Nella fase di compilazione occorre includere le librerie attraverso la macro crootlib

```
 gcc sorgente.c -o eseguibile `crootlib`
```

1.1.2 Windows

Chapter 2: MANUALE D'USO

2.1 INIZIALIZZAZIONE E CONTROLLO DELLA FINESTRA GRAFICA

2.1.1 void R_Init()

Apre ed inizializza una finestra grafica.

2.1.2 void R_End()

Chiude la finestra grafica.

2.1.3 void R_Divide(int nx, int ny)

Divide la finestra in **nx** zone lungo x e **ny** zone lungo y (dette pads).

Esempio: la linea di codice

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

divide la finestra in 4 pads uguali.

La pad corrente (la porzione di finestra su cui il programma disegna) 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. Terminato il numero di pads della finestra la finestra viene ripulita (si gira pagina) e si riparte a disegnare dalla prima pad.
Le opzioni che permettono di intervenire tra un cambio pagina e l'altro sono regolate dalla funzione **R_SetWait**.

2.1.4 void R_cd(int n)

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

2.1.5 void R_Update()

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

2.1.6 void R_Interact()

Ferma l' esecuzione del programma e attiva il pannello dei comandi

2.1.7 void R_SetWait(short int iopt)

Imposta il comportamento del programma al cambio pagina.

iopt=0 la pagina viene girata automaticamente;

iopt=1 (default) alla fine di ogni pagina il programma attende un carattere "invio" da parte dell'utente per continuare;

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

2.1.8 void R_SetUpdate(short int iopt)

Gestisce l' aggiornamento dei grafici in finestra.

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

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

2.1.9 void R_PrintPicture(char *file)

Salva l' immagine della pagina corrente sul file di nome **file**. Formati possibili per il file: .ps (consigliato per la stampa), .eps, .gif.

2.2 GRAFICA 2D

2.2.1 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.

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

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

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

Disegna **n** punti con coordinate date dai vettori **x** e **y**. La stringa opt seleziona le seguenti opzioni:

`P` ogni punto è disegnato con il marker corrente (vedi **R_SetMarker...**)

`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 A non è specificato il grafico viene disegnato sugli assi (o sul grafico) pre-esistenti.
Opzione da usare in alternativa a **R_DrawFrame**.

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

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

2.2.5 void R_PlotFunc1D(double fun(double x), double xmin, double xmax, char *opt)

Disegna la funzione **fun** tra gli estremi **xmin** ed **xmax**. La stringa opt può selezionare solo l' opzione "A" (vedi **R_Graph**).

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

Disegna la funzione parametrica **fun** tra gli estremi **xmin** ed **xmax** con i valore degli **npar** parametri assegnati dal vettore **par**. La stringa opt può selezionare solo l' opzione "A" (vedi **R_Graph**).

2.2.7 void R_Text(double x, double y, char *opt)

Scrive, alle coordinate **x** e **y**, la stringa **opt**.

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

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

2.2.9 void R_AxisTitles(char *cx, char *cy)

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

2.3 GRAFICA 3D

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

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

2.3.2 void R_3DGraph(int n, double *x, double *y, double *z, 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 l' opzione "A").

2.3.3 void R_3DAxisTitles(char *cx, char *cy, char *cz)

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

2.4 GRAFICI MODIFICABILI

Le funzioni

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

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

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

sono analoghe a quelle senza il postfisso Obj.
La differenza principale è che creano grafici che sono modificabili nel seguito. Essi sono iden-

tificati 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.

2.5 ISTOGRAMMI

Un' istogramma (unidimensionale) è un diagramma usato per rappresentare la distribuzione di frequenze o di grandezze di un dato fenomeno. Ogni istogramma è associato ad un intervallo della variabile in ascissa diviso in sotto intervalli (canali). L' istogramma è costruito disegnando, per ogni canale, un rettangolo di base pari alla larghezza del canale e di altezza pari alla frequenza o alla grandezza osservata.
Nella gestione degli istogrammi si individuano quindi tre fasi: creazione, riempimento, disegno.

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

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

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

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

2.5.3 void R_FillHist1D(char *hist, float x, float w)

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

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

Riempe l' istogramma **hist** con la coppia di valori **x** e **y** con peso **w**.

2.5.5 void R_PlotHist(char *hist, char *opt)

Disegna l' istogramma **hist** con l' opzione **opt**.
Opzioni generali:

`AXIS`: Draw only axis

`HIST`: Draw only histo contour. (if histo has errors, errors are not drawn)

`SAME`: Superimpose on previous picture in the same pad

`CYL`: Use Cylindrical coordinates

`POL`: Use Polar coordinates

`SPH`: Use Spherical coordinates

`PSR`: Use PseudoRapidity/Phi coordinates

`LEGO`: Draw a lego plot with hidden line removal

`LEGO1`: Draw a lego plot with hidden surface removal

`LEGO2`: Draw a lego plot using colors to show the cell contents

`SURF`: Draw a surface plot with hidden line removal

`SURF1`: Draw a surface plot with hidden surface removal

`SURF2`: Draw a surface plot using colors to show the cell contents

`SURF3`: same as SURF with in addition a contour view drawn on the top

`SURF4`: Draw a surface using Gouraud shading

A queste opzioni è stata aggiunta l' opzione M (Modifica) che aggiorna un' istogramma già disegnato.
Opzioni per istogrammi 1D:

`AH` : Draw histogram, but not the axis labels and tick marks

`B` : Bar chart option

`C` : Draw a smooth Curve througth the histogram bins

`E` : Draw error bars

`E0` : Draw error bars including bins with o contents

`E1` : Draw error bars with perpendicular lines at the edges

`E2` : Draw error bars with rectangles

`E3` : Draw a fill area througth the end points of the vertical error bars

`E4` : Draw a smoothed filled area through the end points of the error bars

`L` : Draw a line througth the bin contents

`P` : Draw current marker at each bin

`*H` : Draw histogram with a * at each bin

Opzioni per istogrammi 2D:

`ARR`: arrow mode. Shows gradient between adjacent cells

`BOX`: a box is drawn for each cell with surface proportional to contents

`COL`: a box is drawn for each cell with a color scale varying with contents

`COLZ`: same as COL. In addition the color palette is also drawn

`CONT`: Draw a contour plot (same as CONT0)

`CONT0`: Draw a contour plot using surface colors to distinguish contours

`CONT1`: Draw a contour plot using line styles to distinguish contours

`CONT2`: Draw a contour plot using the same line style for all contours

`CONT3`: Draw a contour plot using fill area colors

`CONT4`: Draw a contour plot using surface colors (SURF option at theta = 0)

`FB` : With LEGO or SURFACE, suppress the Front-Box

`BB` : With LEGO or SURFACE, suppress the Back-Box

`SCAT`: Draw a scatter-plot (default)

`TEXT`: Draw cell contents as text

2.5.6 void R_ResetHist(char *hist)

Azzera il contenuto dell' istogramma **hist**.

2.5.7 void R_Delete(char *name)

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

2.5.8 double R_GetBinContentHist1D(char *hist,int bin)

Ritorna il contenuto del canale **bin**.

2.5.9 double R_GetBinContentHist2D(char *hist,int binx, int biny)

Ritorna il contenuto del canale identificato dai numeri di canale **binx** e **biny**.

2.5.10 double R_GetEntries(char *hist)

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

2.5.11 int R_GetBinNumber(char *hist, double x, char *axis)

Ritorna il numero del canale dell' asse **axis** (x o y) dell' istogramma **hist** in cui è contenuto il valore **x**.

2.5.12 double R_GetBinNumber(char *hist, int ibin, char *axis)

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

2.5.13 int R_GetNBins(char *hist, char *axis)

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

2.5.14 double R_SetRangeHist(char *hist, int imin, int imax, char *axis)

Seleziona (per le successive rappresentazioni grafiche) un sotto intervallo dell' istogramma **hist** tra il canale **imin** ed il canale **imax** dell' asse **axis**.

2.6 CONFIGURAZIONI GENERALI

Molte delle configurazioni qui descritte riguardano opzioni a cui si può accedere tramite il pannello di controllo (attivabile con **R_SetWait(2)**). È tuttavia utile, nel caso di operazioni ripetute, poter definire l' opzione desiderata una volta per tutte nel programma.
Tutte le funzioni del tipo R_Set* definiscono opzioni che restano valide dalla chiamata alla funzione in poi.

La variabile che identifica un colore sarà nel seguito identificata con icol (colori principali: 1 nero, 2 rosso, 3 verde, 4 blu, 5 giallo, 6 magenta, 7 cyan; per gli altri vedi tabella), il tipo di linea on ilin (1 continua, 2 tratteggiata, 3 punteggiata, 4 tratteggiata-punteggiata).

2.6.1 void R_SetOptLogx(short int iopt)

Se **iopt**=1(0 default) utilizza la scala logaritmica (lineare) per l'asse x.

2.6.2 void R_SetOptLogy(short int iopt)

Se **iopt**=1(0 default) utilizza la scala logaritmica (lineare) per l'asse y.

2.6.3 void R_SetOptLogz(short int iopt)

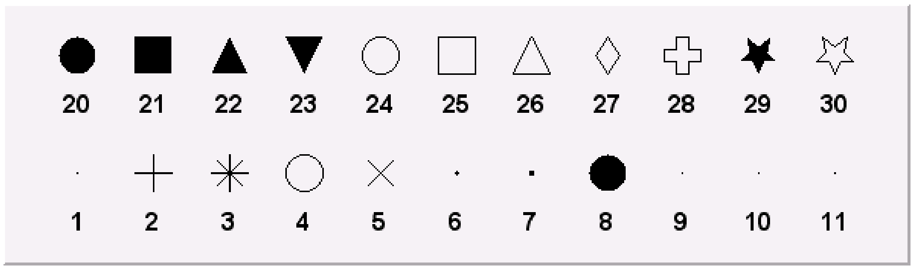
Se **iopt**=1(0 default) utilizza la scala logaritmica (lineare) per l'asse z.

2.6.4 void R_SetGrid(short int iopt)

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

2.6.5 void R_SetMarkerStyle(short int imark)

Definisce il simbolo (marker) associato al disegno di un punto secondo la tavola sottostante (default **imark**=20).



2.6.6 void R_SetMarkerColor(short int icol)

Definisce il colore del "marker"

2.6.7 void R_SetMarkerSize(float fopt)

Definisce la dimensione del marker.

2.6.8 void R_SetLineStyle(short int ilin)

Definisce il tipo di linea da utilizzare.

2.6.9 void R_SetMarkerColor(short int icol)

Definisce il colore della linea.

2.6.10 void R_SetMarkerWidth(short int iopt)

Definisce lo spessore della linea.

2.6.11 void R_SetFillColor(short int icol)

Ridefinisce il colore dello sfondo (bianco default).

2.6.12 void R_SetTitleOffset(float off, char *opt)

Definisce la distanza **off** rispetto agli assi per i title sull' asse **opt** (*opt="x" o "y")

2.6.13 void R_SetHistLineStyle(short int ilin)

Definisce il tipo di linea da utilizzare per gli istogrammi.

2.6.14 void R_SetHistLineColor(short int icol)

Definisce il colore per il bordo dell' istogramma.

2.6.15 void R_SetHistFillStyle(short int iopt)

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

`0` vuoto (default);

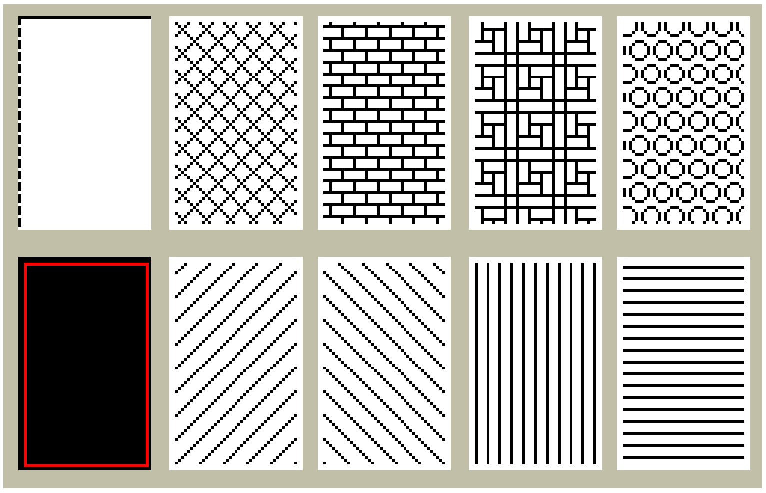
`1001` pieno;

`2001` tratteggiato;

`3001+codice` vedi sotto;

`4000` finestra trasparente;

`da 4001 a 4100` finestra da 100% trasparente a 100% opaca;



2.6.16 void R_SetHistFillColor(short int icol)

Definisce il colore da utilizzare per gli istogrammi.

2.6.17 void R_SetHistLineWidth(short int iopt)

Definisce lo spessore del bordo degli istogrammi.

2.7 MINUIT

MINUIT è un package per la ricerca di minimi di funzioni multiparameteriche.
Le principali applicazioni riguardano analisi statistiche volte a determinare, tramite il metodo dei minimi quadrati, il miglior valore di uno o più parametri e dei loro errori (e delle eventuali correlazioni).

Descriviamo il funzionamento di MINUIT utilizzando l' applicazione più diffusa: la determinazione dei parametri di una funzione teorica a partire dai dati sperimentali.
L' utente fornisce a MINUIT la funzione di χ^2 tra i dati sperimentali ed i valori aspettati (calcolati sulla base di una funzione teorica multiparametrica). Gli unici parametri variabili della funzione χ^2 sono gli stessi parametri della funzione teorica. MINUIT minimizza il χ^2 rispetto a tali parametri o, in altre parole, trova il valore dei parametri che danno il minor valore di χ^2.

2.7.1 R_Minuit(double (*fcn)(int np, double *par), 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 fornita dall' utente (fcn) e dà la miglior stima dei parametri e dei loro errori.

Parametri di ingresso:

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

```
double fcn(int np, double *par){
  ...
  chi2 = ....
```

```
        return chi2;
    }
```

`*mode`Stringa che specifica l'opzione desiderata

`' '` Minimizzazione default;

`'M'` Minuit interattivo.

`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 minimizzazione per i parametri.

`min[]` Vettore con i limiti inferiori per dei parametri.

`max[]` Vettore con i limiti superiori dei parametri.

Parametri di uscita:

`outpar[]` Vettore con il valore fittato dei parametri.

`err[]` Vettore con gli errori sui parametri.

`*chiq` Chisquared of the fit.

Remarks:

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

2.7.2 R_MinuitSimpl(double (*fcn)(int np, double *par), int npar, double *par, double *outpar, double *err, double *chiq)

Funzionamento identico a **R_Minuit()** salvo:

- il modo di minimizzazione "default" è automaticamente selezionato;
- i parametri non sono limitati;
- il nome dei parametri è fisso (p0,p1,p2,...).

2.7.3 Basic concepts of MINUIT

The transformation for parameters with limits.

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

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

so that the internal value $P_{\rm int}$ can take on any value, while the external value $P_{\rm ext}$ 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 $\partial P_{\rm ext}/\partial P_{\rm int} \approx 0$. 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 15) 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 `R_Minuit`, 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`).

Table of Contents