# Nuovo software Field System per l'ottimizzazione di fuoco dell'antenna VLBI di Medicina.

Andrea Orlati[1]

1 - Istituto di Radioastronomia.

**Febbraio 2004**

**IRA 357/04**

# Capitolo 1: Installazione e manuale d'uso.

## 1.1 Introduzione.

Gli spostamenti nelle posizioni di fuoco dell'antenna di Medicina sono due tipi:: traslazioni del ricevitore in fuoco primario e movimenti dello specchio secondario[2]. Nel primo caso si hanno solo due assi possibili: l'asse Y che si muove in direzione Nord/Sud (alto/basso se l'antenna è all'orizzonte anziché allo zenit) e l'asse Z che effettua i movimenti lungo la direzione parabola/cielo. Nel secondo caso, invece, i possibili spostamenti sono la traslazione lungo l'asse Y (Nord/Sud) e lungo l'asse X (Est/Ovest) e la rotazione attorno all'asse X e Y (Z1, Z2, Z3).

Nell'ipotesi del ricevitore K in fuoco primario gli assi hanno i seguenti valori: Y=-151.1mm e Z=-302mm. Se viene usato un ricevitore in fuoco secondario, i valori impostati sugli assi sono: X=176930step, Y=638979step, Z1=-289281step, Z2=-267081step, Z3=-276881step. Questi valori, sono ottimizzati per l'antenna ad un'elevazione di 45°, ma essendo valori fissi, non tengono in considerazione le deformazioni imposte dalla gravità.

L'introduzione del nuovo comando Field System *scu* ha consentito sia di muovere gli assi da schedula, sia di effettuare il tracking col subriflettore spostandolo automaticamente in funzione dell'elevazione. Quest'ultima funzionalità, cioè la necessità di compensare gli effetti della gravità, richiede di determinare per ogni asse una legge che ne descriva la posizione ottimale al variare dell'elevazione. Al pari di *onoff* e *fivept*[1] il Field System aveva già previsto, ma non ancora implementato, un programma che servisse a determinare il modello di deformazione del fuoco attraverso una campagna di osservazioni astronomiche. La tecnica effettua una seria di letture di total power spostando di volta in volta l'asse da esaminare in un intorno della posizione di partenza dell'asse stesso. I vari campioni ottenuti vengono poi interpolati secondo un modello specifico per quell'asse. Dal modello si ricava la posizione dell'asse in cui è possibile ottenere il valore massimo di total power. I passi vengono iterati su tutti gli assi su cui si vuole lavorare. Di seguito sarà descritta l'implementazione realizzata a Medicina di p*eakf*.

## 1.2 Compilazione ed installazione.

Il nuovo programma *peakf* dipende completamente dal Field System, perciò l'unico vero requisito per installarlo con successo su una macchina è che su questa sia presente un Field System un versione 9.5.15 o successivo.

E' necessario avere una directory, che chiameremo cartella d'installazione, nella quale andranno copiati i file che sono necessari alla compilazione del programma. Da questa cartella occorre poter accedere a quella del Field System secondo il percorso *../../fs/*; il nome invece non ha importanza (es */usr2/st/peakf/*). I file per la compilazione del programma sono elencati nella Tabella 1.

**Tabella 1- Elenco dei file necessari per l'installazione**

| | FILE | PERCORSO | DESCRIZIONE MODULO |
|---|---|---|---|
| Header | Common.h | InstallDir | Dichiarazione simboli comuni ai moduli |

[1] Onoff e Fivept sono due programmi del Field System che consentono rispettivamente di ricavare i parametri di guadagno dell'antenna e di ottimizzarne del puntamento.

| | | | |
|---|---|---|---|
| | Const.h | InstallDir | Definizione delle costanti |
| | Export.h | InstallDir | Definizione dei simboli da esportare, ad esempio in STQKR[2] |
| | Peakf.h | InstallDir | Header generico |
| | Private.h | InstallDir | Definizione dei simboli privati |
| | Types.h | InstallDir | Definizione dei tipi di dato |
| | Fitlib.h | InstallDir/fitlib | Simboli esportati dalla libreria di fitting |
| | Nrutil.h | InstallDir/fitlib | Definizione dei simboli utilizzati dalla libreria di fitting |
| Source | Display_fit.c | InstallDir | Stampa i parametri trovati dalla routine di Fit |
| | Display_offset.c | InstallDir | Stampa gli offset trovati |
| | Display_sample.c | InstallDir | Stampa i valori letti da una singola campionatura |
| | Do_scan.c | InstallDir | Routine principale che esegue uno scan completo |
| | Fgauss.c | InstallDir | Funzione di Gauss |
| | Fparabola.c | InstallDir | Funzione parametrica che descrive una parabola |
| | Get_device.c | InstallDir | Ritorna i parametri di un *device* quali la frequenza o la catena IF cui è connesso |
| | Get_gain_par.c | InstallDir | Ritorna i parametri di guadagno di un *device* quali Il valore della marca, il DPFU. |
| | Get_samples.c | InstallDir | Ritorna il risultato di N campionamenti su uno specificato *device* |
| | Go_off.c | InstallDir | Sposta l'antenna di un offset in azimuth e/o elevazione |
| | Go_scu.c | InstallDir | Sposta un asse del subriflettore. |
| | Local.c | InstallDir | Ritorna azimuth, elevazione ed offset attuali dell'antenna |
| | Log.c | InstallDir | Mette nel log file del Field System un buffer di caratteri |
| | Mat.c | InstallDir | Manda un buffer a MATCN[3] |
| | Onsor.c | InstallDir | Attende che il telescopio sia arrivato sulla sorgente comandata |
| | Peakf.c | InstallDir | Punto d'ingresso del programma |
| | Peakf_record.c | InstallDir | Stampa il record iniziale di Peakf |
| | Scmds.c | InstallDir | Lancia un commando SNAP sul Field System. |
| | Source_record.c | InstallDir | Stampa il record sulla sorgente puntata |

[2] Programma che implementa i comandi di stazione.
[3] Programma Field System che comanda la MAT sul MarkIV.

| | | | |
|---|---|---|---|
| | Scu_record.c | InstallDir | Stampa il record con le informazioni sul subriflettore prima di partire |
| | Tpi_get.c | InstallDir | Ritorna una misura di TPI |
| | Tpi_request.c | InstallDir | Effettua una richiesta per una misura di TPI di uno specificato *device* |
| | Tzero.c | InstallDir | Effettua una misura di TPI di uno specificato *device* quando tutti gli attenuatori sono impostati |
| | Covsrt.c | InstallDir/fitlib | Modulo utilizzato per la routine di fitting col metodo di Levenberg-Marquardt[1] |
| . | Gaussj.c | InstallDir/fitlib | Modulo utilizzato per la routine di fitting col metodo di Levenberg-Marquardt[1] |
| | Mrqcof.c | Installdir/fitlib | Modulo utilizzato per la routine di fitting col metodo di Levenberg-Marquardt[1] |
| | Mrqmin.c | InstallDir/fitlib | Modulo utilizzato per la routine di fitting col metodo di Levenberg-Marquardt[1] |
| | Nrutil.c | InstallDir/fitlib | Modulo utilizzato per la routine di fitting col metodo di Levenberg-Marquardt[1] |
| | Nonlinearfit.c | InstallDir/fitlib | Viene utilizzata per approssimare una sequenza di dati con funzioni non lineari |
| | Peakflib.c | InstallDir/lib | Libreria con funzioni generiche |
| Others | Makefile | InstallDir | File di configurazione per la compilazione |

Nella cartella */usr2/control* vanno modificati i file di configurazione *stcmd.ctl*, *stpgm.ctl* e *sterr.ctl*. Il primo file registra i comandi di stazione che il Field System riconosce; va aggiunta la riga sottostante:

```
peakf       stq 1001 01 FFFFFF
```

Questa riga informa il Field System che il comando *peakf* è la routine 10, task 01 di STQKR [4] e che necessita della configurazione hardware FFFFFF (nessun requisito particolare) per essere eseguita.
Il file di configurazione *stpgm.ctl* informa il Field System su quali programmi di stazione devono andare in esecuzione al momento in cui esso è lanciato; la modifica da apportare è la seguente:

```
peakf n peakf &
```

Infine, nel file *sterr.ctl* sono elencati tutti i codici e i messaggi d'errore che sono riportati in caso di malfunzionamenti del software di stazione. Anche in questo caso è necessario aggiungere le seguenti righe:

```
""
PF -1001
Peakf is already running
""
PF -1002
Break detected in Peakf
""
PF -1003
Matcn failure
""
```

[4] Programma che implementa tutti i comandi di stazione.

```
PF -1004
Antcn failure
""
PF -1005
Source not reached in allotted time
""
PF -1006
Parameter is wrong or out of range
""
PF -1007
No scu link
""
PF -1008
Failure in Scu
""
PF -1009
Program internal error
""
PF -1010
Antenna failure
```

Nell'area di memoria riservata al software di stazione è necessario riservare lo spazio per le variabili che p*eakf* utilizza; all'inizio del file */usr2/st/include/stcom.h* va aggiunto:

```
#include "../peakf/export.h"
```

Avendo cura che il percorso specificato sia quello che porta al file *export.h* dentro la cartella d'istallazione. In un punto qualsiasi dentro la *struct stcom* deve comparire la riga:

```
peakf_cmn peakfcommon;
```

Importante ricordare che, affinché queste modifiche prendano corso e per evitare che danneggino gli altri programmi, è necessario ricompilare tutto il software di stazione e riavviare la macchina.

Nella cartella */usr2/st/stqkr* ci sono i file che compongono il programma STQKR. In questa cartella è necessario copiare il file *lpeakf.c* (listato all'appendice B) e modificare il file *stqkr.c*. Nella sezione degli include va aggiunta un'indicazione al file *peakf.h* che si trova nella cartella d'installazione e al file *errno.h*:

```
#include "../peakf/peakf.h"
#include "errno.h"
```

In fondo all'elenco dei moduli inclusi in STQKR deve essere accodato un'indicazione al file *lpeakf.c*:

```
#include "azelrate.c"
#include "radecrate.c"
#include "scu.c"
…
…
…
// PEKAF
#include "lpeakf.c"
```

Infine all'interno del ciclo principale va modificato lo switch come segue:

```
  switch (isub) {
      case 1:
        break;
      case 2:       /* AZELRATE command handling */
```

```
            ierr=azelrate(&command,ip,isub,itask);
            if (ierr) goto error;
          break;
        case 3:       /* radecrate command handling */
            ierr=radecrate(&command,ip,isub,itask);
            if (ierr) goto error;
          break;
…
…
…
      case 10: {
            lpeakf(&command,ip,isub,itask);
            break;
      }
…
…
```

Alla fine di queste modifiche STQKR va ricompilato.

L'ultima fase è quella della compilazione del programma *peakf*. Nella Tabella 1 appaiono anche i file che compongono le due librerie usate da *peakf*, situate rispettivamente nelle cartelle *lib* e *fitlib*. Prima di procedere alla compilazione del programma è necessario produrre queste due librerie che hanno nome rispettivamente *peakflib.a* e *fitlib.a*.

Sia il programma vero e proprio, sia le librerie sono munite dei loro *makefile* (ognuno dentro la propria cartella). Digitando dentro una shell il comando *make* la compilazione dei moduli avviene automaticamente; il file eseguibile p*eakf* viene copiato dentro la cartella */usr2/st/bin* che contiene tutti i programmi di stazione che vengono utilizzati dal Field System. E' possibile eliminare i file temporanei, creati durante la compilazione, attraverso l'utilizzo del comando *make clean.*

Per il corretto funzionamento, *peakf* necessita di un modo per accendere e spegnere la marca di calibrazione, a tal fine esso si aspetta di trovare nella libreria di stazione (*station.prc*) due procedure chiamate *calonpf* e *caloffpf*[5]. Le due procedure hanno rispettivamente il seguente contenuto:

```
calon
sy=go peakf &
```

e

```
caloff
sy=go peakf &
```

Giunti a questo punto, se non si sono riscontrati errori, è possibile lanciare il Field System come di norma. Con il comando *ps ax | grep peakf* si può verificare se *peakf* è andato effettivamente in esecuzione.

## 1.3 Manuale d'uso.

Il Field System utilizza il linguaggio SNAP (Standard Notation for Astronomy Procedures) per il controllo degli esperimenti. Una descrizione approfondita, regole sintattiche e comandi, può essere trovata in [3]. Il programma *peakf*, in quanto parte del Field System, per essere configurato e

[5] Il modo più semplice per aggiungere le procedure è utilizzare il programma *pfmed* [3].

lanciato necessita di un comando SNAP che di seguito verrà illustrato nel dettaglio. Saranno inoltre descritti i record d'output del programma, l'integrazione col programma di schedulazione e i messaggi degli eventuali errori.

### 1.3.1 Utilizzo e comando SNAP.

Il comando per il lancio e la configurazione è un comando SNAP e come tale obbedisce alle sue regole sintattiche, comprese quelle che riguardano l'uso di caratteri speciali quali la '*' e '?'. Esiste una modalità di configurazione dei parametri (set), una di lettura (query) e una per il lancio del programma.

Sintassi: *peakf=Axis,Repeat,Stps,Off,Integ,Dev*

Risposta: *peakf/Axis,Repeat,Stps,Off,Integ,Dev*

Se uno o più parametri vengono impostati con valori non leciti o fuori dai loro range divariazione viene lanciato l'errore (–1006) e i valori dei parametri rimangono immutati. Il significato dei parametri (tutti impostabili) è il seguente:

a. *axis*: Assi su cui ottenere l'ottimizzazione di fuoco. Può essere uno od una combinazione dei seguenti nomi di asse: x,y,z,z1,z2,z3. L'ordine con cui appaiono influenza l'ordine con cui i vari assi vengono massimizzati. A seconda del modo in cui è impostato il subriflettore, secondario o primario, alcune configurazioni non sono legali. L'elenco delle configurazioni accettate è riportato in Tabella 2. Z è il valore di default.

b. *Repeat*: Numero di ripetizioni, cioè il numero di volte che l'algoritmo viene ripetuto. Il default è 2.

c. *Stps*: Numero di punti misurati su ciascun asse. Deve essere dispari per avere un punto centrale; se il numero di punti immesso è pari esso viene incrementato di 1. Il valore di default è 11.

d. *Off*: Massimo offset in millimetri che ogni asse può percorre. Il valore di default è 24, ma può variare a seconda e della freqeunza di lavoro.

e. *Integ*: Numero di campioni che si acquisiscono dal device per ogni punto da misurare; siccome viene fatta una misura ogni secondo esso rappresenta il tempo d'integrazione in secondi. Il valore preimpostato è 5;

f. *Dev*: Codice mnemonico del dispositivo usato per fare la misura. Tutti i dispositivi presenti sul terminale Mark4 sono utilizzabili: *v1, v2, ..., vf* per i video converter, *i1, i2, i3* per le tre catene IF. Il valore di default è *ve*.

La '*' al posto di un valore conserva il valore precedentemente impostato, un omissis, ovvero un valore non impostato implica il settaggio del valore di default per quel parametro; un comando *peakf=* con nessun valore specificato implica il settaggio di tutti i valori di default. La modalità di query, ovvero di richiesta dei valori dei parametri impostati viene effettuata con *peakf=?*.

L'avvio del programma viene effettuato col comando *peakf* senza parametri. La misura inizia col primo asse tra quelli impostati, partendo dalla posizione in cui si trova attualmente[6]. Se la misura ha successo l'asse viene lasciato nel punto su cui è stato trovato il massimo. Se subentra una condizione d'errore o l'esecuzione viene interrotta dall'utente (comando *sy=brk peakf*) l'asse viene riportato all'ultima posizione corretta.

[6] In questo modo viene data la possibilità di partire da posizioni ottimizzate con precedenti sessioni di *peakf*.

Per il buon funzionamento sono necessari alcuni accorgimenti:

a. Durante l'esecuzione è preferibile non lanciare schedule o procedure SNAP che impiegano troppo tempo per essere eseguite.

b. Il dispositivo utilizzato deve essere propriamente configurato. Particolare attenzione deve essere posta ai livelli di potenza e alla stabilità del segnale che in prima approssimazione significa evitare il più possibile interferenze in banda.

c. Gli appropriati comandi *lo=* e *patch=* devono essere usati per impostare frequenza dell'oscillatore locale, polarizzazione e a quale catena IF il dispositivo è collegato. In ogni caso il Field System deve essere in grado di trovare dentro un file *.rgx* l'appropriato valore della marca di rumore [4].

**Tabella 2 – Elenco delle configurazioni legali degl'assi**

| **Primary mode** | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| y | z | zy | yz | | | | | | | | | | | | | | |
| **Secondary mode** | | | | | | | | | | | | | | | | | |
| x | y | z | xy | yx | xz | zx | yz | zy | xyz | xzy | yxz | yzx | zxy | zyx | z1 | z2 | z3 |

## 1.3.2 Record di output.

Così come *Onoff* e *Fivept* anche *Peakf*, durante le iterazioni, raccoglie i dati nel file di log attualmente aperto. I dati sono scritti sotto forma di record, così da semplificarne poi l'analisi e la lettura; p*eakf* ne produce 9 tipi: *source, parameters, Tsys, scu, axis, gaussfit, parabolicfit, err, offset*.

Il record *source* contiene le informazioni sulla sorgente attualmente puntata e che verrà utilizzata per la misura. Vengono riportati nome della sorgente e coordinate, vale a dire ascensione retta, declinazione ed epoca.

```
2003.084.13:36:39.15#peakf#source  cygnusa     19:59:28.3  +40:44:01  2000.0
```

Dopo il record *source* il programma mette a log il record *parameters* che riassume il valore dei parametri di configurazione, così come descritto nel Paragrafo 1.3.1.

```
2003.084.13:36:39.15#peakf#parameters  xyz   1   9  30.00   5  v3
```

Nell'esempio la misura viene fatta in sequenza sugli assi X, Y, Z; viene ripetuta una sola volta ed è fatta campionando (5 secondi d'integrazione) 9 punti distribuiti su una corsa di 30 mm. Il dispositivo utilizzato è il video converter 3.

I record *source* e *parameters* sono annotati soltanto alla partenza. I record *Tsys, scu, axis, gaussfit, parabolicfit, err* sono iterati per ogni asse e per tutte le volte richieste dal parametro *Repeat*.

Prima di iniziare viene eseguito il calcolo della temperatura di sistema (fuori sorgente). Nel record *Tsys* sono riportati azimuth ed elevazione con i relativi offset impostati sull'antenna, la temperatura di sistema (in Kelvin) e il relativo errore sulla misura.

```
2003.084.13:37:10.83#peakf#tsys 305.9  22.2  0.0033  0.0018  73.885 0.037
```

Il record *scu* riporta lo stato dell'asse del subriflettore prima dell'inizio della misura; esso rappresenta anche l'ultimo stato corretto in cui l'asse sarà riportato se dovesse insorgere una condizione d'errore. I campi del record sono: nome dell'asse, posizione in millimetri, offset in millimetri. Il nome dell'asse è composto dal suo mnemonico (x, y, z, …) più una S se è relativo al

subriflettore o una P per fuoco primario. Nel caso dell'asse zS sono riportati posizione ed offset dei tre assi z1, z2, z3 rispettivamente.

```
2003.084.13:37:19.91#peakf#scu   xS  0.000  0.000
```

Il record *axis* è il riepilogo della singola misura, viene ripetuto il numero di volte previsto dal parametro *Stps*. I sei campi in esso contemplati sono:

1) Nome dell'asse su cui si sta lavorando.
2) Contatore d'iterazione, da 0..*Stps*-1.
3) Indicatore di tempo, vale a dire secondi trascorsi dall'inizio della misura.
4) Offset in millimetri,relativo alla posizione iniziale, applicato all'asse per quella misura.
5) Risultato della misura, esso è la media dei campioni ottenuti durante il tempo d'integrazione stabilito da *Integ*, sottratto il valore della temperatura di sistema calcolato in precedenza. In pratica è quanto intensa viene vista la sorgente (in K) con quella determinata configurazione del telescopio.
6) Errore sulla misura precedente.

```
2003.084.13:37:35.43#peakf#axis xS  0  13.4  -13.333  54.915  0.02886
2003.084.13:37:49.57#peakf#axis xS  1  27.6  -10.000  55.127  0.01378
2003.084.13:38:03.72#peakf#axis xS  2  41.7  -6.667  55.240  0.03180
2003.084.13:38:17.86#peakf#axis xS  3  55.8  -3.333  55.178  0.04210
2003.084.13:38:32.01#peakf#axis xS  4  70.0  0.000  55.135  0.03077
2003.084.13:38:46.15#peakf#axis xS  5  84.1  3.333  54.876  0.07170
2003.084.13:39:00.39#peakf#axis xS  6  98.4  6.667  54.808  0.04622
2003.084.13:39:14.54#peakf#axis xS  7  112.5  10.000  54.601  0.02644
2003.084.13:39:28.68#peakf#axis xS  8  126.7  13.333  54.365  0.05717
```

Una volta acquisiti tutti i campioni, *peakf* esegue un'interpolazione per trovare il punto di massimo; il tipo d'interpolazione usato dipende dall'asse esaminato e può essere gaussiano o parabolico; secondo il caso viene generato un record *gaussft* o *parabolicfit*. Un esempio di fit gaussiano è

```
2003.084.13:58:37.24#peakf#gaussfit zS  0.349  51.902  28.446  22.052  0.338  10
2003.084.13:58:37.24#peakf#err zS  0.667  9.910  9.864  9.878  0.034  1.347
```

dove vengono riportati in sequenza, nome dell'asse e parametri della curva gaussiana; cioè il punto di massimo, sigma, l'ampiezza, termine noto e coefficiente angolare della retta (vedi Paragrafo 2.4.4), l'ultimo valore è il numero d'iterazioni impiegate per riuscire ad interpolare i punti. Se la matrice generata dall'algoritmo d'interpolazione (*Levenberg-Marquardt*) è singolare, oppure il metodo non converge questo campo avrà un –1 o un –2 rispettivamente. Il record *parabolicfit*, risultato del tentativo di interpolare i punti con una curva di secondo grado, è il seguente:

```
2003.084.13:53:24.71#peakf#parabolicfit xS  4.050  0.020  54.778  5
2003.084.13:53:24.71#peakf#err xS  0.035  0.002  0.022  20.483
```

i campi sono nell'ordine: nome dell'asse, punto di massimo e coefficienti b e c dell'equazione della parabola. L'ultimo valore ha lo stesso significato di quello appena descritto.

Gli ultimi due record descritti sono sempre seguiti dal record *err,* che riporta in corrispondenza di ogni campo la relativa stima dell'errore. L'ultimo valore è l'$\chi^2$ per il test della bontà dell'interpolazione (vedi definizione 3).

Alla fine delle iterazioni previste, per ogni asse analizzato viene riportato un record *offset* che riassume i risultati:

```
2003.084.13:58:39.41#peakf#offset xS 20.248  4.050  -0.345  -0.350  5
2003.084.13:58:39.41#peakf#offset yS 19.836  0.000  0.000  0.000  -3
```

```
2003.084.13:58:39.41#peakf#offset zS 19.395  0.349  -1.221  -1.221  -1.221  -
1.221  -1.221  -1.221  10
```

I campi del record sono: il nome dell'asse, l'elevazione a cui è stata fatta la misura, e il punto in cui è stato trovato il massimo; in particolare di questo punto viene fornito l'offset trovato rispetto alla posizione di partenza, il nuovo offset (somma del precedente più quello trovato) e la posizione assoluta dell'asse. Ancora una volta, nel caso che dell'asse zS vengono aggiunti gli offset per gli assi z1, z2, z3. L'ultimo numero fornito riassume il comportamento della routine di fitting; se positivo è il numero d'iterazioni impiegate per convergere, se –1 o –2 la routine non ha dato risultato, se –3 la routine ha fornito un risultato fuori dai range definiti dal parametro *Off*. In ogni caso un valore negativo in questo campo significa che la misura non ha avuto successo e va scartata.

### 1.3.3 Integrazione con ACQUIRE.

Le calibrazioni di puntamento e di guadagno fatte con l'ausilio del Field System sono solitamente, per quanto riguarda l'acquisizione dati, condotte tramite il programma *Acquir*[3]. *Peakf* è stato studiato in modo da poter essere utilizzato, come *Onoff* e *Fivept*, anche tramite questo programma di schedulazione. Per la parte di programmazione e configurazione di A*cquir* e del telescopio in generale, nonché per una spiegazione della nomenclatura e delle procedure di calibrazione si rimanda a [3] e [4]. Di seguito verrà descritta solo la modifica da apportare al file di configurazione di *acquir* per l'uso di *peakf* da scheduler. Uno stralcio di un tipico file di configurazione (file *ctl*), è riportato sotto:

```
3C84        031629.54  411951.7 1950 PREP    -1   0  15   0 POSTP    -2
3C161       062443.2  -055112.  1950 PREP    -1  20   0   0 POSTP    -2
2134P004    213405.23  002825.0 1950 PREP    -1 20   0  10 POSTP    -2
```

In corrispondenza a ciascuna sorgente è possibile riservare ad *Onoff, Fivept e Peakf* una certa quantità di tempo. Il settimo, ottavo e nono campo di ciascuna riga (sorgente) sono il tempo in minuti concesso ai tre programmi. Nell'esempio del riquadro, quando il telescopio sarà sulla sorgente 213AP004 a *peakf* saranno assegnati 10 minuti per completare le sue misure; se queste non saranno ultimate entro il tempo assegnato l'ottimizzazione di fuoco sarà interrotta.

### 1.3.4 Codici d'errore.

Nella Tabella 3 viene data la lista degli errori che si possono incontrare ed una spiegazione delle possibili cause. L'elenco fornito è quello degli errori che *peakf* è in grado di riconoscere e gestire in modo appropriato.

**Tabella 3 – Errori riconosciuti e gestiti da *Peakf*.**

| Cod. | Messaggio | Causa | Soluzione |
|---|---|---|---|
| -1001 | Peakf is already running | *Peakf* sta già eseguendo una misura e si sta tentando di lanciarne un'altra prima che questa sia ultimata. | Attendere la fine della misura oppure interromperla con *sy=brk peakf* prima di lanciarne una nuova. |
| -1002 | Break detected in Peakf | Errore che viene generato quando *peakf* è interrotto manualmente con *sy=brk peakf*. Un'interruzione manuale, data senza che Peakf sia in esecuzione, fa si che il | Rilanciare *peakf*. |

| | | | |
|---|---|---|---|
| | | Field System ricordi il comando e interrompa *peakf* la prima volta che è lanciato. | |
| -1003 | Matcn failure | Quest'errore nasce quando *Matcn* non risponde o ritorna a sua volta una condizione d'errore. *Matcn* è il programma Field System che gestisce il bus MAT al quale sono collegati i dispositivi del MarkIV. *Peakf* usa questo sistema per effettuare le letture di total power. | Provare una lettura di total power da Field System (*vc01*, *vc02*, ….) oppure controllare, sul dispositivo utilizzato, il led che indica se il collegamento MAT è funzionante. |
| -1004 | Antcn failure | Per muovere l'antenna o per controllare che essa sia arrivata sulla sorgente *peakf* fa delle richieste ad *Antcn*. Se esso non risponde o risponde con un errore Peakf esce con questo codice. | Controllare che *Antcn* sia "vivo" ed eventualmente riavviare il Field System. |
| -1005 | Source non reached in allotted time | *Peakf* attende per un determinato tempo (450 secondi) che l'antenna raggiunga una sorgente, ovvero che sia asserito il flag di *on-source*. Se ciò non avviene allora *peakf* esce con questo codice d'errore. | Verificare di aver comandato una sorgente e che questa sia visibile, ovvero sopra il limite impostato nel file di controllo. Controllare che gli azionamenti dell'antenna siano funzionanti. |
| -1006 | Parameter is wrong or out of range | L'errore si verifica quando, durante la configurazione di *peakf*, i parametri non vengono correttamente valorizzati o indicati. | Verificare che parametri siano usati nel corretto numero e ordine e che il loro valore sia consistente. |
| -1007 | No scu link | L'errore viene segnalato qualora non vi sia collegamento tra il Field System e il PC incaricato delle movimentazioni del subriflettore. | Verificare che il PC (VLBISERV) sia acceso e che il programma incaricato del controllo del subriflettore sia in esecuzione. Accertarsi che il collegamento ethernet fra Field System e PC sia funzionante. Provare a ristabilire il link col comando *scu=connect*. |
| -1008 | Failure in Scu | Questa condizione accade quando il programma *scu* che controlla il subriflettore (lato Field System) segnala un errore dopo che una nuova posizione è stata comandata. | Controllare che le movimentazioni del subriflettore siano funzionanti. Accertarsi che prima di partire il subriflettore sia in una posizione consistente: primario |

| | | | |
|---|---|---|---|
| | | | o secondario; ricevitore CCC o KKP o LHP ecc. impostato. |
| -1009 | Program internal error | Si verifica quando ci sono problemi interni al programma o al Field System. Possono essersi verificati problemi di allocazione di memoria oppure non è stato possibile capire a quale catena IF è collegato il dispositivo utilizzato per la misura. | Accertarsi di avere dato i corretti comandi *patch=* e *lo=*. Riprovare a lanciare la misura. Se il problema persiste provare a riavviare il Field System. |
| -1010 | Antenna failure | Nonostante i comandi siano stati inoltrati correttamente e correttamente interpretati da *Antcn*, l'antenna non si muove. | Controllare che il servo sistema dell'antenna siano funzionante. |

# Capitolo 2: Note sull'implementazione

## 2.1 Introduzione.

*Peakf* consta essenzialmente di due parti: il modulo aggiuntivo di *stqkr* che implementa il comando SNAP (vedi Appendice B) e il programma vero e proprio che esegue le misure. In particolare il programma principale si appoggia su due librerie; una che implementa le funzioni d'uso generale l'altra che contiene la routine necessaria per l'interpolazione. Di seguito verrà descritto il funzionamento di tutto il pacchetto senza soffermarsi sui dettagli implementativi, per i quali si rimanda alle appendici dove è riportato tutto il codice sorgente.

## 2.2 Note sul Field System.

Il Field System è un insieme di programmi che cooperano per ottenere le funzionalità di questo software che è alla base di tutte le osservazioni VLBI. I programmi Field System sono divisi in due gruppi, quelli on-line e quelli off-line. I programmi appartenenti alla prima categoria sono elencati nel file */usr2/fs/control/fspgm.ctl*; essi sono lanciati e generalmente operano in background (incom escluso che si occupa d'inizializzare l'area di memoria comune) all'avvio del Field System stesso. Il programma *fs* ha come unico compito quello di eseguirli e di terminarli qualora uno di loro finisca in modo anormale. Se esistono esigenze particolari, ogni stazione è libera di aggiungere a quest'elenco nuovi programmi; sarà sufficiente aggiornare il file */usr2/control/stpgm.ctl* perché fs si prenda in carico la loro esecuzione. *Peakf* è un programma di stazione on-line. I programmi off-line come pfmed, setcl, ecc sono generalmente d'utilità generale, vengono avviati come normali programmi Linux il cui funzionamento non influenza il resto del sistema.

Come già detto il Field System capisce ed interpreta il linguaggio SNAP. Questa notazione prevede due tipi d'istruzioni: i comandi immediati ed i comandi a tempo. I secondi consentono una sincronizzazione d'eventi che devono succedersi oppure di eseguire periodicamente determinate operazioni. Questi comandi possono essere invocati singolarmente oppure raggruppati in una sorta di macro-comandi, dette procedure. Le procedure sono comunque eseguite come operazioni "atomiche".

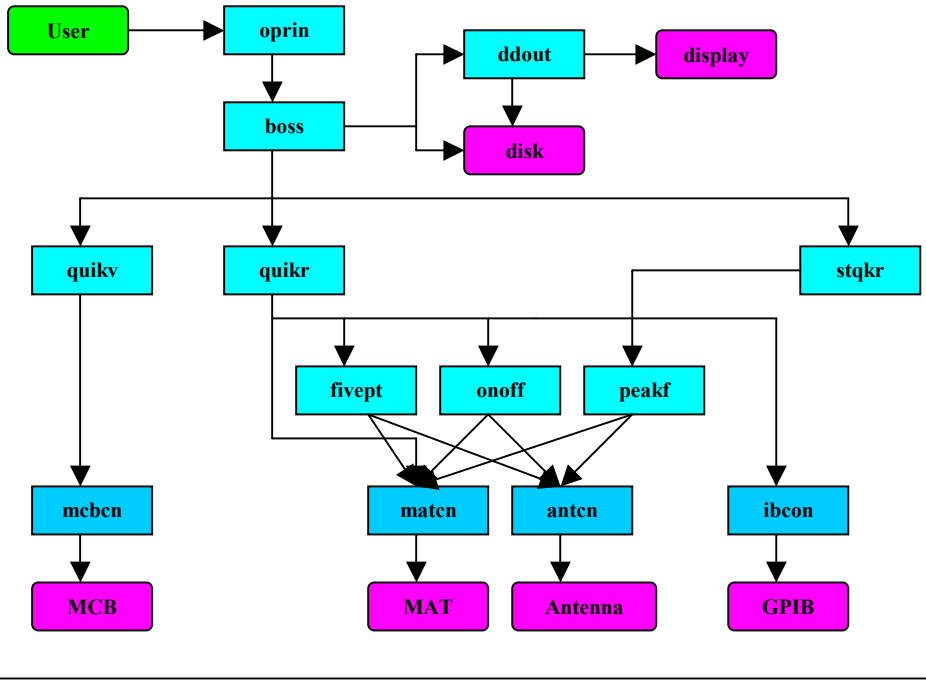
Il Field System fornisce due flussi di input logicamente separati: l'input da operatore e quella schedula. L'input da operatore avviene attraverso *oprin*, mentre l'input da schedula è gestito direttamente da *boss*; che è l'interprete dei comandi vero e proprio. Una volta interpretato un comando, *boss* ne calcola la tempistica ed eventualmente delega l'esecuzione al modulo adibito. Per l'esecuzione *boss* schedula 3 moduli: *quikr*, *quikv* e *stqkr*. Il primo implementa tutti i comandi MarkIII e MarkIV, interagisce con l'hardware attraverso i programmi controllori (*matcn*, *ibcon*, *antcn*). Q*uikv* è il gemello di *quikr* ma è stato pensato per eseguire i comandi specifici del rack VLBA. Infine *stqkr* è il programma usato per implementare le funzioni specifiche di stazione ed estendere, quindi, il set di comandi SNAP senza fare cambiamenti invasivi ai sorgenti del Field System. Il programma *ddout* s'incarica di aggiornare il log file nel quale annota tutti gli eventi che si sono verificati. Quanto detto sin d'ora è schematizzato brevemente nella Figura 1.

Ci sono quattro aree necessarie al funzionamento di tutti i programmi che costituiscono il Field System, questi meccanismi sono implementati principalmente da funzioni (vedi Appendice A) contenute in tre librerie distribuite insieme alle release del Field System:

1. *flib*, fornisce le funzioni da chiamare in ambiente Fortran.

2. *clib*, contiene le funzioni C che realizzano le funzionalità vere e proprie
3. *fclib*, le routine esportate da questa libreria fanno da intermediarie verso le funzioni della *clib*, che sono quelle che implementano le funzionalità vere e proprie. Le routine *flib* si appoggiano a queste per funzionare.

**Figura 1 – Schema a blocchi dell'architettura Field System.**




Le quattro aree citate sopra sono:

1. Memoria condivisa. All'avvio della macchina, va in esecuzione il programma *fsalloc* che si occupa di riservare il segmento che ospiterà l'area condivisa del Field System. Quest'area di memoria che è utilizzata per mantenere traccia dello stato del sistema sotto diversi aspetti, ha una regione usata per memorizzare le variabili Fortran e un'altra riservata all'ambiente C. Ciascun programma C che richiede accesso all'area condivisa deve averla preventivamente collegata al suo spazio indirizzi; questo meccanismo si realizza semplicemente includendo nel sorgente i file *fscom.h* e *shm_addr.h* e utilizzando la funzione *setup_ids()*. Anche il software di stazione ha una propria area dati condivisa allocata da *stalloc* al boot; essa è utilizzata includendo i file *stcom.h* e *stm_addr.h* e tramite la funzione *setup_st()*.
2. Comunicazione inter-processo. Essa avviene tramite il meccanismo delle classi. Le classi forniscono un insieme di code di messaggi FIFO (First In First Out) che possono essere usate per trasmettere informazioni tra vari processi. Ciascuna coda è identificata univocamente con un numero o numero della classe. I numeri di classe possono essere allocati o deallocati, manualmente o automaticamente; l'allocazione automatica avviene quando si cerca di allocare un numero di classe zero. Quando un numero di classe è stato allocato un programma può attendere l'arrivo di un messaggio o spedirne uno attraverso quella classe.

3. Scheduling di programmi. Il meccanismo di schedulazione è così implementato: ciascun programma è lanciato come un normale programma Linux, immediatamente dopo si sospende in attesa di un messaggio nella sua coda di messaggi. Quando un altro programma (schedulatore) ha necessità di sbloccarlo, manda un messaggio su quella coda. Quando il processo che si era sospeso (schedulato) riceve il messaggio riprende la sua normale esecuzione finché non ha completato il suo lavoro, a questo punto si sospende nuovamente in attesa di un nuovo messaggio. Se lo schedulatore si era messo in modo d'attesa il processo schedulato lo avvisa con un altro messaggio. Schedulatore e schedulato possono scambiarsi informazioni attraverso l'invio di 5 interi da 32 bit quali parametri. Questo meccanismo è valido per tutti i programmi on-line fatti partire da *fs*. Ciascuna coda è implementata inviando messaggi il cui tipo è derivato direttamente dal nome del programma (max 5 caratteri). Questo tipo di meccanismo non protegge contro la possibilità di aver più istanze dello stesso programma in esecuzione.
4. Semafori. I semafori sono i meccanismi di base per sincronizzare i processi che vogliono accedere ad una risorsa, vale a dire per indicare se la risorsa è in uso o disponibile. L'implementazione messa a disposizione dal Field System è di tipo binario, ovvero se un semaforo è disponibile può essere preso, se non lo è può essere rilasciato. Una richiesta per ottenere un semaforo non disponibile blocca il programma che la ha effettuata in attesa che la risorsa sia di nuovo libera. Ogni semaforo è solitamente identificato con un numero; il Field System però associa ogni numero con un nome di 5 caratteri e consente di effettuare le operazioni sui semafori riferendosi a tale nome. Se viene utilizzato un nome non ancora specificato, il Field System lo associa ad un numero di semaforo non ancora utilizzato; tale associazione risulterà la stessa, fino al successivo riavvio del sistema. La funzione che implementa quest'associazione è semplice ed è riportata sotto:

```
static long mtype(name)
char name[5];
{
    int i;
    long val;
    val=0;
    for (i=0;i<5;i++) {
       if(name[i] != ' ' && name[i] != 0 ) {
           val+=(tolower(name[i])-'a')<<(5*i);
       }
    }
    return(val);
}
```

## 2.3 Il comando SNAP.

Il comando SNAP *peakf* è implementato come modulo di *stqkr*. Quando *boss* trova nel suo stack il comando in questione ne delega a *stqkr* l'esecuzione; per far questo tramite il meccanismo della schedulazione a *stqkr* vengono passati alcuni parametri tra i quali la sottoroutine che deve essere eseguita, il task di quella sottoroutine e la stringa contenente l'intera riga di comando. Tramite queste informazioni *stqkr* capisce quale modulo deve essere eseguito. Prima di passare

l'esecuzione al modulo adibito *stqkr* effettua il parsing della linea di comando e compila una struttura come quella illustrata sotto:

```
#define MAX_ARGS  100          /* maximum number of args after `=' */
struct cmd_ds {                /* command data structure */
      char *name;              /* pointer to command name STRING */
      char equal;              /* '=' if '=' follows command name,
                                    '\0' otherwise */
      char *argv[MAX_ARGS];    /* pointers to argument STRINGS,
                                  vaild data terminated by a NULL pointer */
};
```

Il modulo adibito all'esecuzione del comando SNAP *peakf* (Appendice B) non fa altro che analizzare questa struttura e in base a questa, compiere determinati comportamenti. Per prima cosa viene controllato il campo *equal*; se esso è diverso da '=' allora l'unica operazione fatta è quella di schedulare il programma *peakf* in modo non bloccante; in caso contrario significa che il comando è stato usato o in modalità query o nella modalità per settaggio parametri. Il campo *argv* è un vettore di stringhe che elenca i vari parametri usati nella riga di comando. Se il valore d'indice 0 è un "?" allora il comando è una modalità query; in questo caso viene composto un buffer con i valori di dei parametri di configurazione di *peakf* attualmente in vigore e inserito nel log file. L'ultimo caso è quello della modalità settaggio parametri; se *argv* è una stringa vuota, significa che il valore è stato omesso quindi verrà usato per quel parametro il valore di default, se è una "*" allora il parametro rimane invariato, altrimenti si esegue la decodifica del valore e lo si assegna al parametro. Un qualsiasi caso non contemplato in quelli appena descritti o un valore non adatto al corrispondente parametro genera un errore con codice –1006.

I parametri di configurazione di *peakf* sono immagazzinati nell'area di memoria comune dei programmi di stazione. *Peakf* quando viene schedulato dalla parte di codice appena descritta per configurarsi utilizza i valori immagazzinati in quest'area.

## 2.4 La struttura generale del programma.

In Figura 2 è illustrato il diagramma di flusso che schematizza com'è strutturato *peakf*. I riquadri verdi indicano azioni che sono considerate "singole" per cui non se ne dà ulteriore spiegazione, se non un rimando alle appendici per dettagli implementativi; i riquadri azzurri descrivono operazioni che saranno analizzate nel dettaglio, gli ovali gialli infine, sono operazioni svolte dalle funzioni di libreria descritte in Appendice A.

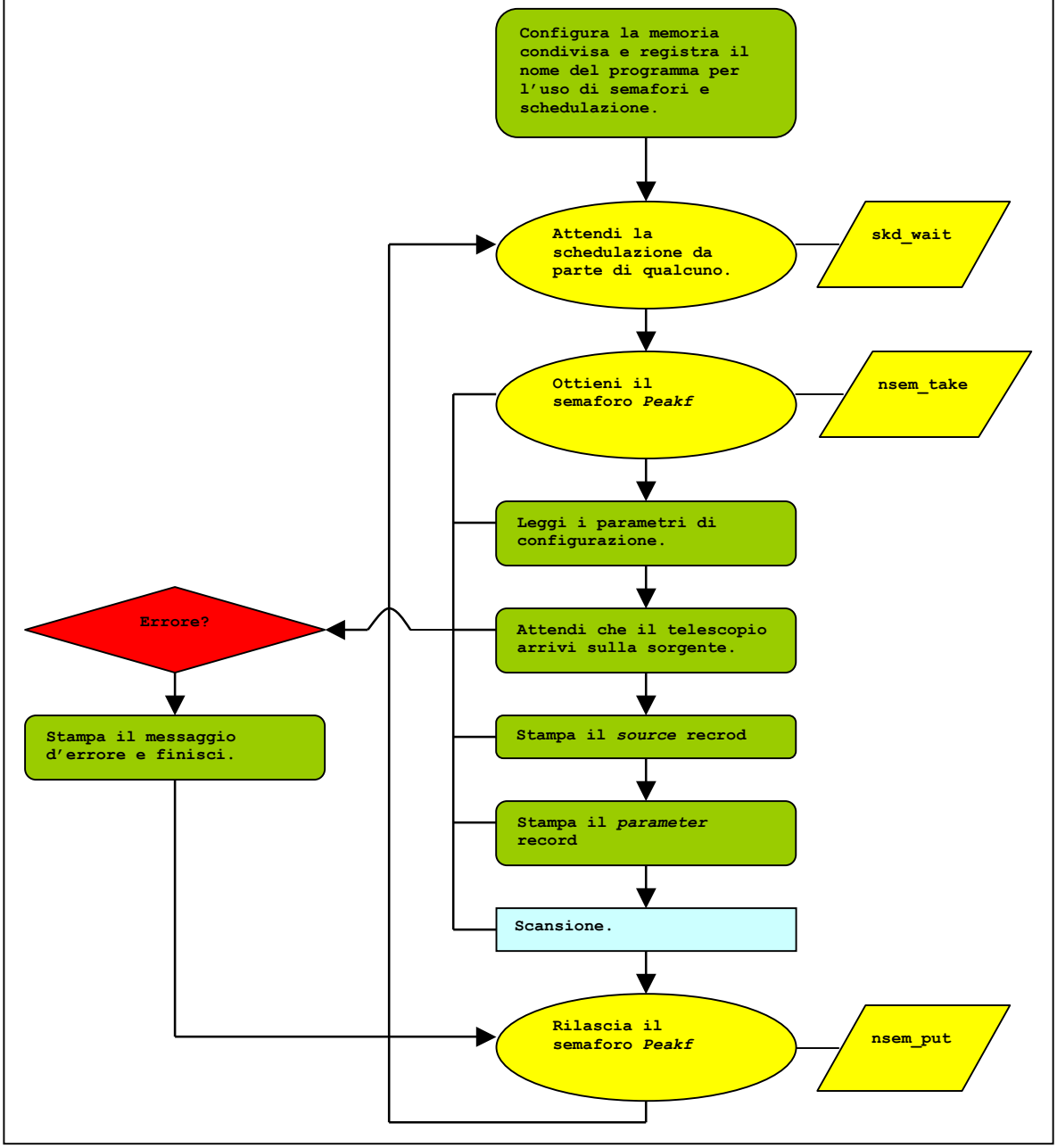
La gestione degli errori è effettuata completamente a questo livello del programma; da sottolineare che se all'uscita dei vari moduli è riscontrata una qualsiasi condizione d'errore, l'esecuzione viene interrotta e il corrispondente messaggio d'errore viene registrato nel log file. Le eventuali operazioni di ripristino necessarie dopo un errore invece, sono riservate al modulo che lo ha rilevato.

### 2.4.1 L'algoritmo di scansione.

Il primo passo della routine di misura è di salvare le posizioni degli assi su cui dovrà lavorare, in questo modo, se si verifica un errore, il fuoco potrà essere mosso all'ultima posizione corretta (ad eccezione del caso in cui l'errore sia dovuto proprio alla movimentazione degli assi).

Successivamente viene ricavata la marca di rumore per il dispositivo utilizzato (parametro *Dev*), in funzione della frequenza e della polarizzazione cui è collegato.

**Figura 2 - Diagramma di flusso del main di *Peakf*.**



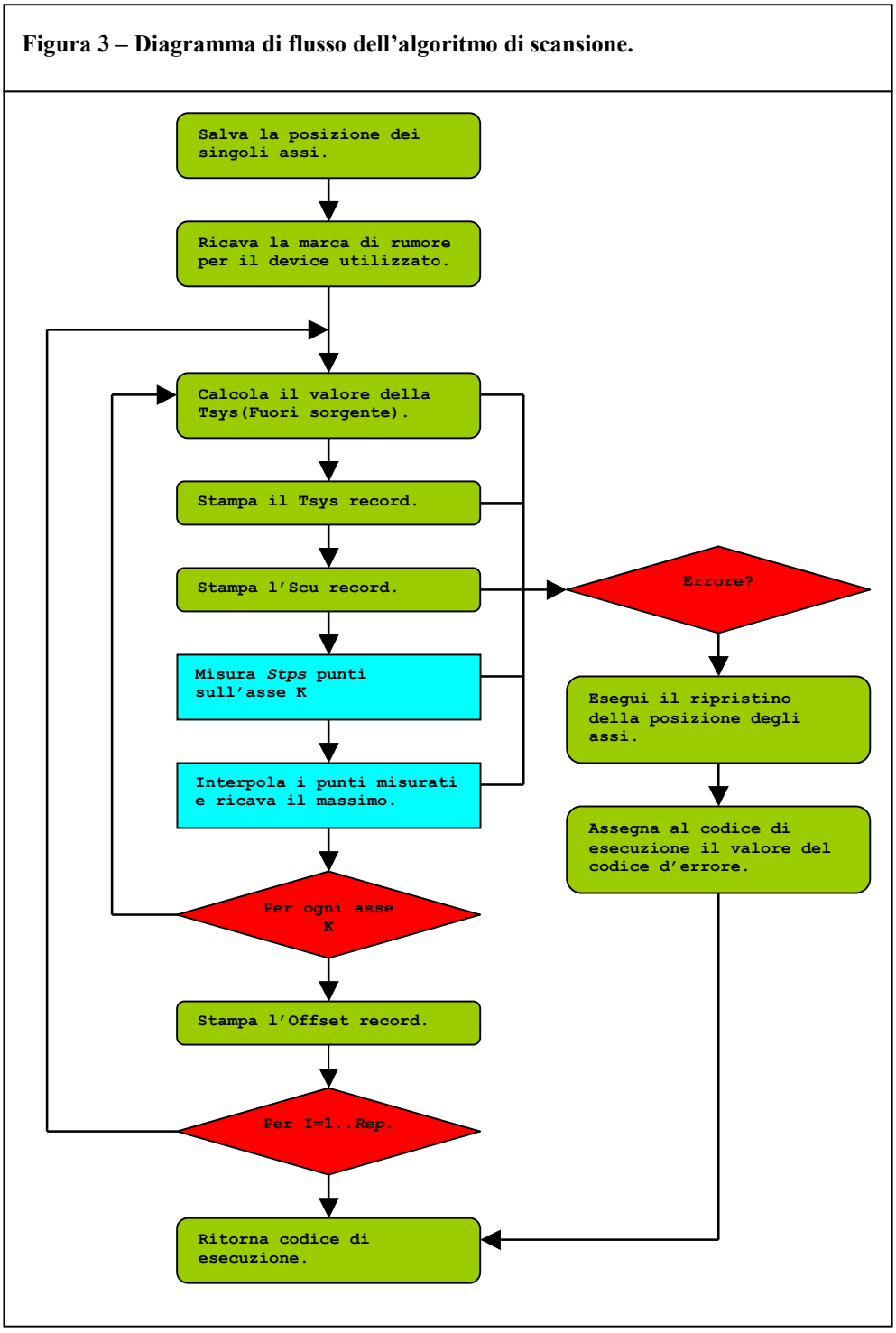

Da questo punto i passi sono iterati per il numero di ripetizioni programmate (parametro *Repeat*) e per il numero d'assi su cui va eseguita la misura. Tutti questi passi sono descritti in Figura 3.

Il calcolo della temperatura di sistema del dispositivo viene effettuato fuori dalla sorgente. L'asse di azimuth dell'antenna è spostato di un grado diviso per il coseno dell'elevazione, in seguito vengono fatte le misure di TPI con la marca accesa e spenta. La formula applicata è la solita:

$$Slope = \frac{T_{Cal}}{TPI_{Cal} - TPI_{Sky}} \quad K/Counts \tag{1}$$

$$T_{Sys} = (TPI_{sky} - TPI_{Zero}) \cdot Slope \quad K \tag{2}$$

**Figura 3 – Diagramma di flusso dell'algoritmo di scansione.**




La procedura passa poi all'effettuazione della misura vera e propria e, quindi alla ricerca del punto di massimo. Di seguito ne verrà fornita una descrizione.

## 2.4.2 Acquisizione dei punti.

In base ai parametri di programmazione e alla posizione assoluta dell'asse (posizione più relativo offset) vengono calcolati i punti in cui muovere l'asse per effettuare le misure. Complessivamente vengono generati *Stps* punti spaziati l'uno dall'altro di un valore dato da $Off/Steps$ millimetri; il punto centrale è quello dato dalla posizione attuale dell'asse.

A questo punto l'asse viene portato sul primo punto calcolato per iniziare la misura dei TPI. Il comando di una nuova posizione sull'asse si può riassumere in questo modo:

a) Invia una classe a *scu* per informarlo che c'è una nuova posizione da comandare.

b) Calcola il tempo d'attesa per l'asse K $wait = |D \cdot Delay(K)| + 150$. Dove D*elay* è la tabella con i tempi di spostamento dei singoli assi riportati in Tabella 4 e *D* è la distanza in millimetri da percorrere.

c) Aspetta *wait* centisecondi.

d) Controlla che l'asse abbia raggiunto la posizione comandata, altrimenti riparti da a). Se anche il secondo tentativo fallisce dichiara errore.

**Tabella 4 – Tempi di percorrenza di ciascun asse per 1 millimetro (centisecondi/millimetro).**

| **Primary Focus** | **Z** | | **Y** | | |
|---|---|---|---|---|---|
| | 28.57 | | 28.57 | | |
| **Secondary Focus** | **X** | **Y** | **Z1** | **Z2** | **Z3** |
| | 2.0 | 5.88 | 2.08 | 2.08 | 2.08 |

Le traslazioni lungo l'asse Y del ricevitore in fuoco primario e i movimenti dello specchio secondario lungo l'asse Y e l'asse X impongono un errore di puntamento in cielo che deve essere compensato prima di effettuare la misura [2]. La Tabella 5 riassume le correzioni applicate da *peakf* quando lavora sugli assi prima menzionati; ΔP indica la deviazione dell'angolo di puntamento (va corretta per il coseno dell'elevazione in caso si stia parlando dell'asse di azimuth), il rapporto $\Delta P/\Delta Axis$ è i gradi per millimetro di spostamento effettivamente usati per compensare.

**Tabella 5 – Fattori di compensazione del puntamento.**

| **Asse** | **Convenzione** | **$\Delta P/\Delta Axis$** |
|---|---|---|
| | **YP** | |
| Elevazione | Se ΔP>0 l'antenna va verso nord.<br>Se ΔYP>0 l'attuatore si sposta verso nord. | **+0.0044 (°/mm)** |
| | **YS** | |
| Elevazione | Se ΔP>0 l'antenna va verso nord.<br>Se ΔYS>0 l'attuatore si sposta verso sud | **-0.0037 (°/mm)** |
| | **XS** | |
| Azimuth | Se ΔP>0 l'antenna va verso ovest.<br>Se ΔZS>0 l'attuatore si sposta verso est | **-0.0037 (°/mm)** |

Una volta ottenute le misure di TPI, i dati vengono convertiti in gradi Kelvin mediante il fattore di scala ricavato nella (1); a questo valore viene sottratto il valore della $T_{sys}$, calcolata con la (2) per ottenere i dati relativi solamente al contributo della sorgente.

Al termine di ogni misura viene stampato il record *axis*.

## 2.4.3 Algoritmo di fitting.

Nel caso più generale, i dati raccolti nella fase descritta precedentemente hanno un andamento che obbedisce ad un modello che non dipende linearmente dai parametri $a_k$ $k=1,2,...M$ che lo descrivono. Per questo motivo si è scelta una routine di fitting dei dati non-lineare.

Il migliore approccio a questo problema è definire una funzione di merito $\chi^2$ e determinare i parametri che meglio interpolano i dati attraverso la minimizzazione di questa. Dati dei valori approssimativi per i parametri, occorre sviluppare una procedura iterativa (dovuta alla non-linearità del problema) che migliora di volta in volta la soluzione di prova.

Se il modello usato per il fitting è $y = y(x;a)$, la funzione di merito sarà:

$$\chi^2(a) = \sum_{i=1}^{N}\left[\frac{y_i - y(x_i;a)}{\sigma_i}\right]^2 \tag{3}$$

dove $\sigma_i$ è la standard deviation delle misure $y_i$ nel punto $x_i$.

Il metodo utilizzato è quello di *Levenberg-Marquardt* che è divenuto lo standard per questo tipo di problemi. Se l'approssimazione iniziale è buona, passare da questa all'ottima significa:

$$a_{\min} = a_{cur} + D^{-1}\cdot\left[-\nabla\chi^2(a_{cur})\right] \tag{4}$$

D'altra parte l'approssimazione può rivelarsi un'approssimazione locale, allora:

$$a_{next} = a_{cur} - Const\times\nabla\chi^2(a_{cur}) \tag{5}$$

dove $D$ è la matrice MxM delle derivate parziali di secondo ordine rispetto ai parametri della funzione di merito $\chi^2$ e $\nabla$ (Nabla) è l'operatore gradiente. Calcolando il gradiente di $\chi^2$, ponendo $[\alpha] = \frac{1}{2}D$ ovvero $\alpha_{kl} = \frac{1}{2}\frac{\delta^2\chi^2}{\delta a_k \delta a_l}$ e $\beta_k = -\frac{1}{2}\frac{\delta\chi^2}{\delta a_k}$ la (4) diventa un insieme di equazioni lineari:

$$\sum_{l=1}^{M}\alpha_{kl}\delta a_l = \beta_k \tag{6}$$

la soluzione delle equazioni dà l'incremento $\delta a_l$ che, aggiunto all'approssimazione corrente ricava la prossima approssimazione. La (5) si traduce in

$$\delta a_l = Const\times\beta_l \tag{7}$$

La (7), per il metodo *Marquardt*, diventa:

$$\delta a_l = \frac{1}{\lambda a_{ll}}\beta_l \tag{8}$$

Le equazioni (8) e (6) possono essere combinate definendo una nuova matrice come segue:

$$\begin{cases} \alpha' = \alpha_{jj}(1+\lambda) \\ \alpha'_{jk} = \alpha_{jk} \quad (j \neq k) \end{cases} \tag{9}$$

da cui si ricava:

$$\sum_{l=1}^{M} \alpha'_{kl}\, \delta a_l = \beta_k \qquad (10)$$

Quando $\lambda$ è grande $\alpha'$ diventa diagonalmente dominante così la (10) diventa identica alla (8), se $\lambda$ si avvicina a zero la (10) si approssima alla (6).

Data una stima iniziale dei parametri *a*, il metodo procede attraverso i seguenti passi:

1. Calcola $\chi^2$.
2. Prendi un piccolo valore per $\lambda$, diciamo $\lambda$=0.001.
3. Risolvi le equazioni lineari (10) e ricava $\delta a$ e calcola $\chi^2(a+\delta a)$.
4. Se $\chi^2(a+\delta a) \geq \chi^2(a)$ incrementa $\lambda$ di un fattore elevato, per esempio 10. Torna al punto 3.
5. Se $\chi^2(a+\delta a) < \chi^2(a)$ decrementa $\lambda$ di un fattore elevato, per esempio 10. Aggiorna la soluzione parziale $a \leftarrow a+\delta a$. Torna al punto 3.

Iterare fino al raggiungimento del limite dato dalla precisione di macchina è generalmente inutile visto che la soluzione ottenuta con questo metodo è solo una stima statistica dei parametri. Solitamente un cambio nel valore di $\chi^2$ molto inferiore a 1 non è statisticamente significativo, per questo motivo si sceglie di interrompere le iterazioni alla seconda volta che $\chi^2$ decrementa di un valore trascurabile ($10^{-3}$). Il motivo per cui il metodo può non convergere è che la matrice $\alpha'$ è singolare e il sistema (10) non ha soluzione.

Una volta che la soluzione è stata trovata è possibile calcolare, ponendo $\lambda$=0 la matrice *C* che è la matrice di covarianza sugli errori dei parametri fittati:

$$[C] = [\alpha]^{-1} \qquad (11)$$

Il metodo è implementato nella libreria *fitlib* e il codice è riportato in appendice.

## 2.4.4 Scelta del modello per l'interpolazione.

Per la scelta del modello o funzione interpolante sono stati raccolti alcuni dati utilizzando il ricevitore K in fuoco primario. La sorgente era W3OH, la banda del video converter, centrato in frequenza sulla riga maser, 500 KHz. I risultati ottenuti per l'asse ZP e YP sono mostrati in Figura 4 e in Figura 5 rispettivamente.

Dall'analisi delle due figure è facile evincere che i due assi corrispondono a due modelli diversi. Per ottenere una misura più accurata possibile *peakf* utilizza due funzioni diverse come modello per l'algoritmo d'interpolazione: una funzione di tipo gaussiano e una di tipo parabolico.

La funzione di tipo gaussiano è ottenuta come somma di una funzione di Gauss e l'equazione di una retta. Questo modello è applicato agli assi ZS, ZP ed ha 5 parametri: Ampiezza della gaussiana (*A*), il valor medio ($\mu$), la deviazione standard ($\sigma$), coefficiente angolare e termine noto della retta (*m, c*):

$$f(x) = Ae^{-2.7725882\left(\frac{x-\mu}{\sigma}\right)^2} + mx + c \qquad (12)$$

le cui derivate parziali rispetto ai parametri, necessarie per l'algoritmo di fitting, sono:

$$\frac{\delta f(x)}{\delta A} = e^{-2.7725882\left(\frac{x-\mu}{\sigma}\right)^2}$$

**Figura 4 – Andamento dei campioni ottenuto spostando il ricevitore in fuoco primario lungo l'asse Z.**

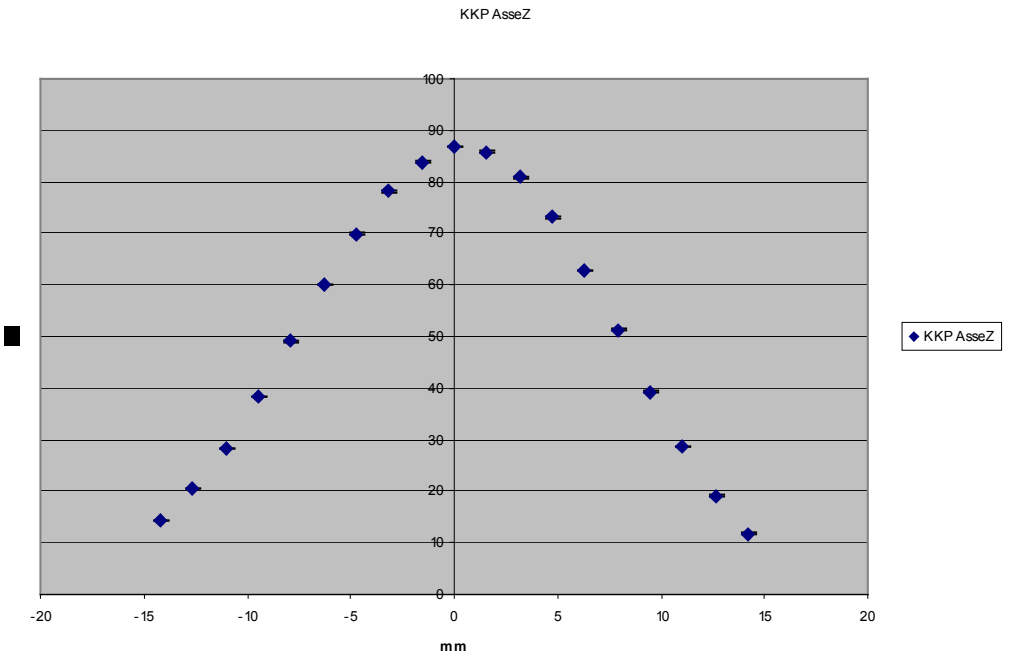
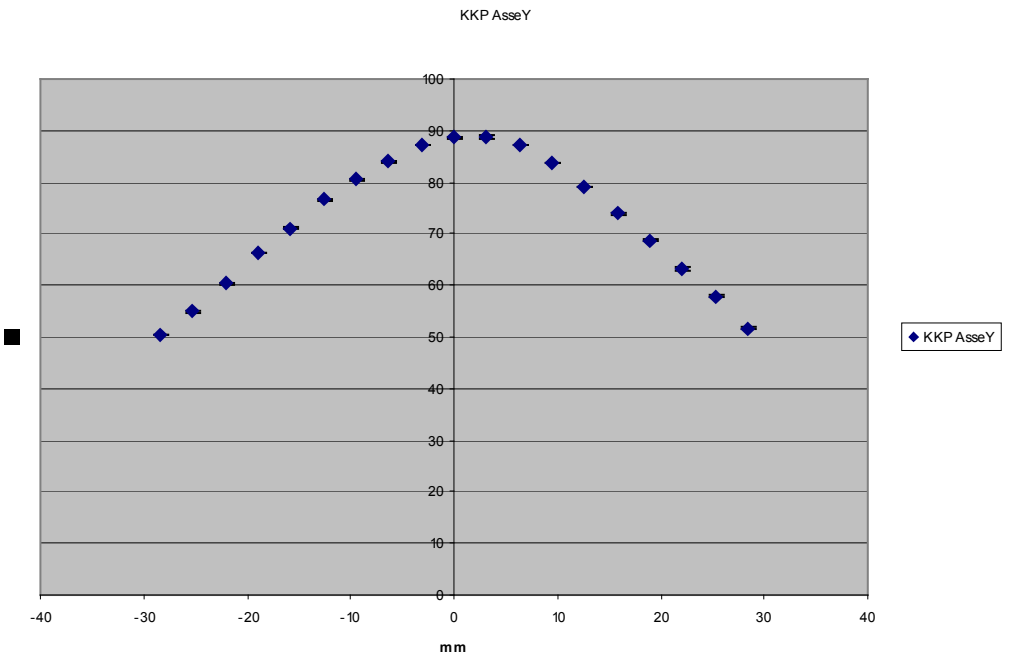



**Figura 5 – Andamento dei campioni ottenuto spostando il ricevitore in fuoco primario lungo l'asse Y.**



$$\frac{\delta f(x)}{\delta \mu} = 5.5451774 A \left( \frac{x-\mu}{\sigma} \right) e^{-2.7725887\left(\frac{x-\mu}{\sigma}\right)^2} \frac{1}{\sigma}$$

$$\frac{\delta f(x)}{\delta \sigma} = 5.5451774 A \left( \frac{x-\mu}{\sigma} \right)^2 e^{-2.7725887\left(\frac{x-\mu}{\sigma}\right)^2} \frac{1}{\sigma}$$

$$\frac{\delta f(x)}{\delta m} = x$$

$$\frac{\delta f(x)}{\delta c} = 1$$

Il modello parabolico è descritto invece, dalla classica equazione di secondo grado in cui i parametri sono ovviamente 3: *a,b,c*. I dati per gli assi YP, YS, XS sono interpolati secondo questo modello:

$$f(x) = ax^2 + bx + c \tag{13}$$

le cui derivate parziali sono ovviamente:

$$\frac{\delta f(x)}{\delta a} = x^2$$

$$\frac{\delta f(x)}{\delta b} = x$$

$$\frac{\delta f(x)}{\delta c} = 1$$

## 2.4.5 Stima dei parametri iniziali e localizzazione del massimo.

Come visto in precedenza la stima dei parametri iniziali è importante non solo per far si che l'algoritmo di fitting converga più velocemente ad una soluzione ma soprattutto perché questa sia la migliore possibile. Per i due modelli presi in considerazione precedentemente la stima dei parametri è ovviamente differente. Sia $X=\{x_1,x_2,\ldots,x_n\}$ l'insieme delle posizioni sull'asse, $Y=\{y_1,y_2,\ldots,y_n\}$ l'insieme dei campioni ottenuti.

Nel caso gaussiano:

$$A = \max\{y_i ; i = 1..n\}$$

$$\mu = x_j : (x_j, A) \in X \times Y$$

$$\sigma = Off\ \ parameter$$

$$m = \frac{y_n - y_1}{x_n - x_1}$$

$$c = y_1 - mx_1$$

Alla fine dell'interpolazione il parametro $\mu$ darà l'indicazione dell'offset da applicare alla posizione dell'asse per ottenere la massimizzazione del fuoco.

Nel caso di fit parabolico la stima dei tre parametri avviene sfruttando le espressioni che danno le coordinate del vertice della parabola $V\left(x_v = -\frac{b}{2a} \quad y_v = -\frac{\Delta}{4a}\right)$ e supponendo che $(x_v\ y_v) = (x_i\ y_i) : (x_i\ y_i) \in X \times Y, y_i = \max\{y_j ; j = 1..n\}$, in altre parole che la coppia che occupa il vertice della parabola è quella col campione di valore maggiore.

$$c = y_{n/2}$$

$$a = \frac{c - y_v}{x_v^2}$$

$$b = -2ax_v$$

L'offset da applicare per ottenere la massimizzazione del fuoco è dato dalla:

$$x = -\frac{b_{fitted}}{2a_{fitted}} \tag{14}$$

In questo caso si è resa necessaria l'applicazione delle formule di propagazione per la stima degli errori commessi [5]. Siano $\sigma_a^2$ e $\sigma_b^2$ le varianze fornite dall'algoritmo dei due parametri *a* e *b*, allora

$$\frac{\sigma_x^2}{x^2} = \frac{\sigma_a^2}{a^2} + \frac{\sigma_b^2}{b^2} - 2\frac{\sigma_{ab}^2}{ab} \tag{15}$$

I risultati della routine d'interpolazione sono riassunti nei record *gaussfit*, *parabolicfit* e *err* che riporta la stima d'errore sui parametri forniti nei precedenti due.

Se l'algoritmo converge e l'offset trovato è all'interno del range definito dal parametro di configurazione *Off*, la posizione assoluta dell'asse (rilevata prima di partite con la misura) è corretta col valore trovato. La meccanica viene spostata in questa nuova posizione; se richiesto anche il puntamento dell'antenna viene corretto così come spiegato nel Paragrafo 2.4.2.

# Capitolo 3: Calibrazione a 22 GHz.

L'uso del programma ha permesso di ottimizzare il posizionamento del ricevitore in banda K nel fuoco primario dell'antenna di Medicina. Il ricevitore a 22 GHz è l'unico, tra quelli a disposizione che, data la lunghezza d'onda piuttosto corta, permette di verificare in maniera tangibile le deformazioni che la gravità impone sulla struttura dell'antenna.

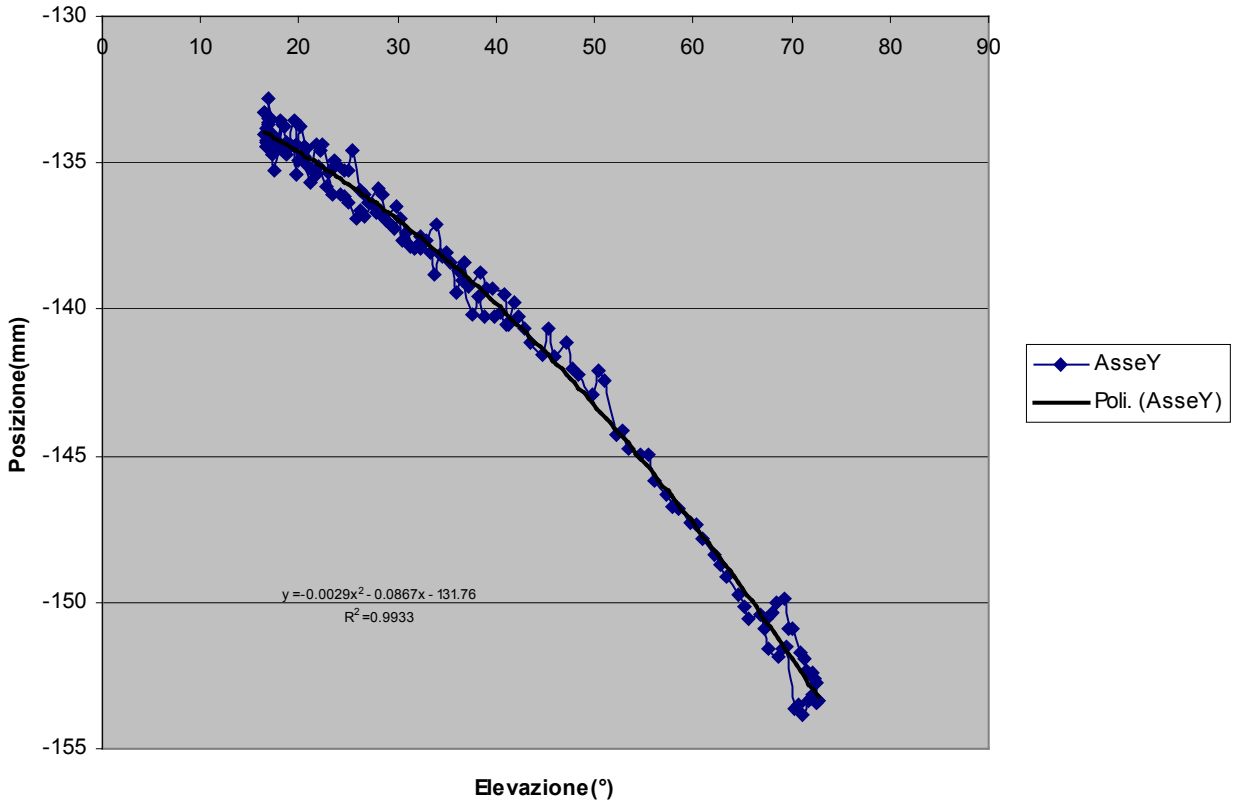
## 3.1 Curve di compensazione del fuoco e di guadagno.

Le curve degli assi YP, ZP sono state ricavate dopo una campagna di calibrazione effettuata tramite il programma Field System *acquir*. La sorgente utilizzata è DR21 seguita dal sorgere fino al culminare, il che ha garantito di coprire gran parte del range di elevazione. Sono stati alternati *peakf* per eseguire la misura e *fivept* per controllare costantemente il puntamento dell'antenna. I parametri di programmazione di *peakf* sono stati:

```
peakf=zy,3,11,30,5,v3
```

L'analisi del log file e in particolare dei record *offset* in cui compare il dato più significativo,

**Figura 6 - Curva che descrive il movimento ottimizzato per l'asse Y del fuoco primario in funzione dell'elevazione.**

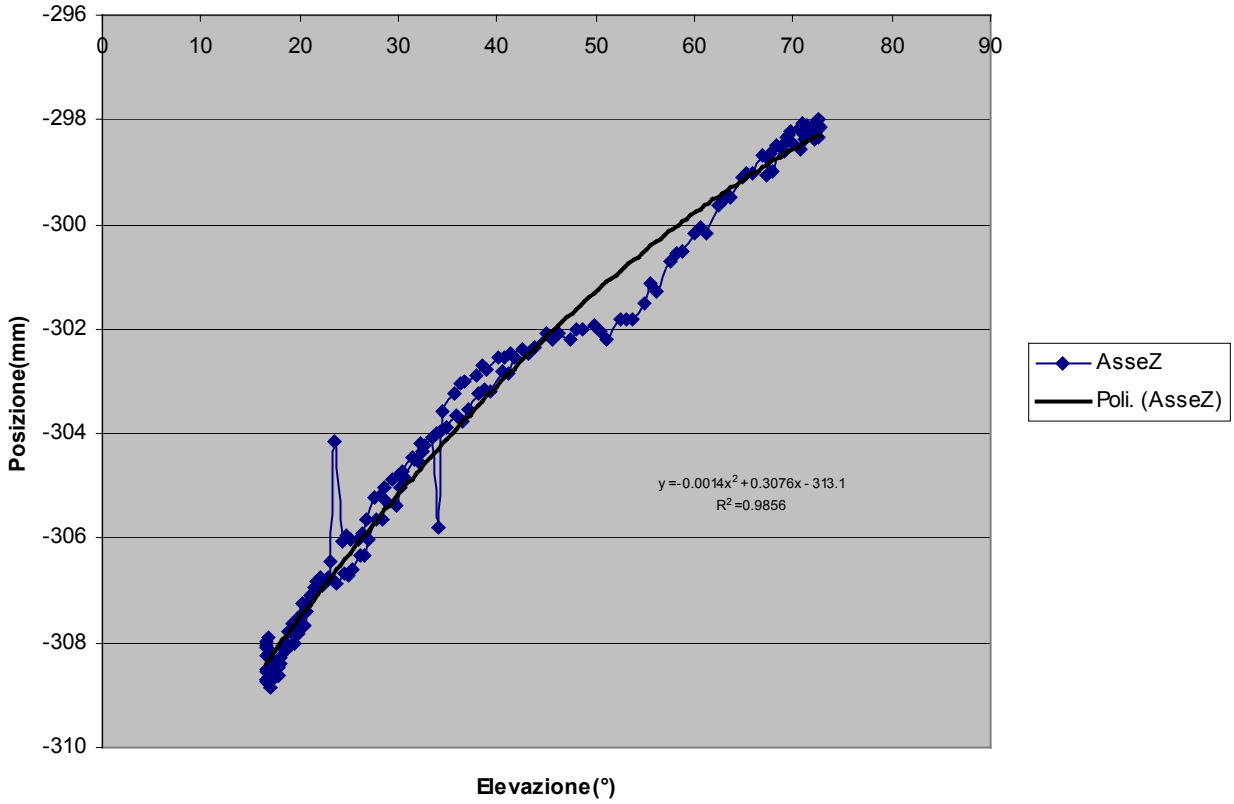



ovvero il punto dato da elevazione e posizione dell'asse in cui è stato rilevato il massimo, ha consentito di ottenere i due grafici riportati in Figura 6 e 7. In ascissa è riportata l'elevazione mentre, in ordinata la posizione in millimetri dell'asse in questione. L'interpolazione dei punti ha prodotto i polinomi di secondo grado che descrivono i movimenti dei due assi in funzione

dell'elevazione. Attualmente quando l'antenna è in osservazione a 22 GHz, il Field System provvede a spostare il ricevitore in fuoco primario secondo questi due polinomi.

Al fine di verificare la bontà dei risultati ottenuti è stata effettuata una campagna per determinare la curva di guadagno. Nelle Figure 8 e 9 sono illustrate le curve ottenute per la

**Figura 7 – Curva che descrive il movimento ottimizzato per l'asse Z del fuoco primario in funzione dell'elevazione.**




polarizzazione destra e sinistra. In ciascuna di esse è riportata la curva col fuoco primario in posizione fissa, quella col fuoco in tracking secondo i polinomi[4] e la stessa curva compensata per l'attenuazione atmosferica (*tau* medio di circa 0.11). Si può facilmente verificare come, specialmente alle basse elevazioni, l'efficienza del telescopio con il fuoco in inseguimento risenta molto meno del degrado imposto dalla gravità.

**Figure 8 – Curve di guadagno per la polarizzazione sinistra.**

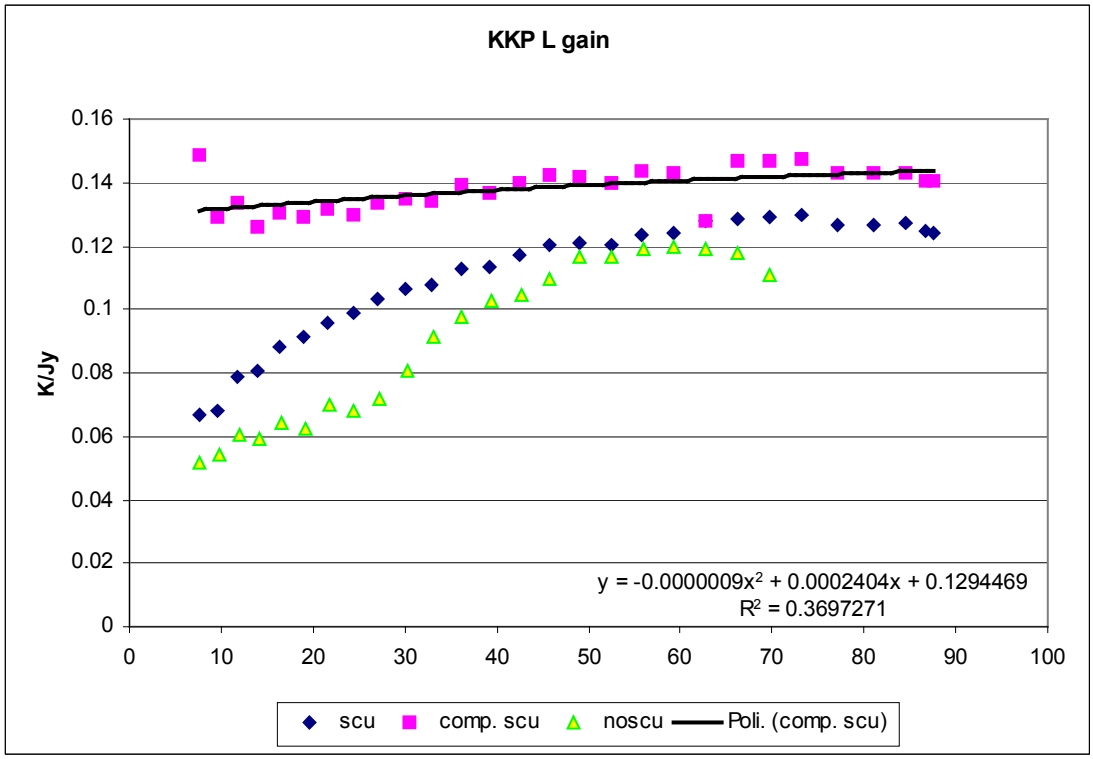
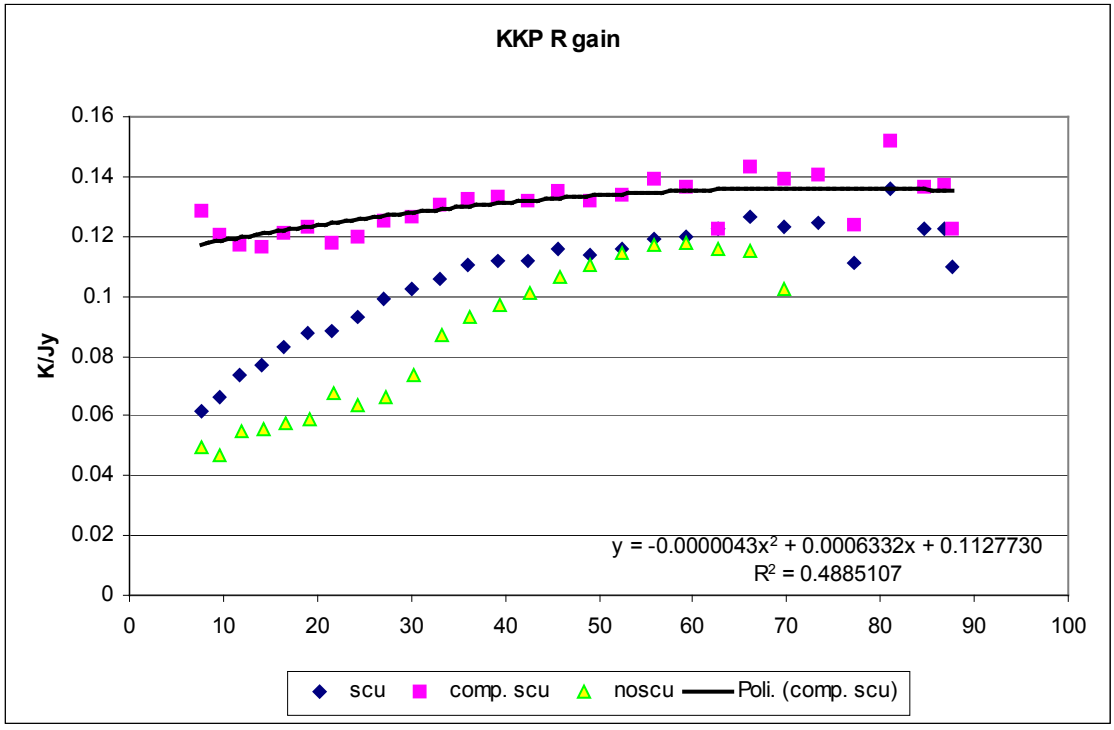



**Figure 9 – Curve di guadagno per la polarizzazione destra.**



# Appendice A: Funzioni di libreria Field System

Segue una breve presentazione delle funzioni di libreria più significative, usate da *peakf* per interfacciarsi col Field System, messe a disposizione dal sistema.

- **void cls_clr(long class)**

  Questa routine cancella un numero di classe, rimuovendo tutti i messaggi dentro la classe e deallocando il numero di classe.

  **class:** numero della classe da cancellare.

- **int cls_rcv(long class,char *buf,int iblen,int *r1,int *r2,int p1,init p2)**

  Questa routine ritorna il messaggio di testo e la sua lunghezza alla classe specificata. Il valore restituito è la lunghezza del messaggio.

  **class:** numero di classe dalla quale reperire il messaggio

  **buf:** buffer che conterrà il messaggio ritornato

  **iblen:** lunghezza di buf, viene controllato per non uscire dai limiti dell'array

  **p1,p2:** parametri di controllo, solitamente a 0.

  Output:

  **r1,r2:** parametri di uso interno al Field System.

- **void cls_snd(long *class,char *buf,int iblen,int p1,int p2)**

  Questa funzione accetta in ingresso un messaggio di testo e lo mette nella coda della classe. Un nuovo numero di classe viene allocata se quello fornita in ingresso è zero.

  **class:** numero di classe da usare. Se il valore fornito è zero, in output viene fornito il nuovo numero di classe.

  **buf:** messaggio da accodare nella classe.

  **iblen:** numero di caratteri del messaggio in buf.

  **p1,p2*:*** parametri addizionali, non usati normalmente.

- **void logita(char *msg,int ierr,char *who,char *what)**

  Formatta un messaggio, generico o d'errore per il logging ed invia il buffer a *ddout*. Questa funzione rappresenta l'unico modo per aggiungere messaggi al file di log programmaticamente.

  **msg:** messaggio da mettere a log. Null se nessun messaggio deve essere aggiunto.

  **ierr:** codice dell'errore da mettere a log, 0 se nessun errore deve essere aggiunto.

  **who:** stringa di due caratteri, identificativo del programma che aggiunge la log entry.

  **what:** stringa con informazioni aggiuntive.

- **int nsem_test(char pname[5])**

  Questa routine controlla se il semaforo associato ad un nome(nome di programma) è asserito oppure no. Ritorna 0 se il programma è in esecuzione, 1 altrimenti.

  **pname:** nome di programma da controllare, massimo 5 caratteri.

- **void nsem_put(char pname[5])**

  La routine rilascia il semaforo associato al nome di programma indicato come parametro.

  **pname:** nome del programma. Massimo 5 caratteri.

- **int nsem_take(char pname[5],int flags)**

  Questa funzione asserisce il semaforo associato al nome di programma associato. Può essere bloccante o non bloccante.

  **pname:** nome del programma. Massimo 5 caratteri.

  **flags:** se zero indica che il processo che esegue la chiamata a questa funzione rimarrà in attesa che il semaforo sia liberato e poi lo allocherà. Se non zero la funzione sarà non bloccante.

- **void rte_time(int it[5],int *iyear)**

  La funzione serve ad ottenere l'ora attualmente impostata sul Field System.

  **it:** nei campi di questo vettore la funzione immagazzina rispettivamente centesimi di secondo, secondi, minuti, ore, giorno dell'anno.

  **iyear:** Anno.

- **void skd_run(char pname[5],char stat,long ip[5])**

  Questa routine schedula un programma per l'esecuzione con i parametri ip. Il parametro stat controlla se la funzione aspetterà o meno che il programma finisca.

  **pname:** nome del programma da eseguire. Deve essere esattamente 5 carattreri.

  **stat:** 'w' per aspettare il termine dell'esecuzione, 'n' per continuare il normale flusso d'esecuzione senza aspettare.

  **ip:** array di parametri passati al programma quando inizia l'esecuzione. Solitamente i campi hanno il seguente significato:

  1. ip[0]: numero di classe.
  2. ip[1]: numero di registrazioni (messaggi) nella classe.
  3. ip[2]: risultato dell'esecuzione (usato come valore di ritorno)
  4. ip[3]: identificatore di 2 caratteri del programma chiamante.
  5. ip[4]: non usato.

- **void skd_wait(char pname[5],long ip[5],unsigned par)**

  Quando questa funzione viene invocata, il programma chiamante rimane sospeso finché qualche altro programma lo sblocca con la funzione skd_run. Quando il programma esce da questa funzione per continuare l'esecuzione nell'array ip ci sono i parametri impostati con la funzione skd_run.

  **pname:** il nome del programma chiamante. Deve essere esattamente 5 caratteri.

  **ip:** array dei parametri.

  **par:** non normalmente usato.

- **void skd_par(long ip[5])**

Questa funzione restituisce i parametri di risposta di un programma appena schedulato. Va utilizzata subito dopo una chiamata a skd_run per recuperare i parametri che esso può aver ritornato.

**ip:** array dei parametri ritornati.

- **int brk_chk(char name[5])**

  Controlla che nessun messaggio di terminare sia stato inviato al programma specificato. Ritorna 0 se nessun messaggio è stato ricevuto.

  **name:** nome del programma da controllare.

- **void brk_snd(char name[5])**

  Manda un messaggio di terminare l'esecuzione al programma specificato. Questa funzione è utilizzata dal programma off-line *brk*. Se il programma che si vuole terminare non è in esecuzione in quel momento riceverà il messaggio appena si avvierà.

  **name:** nome del programma cui far pervenire il messaggio di terminare.

# Appendice B: Modulo aggiuntivo di *stqkr*

```
/*
*****************************************************************************
* File name  : lpeakf.c                                                     *
* Desciption : lpeakf starts peakf program. This file is included           *
*              in station quickr.                                           *
*                                                                           *
* Author     : Andrea Orlati a.orlati@ira.cnr.it                            *
* Update Log                                                                *
* When          Who         What                                          *
* 18/09/2202    AO            Creation                                    *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*****************************************************************************
*/

#if !defined LPEAKF_DEFINITION
#define LPEAKF_DEFINITION

// {}

int ParamType(param,value,type)
char *param;
void *value;
int type;
{
        char *dummy;
        switch (type) {
                case 1 : {       // string
                        char *tmp;
                        tmp=(char *)value;
                        strcpy(tmp,param);
                        break;
                }
                case 2  : {      // integer
                        int *tmp;
                        int conv;
                        tmp=(int *)value;
                        conv=strtol(param,&dummy,10);
                        if (errno==ERANGE) return -1;
                        else *tmp=conv;
                        break;
                }
                case 3 : {       // float
                        float *tmp;
                        tmp=(float *)value;
                        if (sscanf(param,"%f",tmp)!=1) return -1;
                        break;
                }
        }
        return 0;
}

int DeviceNumber(Names,Name,size)
char *Names[];
char *Name;
int size;
{
        int i;
        for (i=0;i<size;i++) {
                if (strcasecmp(Names[i],Name)==0) return i;
        }
        return -1;
}


void lpeakf(command,ip,isub,itask)
struct cmd_ds *command;
long ip[5];
int isub,itask;
{
        long i;
        int ierr;
        char buff[MAX_BUF];
        char Tmp[MAX_BUF];
        peakf_cmn *cmn;
        int Axis;
        int Repetitions,NStep,NSamples;
        T_Devices Device;
```

```
        float DeltaStep;
        for (i=0;i<5;i++) ip[i]=0;
        ierr=0;
        if ((command->equal)=='=') {
                if ((command->argv[0]!=NULL) && (strcmp(command->argv[0],"?"))==0) {
                        strcpy(buff,"/");
                        strcat(buff,PROGRAM_NAME);
                        strcat(buff,"/");
                        cmn=&(st->peakfcommon);
                        sprintf(Tmp,"%s,%d,%d,%.4f,%d,%s",AxisConfig[cmn->Axis],cmn->Repetitions,cmn->NStep,cmn-
>DeltaStep,cmn->NSamples,DeviceName[cmn->Device]);
                        strcat(buff,Tmp);
                        cls_snd (&ip[0],buff,strlen(buff), 0, 0);
                        ip[1]=1;
                        return;
                }
                else {
                        cmn=&(st->peakfcommon);
                        // load defaults
                        Axis=PEAKF_AXIS;
                        Repetitions=PEAKF_REPETITIONS;
                        NStep=PEAKF_NSTEP;
                        DeltaStep=PEAKF_DELTASTEP;
                        NSamples=PEAKF_NSAMPLES;
                        Device=PEAKF_DEVICE;
                        for (i=ierr=0;((i<PEAKF_PARAM_NUMBER) && (!ierr));i++) {
                                if (command->argv[i]==NULL) break;  // no other parameters are specified,
defaults are used
                                switch (i) {
                                        case 0 : {    // axis
                                                if (command->argv[i][0]==0) {     // use defaults;
                                                }
                                                else if (command->argv[i][0]=='*') {  // use current value
                                                        Axis=cmn->Axis;
                                                }
                                                else if ((Axis=DeviceNumber(AxisConfig,command-
>argv[i],PEAKF_AXISCONFIG_SIZE))<0) {
                                                        ierr=-1006;
                                                }
                                                break;
                                        }
                                        case 1 : {    // repetitions
                                                if (command->argv[i][0]==0) {     // use defaults;
                                                }
                                                else if (command->argv[i][0]=='*') {  // use current value
                                                        Repetitions=cmn->Repetitions;
                                                }
                                                else if (ParamType(command->argv[i],&Repetitions,2)) {
                                                        ierr=-1006;
                                                }
                                                break;
                                        }
                                        case 2 : {    // NStep
                                                if (command->argv[i][0]==0) {     // use defaults;
                                                }
                                                else if (command->argv[i][0]=='*') {  // use current value
                                                        NStep=cmn->NStep;
                                                }
                                                else if (ParamType(command->argv[i],&NStep,2)) {
                                                        ierr=-1006;
                                                }
                                                break;
                                        }
                                        case 3 : {    // DeltaStep
                                                if (command->argv[i][0]==0) {     // use defaults;
                                                }
                                                else if (command->argv[i][0]=='*') {  // use current value
                                                        DeltaStep=cmn->DeltaStep;
                                                }
                                                else if (ParamType(command->argv[i],&DeltaStep,3)) {
                                                        ierr=-1006;
                                                }
                                                break;
                                        }
                                        case 4 : {    // NSamples
                                                if (command->argv[i][0]==0) {     // use defaults;
                                                }
                                                else if (command->argv[i][0]=='*') {  // use current value
                                                        NSamples=cmn->NSamples;
                                                }
                                                else if (ParamType(command->argv[i],&NSamples,2)) {
                                                        ierr=-1006;
                                                }
                                                break;
                                        }
                                        case 5 : {    // device
                                                if (command->argv[i][0]==0) {     // use defaults;
                                                }
                                                else if (command->argv[i][0]=='*') {  // use current value
                                                        Device=cmn->Device;
```

```
                                                }
                                                else if ((Device=DeviceNumber(DeviceName,command-
>argv[i],PEAKF_DEVICENAME_SIZE))<0) {
                                                        ierr=-1006;
                                                }
                                                break;
                                        }
                                }
                        }
                        if (ierr==0) { // range checks
                                if (NStep>PEAKF_MAX_NSTEP) {
                                        NStep=PEAKF_MAX_NSTEP;
                                }
                                else if ((NStep%2)==0) {  // even
                                        if (NStep<PEAKF_MAX_NSTEP) NStep++;
                                        else NStep=PEAKF_MAX_NSTEP;
                                }
                        }
                        else {
                                ip[2]=ierr;
                                ip[1]=0;
                                memcpy(ip+3,PROGRAM_NICK,2);
                                return;
                        }
                        if (ierr!=0) {
                        }
                        else {
                                cmn->Axis=Axis;
                                cmn->Repetitions=Repetitions;
                                cmn->NStep=NStep;
                                cmn->DeltaStep=DeltaStep;
                                cmn->NSamples=NSamples;
                                cmn->Device=Device;
                        }
                }
        }
        else {
                // ***********************************************
                // controllo che il programma non stia già in esecuzione.
                skd_run("peakf",'n',ip);
                return;
        }
}

#endif
```

# Appendice C: Sorgenti

## c.1 Antcn.c

```
/* antcn */
/* this function send to Antcn a request */
/* return zero if no error was found, otherwise -1. */
/* ip1 : request to antcn (see below) */
/* ierr : error code */
/* Thank you Ed, for this function */
/* possible antcn modes: */
/* 0 : init */
/* 1 : new source request */
/* 2 : new offset Request */
/* 3 : On-source status with error logging */
/* 4 : Direct Antenna comunication */
/* 5 : On source Status withaout error logging */
/* 6 : Reserved for field system */
/* 7 : Additiona information logging */

#include <stdio.h>
#include "peakf.h"
#include "private.h"

int antcn(ip1,ierr)
long ip1;
int *ierr;
{
        long ip[5] = {0,0,0,0,0};
        int i;
        ip[0]=ip1;
        ierr=0;
        for(i=0;i<MAX_RETRY;i++) {
                if(brk_chk(PROGRAM_NAME)!=0) {
                        *ierr=ERR_BRK_DTCTD;
                        return -1;
                }
                skd_run("antcn",'w',ip);
                if(ip[2]>=0) return 0;
                if (ip[1]!=0) {
                        cls_clr(ip[0]);
                        ip[0]=ip[1]=0;
                }
        }
        *ierr=ERR_NTCN_FLR;
        return -1;
}
```

## c.2 Common.h

```
#if !defined PEAKF_CONSTANTS
#define  PEAKF_CONSTANTS

/* Error book */
#define ERR_PRG_LRD_RNNNG -1001 //program is already runinng
#define ERR_BRK_DTCTD -1002       // break detected on peakf
#define ERR_MTCN_FLR -1003      // failure in Matcn
#define ERR_NTCN_FLR -1004        // failure in Antcn
#define ERR_SRC_TM   -1005        // didn't reach source in allotted time


/* Constant definition */

#define PROGRAM_NAME "peakf"
#define PROGRAM_NICK "pf"
#define MAX_BUF 80
#define MAX_RETRY 2

static char*DeviceName[] =
{"","v1","v2","v3","v4","v5","v6","v7","v8","v9","va","vb",
"vc","vd","ve","vf","if","if","i3"};

#endif
```

## c.3 Display_fit.c

```
/* display_fit */
// display fit parameters
// params = fit parameters
// eparams = sigma parameters
// chi = reduced chi
// back = fit return code

// {}
#include "peakf.h"
#include "private.h"

void display_fit(name,params,eparams,chi,back,fit)
char *name;
float *params,*eparams,chi;
int back,fit;
{
        char buff[MAX_BUF];
        if (!fit) {
                sprintf(buff,"parabolicfit%3s  %5.3f  %5.3f  %5.3f  %d",name,params[0],params[1],params[2],back);
                loglog(buff);
                sprintf(buff,"err%3s  %5.3f  %5.3f  %5.3f  %5.3f",name,eparams[0],eparams[1],eparams[2],chi);
                loglog(buff);
        }
        else {
                sprintf(buff,"gaussfit%3s  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f
%d",name,params[1],params[2],params[0],params[4],params[3],back);
                loglog(buff);
                sprintf(buff,"err%3s  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f
%5.3f",name,eparams[1],eparams[2],eparams[0],eparams[4],eparams[3],chi);
                loglog(buff);
        }

}
```

## c.4 Display_offset.c

```
/*display_offset*/
// display new and old offsets offsets


#include "peakf.h"
#include "private.h"


// {}

void display_offset(scan,count)
T_Scan *scan;
int count;
{
        char buff[MAX_BUF];
        int i;
        T_AxisPosition Sum;
        for (i=0;i<count;i++) {
                copyAxe(&Sum,&(scan[i].PPosition),scan[i].Axe);
                addAxe(&Sum,scan[i].Diff,scan[i].Axe);
                if (scan[i].Axe!=AXIS_ZS) {
                        sprintf(buff,"offset%3s %.3f  %.3f  %.3f  %.3f  %d",

scan[i].AxisName,scan[i].Elevation,scan[i].Diff,scan[i].NewOffset.AllAxis,Sum.AllAxis,scan[i].FitRes);
                        loglog(buff);
                }
                else {
                        sprintf(buff,"offset%3s %.3f  %.3f  %.3f  %.3f  %.3f  %.3f  %.3f  %.3f  %d",

scan[i].AxisName,scan[i].Elevation,scan[i].Diff,scan[i].NewOffset.ZAxis[0],Sum.ZAxis[0],

        scan[i].NewOffset.ZAxis[1],Sum.ZAxis[1],scan[i].NewOffset.ZAxis[2],Sum.ZAxis[2],scan[i].FitRes);
                        loglog(buff);
                }
        }
}
```

## c.5 Display_sample.c

```
/* display_sample */


// {}
```

```
#include "peakf.h"
#include "private.h"

void display_sample(scan,pos)
T_Scan *scan;
int pos;
{
        char buff[MAX_BUF];
        T_Sample *s;
        s=&(scan->Sample[pos]);
        sprintf(buff,"axis%3s  %d  %.1f  %.3f  %.3f  %.5f",scan->AxisName,pos,s->stm,s->position,s->average,s-
>sigma);
        loglog(buff);
}
```

## c.6 Do_scan.c

```
/*do_scan */

// {}

#include <stdio.h>
#include <math.h>

#include "../../fs/include/pmodel.h"
#include "../include/stparams.h"
#include "../include/stcom.h"

#include "peakf.h"
#include "private.h"
#include "common.h"

#include "./fitlib/fitlib.h"

void fgauss(float x,float a[],float *y,float dyda[],int na);
void fparabola(float x,float a[],float *y,float dyda[],int na);

int do_scan(axis,axisname,count,cmn,mode,ierr)
T_Axis *axis;
char **axisname;
int count;
peakf_cmn *cmn;
T_Mode mode;
int *ierr;
{
        int i,j,app,k,x,npar,imax,go;
        float Step;
        int time[6],ifchain;
        float rut,azimutstep,elevationstep,slope,cnst,chi,tmax,ctmp,diff;
        T_AxisPosition AbsOffset[PEAKF_MAX_NSTEP];
        float offset[PEAKF_MAX_NSTEP];
        float off[PEAKF_MAX_NSTEP];
        float tpi[PEAKF_MAX_NSTEP];
        float sig[PEAKF_MAX_NSTEP];
        float fwhm,dpfu,gain;  // dummies, not used
        T_AxisPosition *Offset,*NewOffset;
        T_Scan scan[MAX_SAMETIME_AXIS];
        float params[5],eparams[5],fittedoffset,NoiseCal;
        char buff[MAX_BUF];
        int res,antennamoved,gaussfit;
        T_Sample TpZero;
        double center;

        //save old offsets and old posistions
        Offset=NewOffset=NULL;
        Offset=(T_AxisPosition *)calloc(count,sizeof(T_AxisPosition));
        NewOffset=(T_AxisPosition *)calloc(count,sizeof(T_AxisPosition));
        if ((Offset==NULL) || (NewOffset==NULL))  {
                *ierr=ERR_NTRNL;
                app=-1;
                goto End;
        }
        // save offset before begging
        for (i=0;i<count;i++) {
                loadAxe(&(Offset[i]),st->Sofs,axis[i],mode);
        }
        // step in millimeters
        Step=cmn->DeltaStep/(float)cmn->NStep;
        // get noise cal for selected device
        if (get_device(cmn->Device,&ifchain,&center,ierr)) goto Error;
        get_gain_par(ifchain,center,&fwhm,&dpfu,&gain,&NoiseCal);
        for (i=0;i<cmn->Repetitions;i++) {
                for (x=0;x<count;x++) {
                        // save antenna coordinates and offsets
                        local (&(scan[x].Azimut),&(scan[x].Elevation),&(scan[x].AzOff),&(scan[x].ElOff));
```

```
                        // drive antenna off source (azimut) and then get background sky;
                        ctmp=cos(scan[x].Elevation*DEG2RAD);
                        azimutstep=OFFSOURCE_OFFSET/ctmp;
                        antennamoved=1;
                        if (go_off(scan[x].AzOff*DEG2RAD+azimutstep,scan[x].ElOff*DEG2RAD,MAX_ANTENNA_WAIT,ierr))
goto Error;
                        if (tzero(ifchain,cmn->Device,cmn->NSamples,&(scan[x].TpZero),0,ierr)) goto Error;
                        // get tpi measure
                        if (get_samples(cmn->Device,cmn->NSamples,ierr,&(scan[x].OffSource),0)) goto Error;
                        // turn noise cal on
                        scmds("calonpf");
                        if (get_samples(cmn->Device,cmn->NSamples,ierr,&(scan[x].TpiNoise),0)) goto Error;
                        // turn noise cal off
                        scmds("caloffpf");
                        scan[x].Slope=NoiseCal/(scan[x].TpiNoise.average-scan[x].OffSource.average);
                        scan[x].OffSource.average=(scan[x].OffSource.average-
scan[x].TpZero.average)*scan[x].Slope;
                        scan[x].OffSource.sigma*=scan[x].Slope;
                        sprintf(buff,"tsys %03.1f  %02.1f  %.4f  %.4f  %.3f
%.3f",scan[x].Azimut,scan[x].Elevation,scan[x].AzOff,scan[x].ElOff,scan[x].OffSource.average,
                          scan[x].OffSource.sigma);
                        loglog(buff);
                        if (go_off(scan[x].AzOff*DEG2RAD,scan[x].ElOff*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto
Error;
                        antennamoved=0;
                        // init scan
                        scan[x].Axe=axis[x];
                        scan[x].Mode=mode;
                        loadAxe(&(scan[x].POffset),st->Sofs,scan[x].Axe,scan[x].Mode);
                        loadAxe(&(scan[x].PPosition),st->Sactpos,scan[x].Axe,scan[x].Mode);
                        scan[x].Samples=0;
                        strcpy(scan[x].AxisName,axisname[x]);
                        scu_record(&(scan[x]));
                        // get start time in seconds from the begging of the day
                        rte_time(time,time+5);
                        rut=time[3]*3600.0+time[2]*60.0+time[1]+((float)time[0])/100.0;

                        // compute absolute position offsets for the axe
                        app=(cmn->NStep/2);
                        for (j=0;j<cmn->NStep;j++) {
                                copyAxe(&(AbsOffset[j]),&(scan[x].POffset),scan[x].Axe);
                                if (j==app) {
                                        addAxe(&(AbsOffset[j]),0.0,scan[x].Axe);
                                        offset[j]=0.0;
                                }
                                else {
                                        addAxe(&(AbsOffset[j]),(float)(j-app)*Step,scan[x].Axe);
                                        offset[j]=(float)(j-app)*Step;
                                }
                        }
                        // start acquisition
                        antennamoved=0;
                        gaussfit=1;
                        for (k=0;k<cmn->NStep;k++) {
                                scan[x].Samples++;
                                // move scu
                                if (go_scu(scan[x].Axe,&(AbsOffset[k]),scan[x].Mode,ierr)) goto Error;
                                if (scan[x].Axe==AXIS_YP) {
                                        elevationstep=offset[k]*ELOFF_4_AXISYP;
                                        if
(go_off(scan[x].AzOff*DEG2RAD,(scan[x].ElOff+elevationstep)*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto Error;
                                        antennamoved=1;
                                        gaussfit=0;
                                }
                                else if (scan[x].Axe==AXIS_YS) {
                                        elevationstep=offset[k]*ELOFF_4_AXISYS;
                                        if
(go_off(scan[x].AzOff*DEG2RAD,(scan[x].ElOff+elevationstep)*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto Error;
                                        antennamoved=1;
                                        gaussfit=0;
                                }
                                else if (scan[x].Axe==AXIS_XS) {
                                        ctmp=cos(scan[x].Elevation*DEG2RAD);
                                        azimutstep=offset[k]*(AZOFF_4_AXISXS/ctmp);
                                        if
(go_off((scan[x].AzOff+azimutstep)*DEG2RAD,scan[x].ElOff*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto Error;
                                        antennamoved=1;
                                        gaussfit=0;
                                }
                                if (get_samples(cmn->Device,cmn->NSamples,ierr,&(scan[x].Sample[k]),rut)) goto
Error;
                                scan[x].Sample[k].position=offset[k];
                                scan[x].Sample[k].average=(scan[x].Sample[k].average-
scan[x].TpZero.average)*scan[x].Slope;
                                scan[x].Sample[k].average-=scan[x].OffSource.average;
                                scan[x].Sample[k].sigma*=scan[x].Slope;
                                display_sample(&(scan[x]),k);
                        }
                        if (antennamoved) {
```

```
                                if (go_off(scan[x].AzOff*DEG2RAD,scan[x].ElOff*DEG2RAD,MAX_ANTENNA_WAIT,ierr))
goto Error;
                        }
                        // compute linear drift
                        unslope(&(scan[x].Sample[0]),&(scan[x].Sample[cmn->NStep-1]),scan[x].Sample,cmn-
>NStep,&slope,&cnst);
                        // fit
                        for (k=0;k<5;k++) {
                                params[i]=eparams[i]=0.0;
                        }
                        tmax=scan[x].Sample[0].average;
                        imax=0;
                        for (k=1;k<cmn->NStep;k++) {
                                if (scan[x].Sample[k].average>tmax) {
                                        tmax=scan[x].Sample[k].average;
                                        imax=k;
                                }
                        }
                        for (k=0;k<cmn->NStep;k++) {
                                off[k]=scan[x].Sample[k].position;
                                tpi[k]=scan[x].Sample[k].average;
                                sig[k]=scan[x].Sample[k].sigma;
                                if (sig[k]==0.0) sig[k]=tmax/1000;
                        }
                        if (gaussfit) {
                                params[1]=scan[x].Sample[imax].position;
                                params[0]=tmax;
                                /*if (cmn->NStep>=5) {
                                        params[4]=cnst;
                                        params[3]=slope;
                                }
                                else
                                {
                                        params[4]=0.0;
                                        params[5]=0.0;
                                }*/
                                params[4]=0.0;
                                params[5]=0.0;
                                params[2]=cmn->DeltaStep;
                                if (cmn->NStep>=5) npar=5;
                                else npar=3;
                                res=NonLinearFit(off,tpi,sig,cmn-
>NStep,params,eparams,npar,MAX_ITER,&chi,fgauss);
                                /*if (cmn->NStep<5) {
                                        params[4]=slope;
                                        params[3]=cnst;
                                }*/
                                /*params[3]+=slope;
                                params[4]+=cnst;*/
                                fittedoffset=params[1];
                        }
                        else {
                                /*params[0]=scan[x].Sample[0].average-slope*scan[x].Sample[0].position-tmax;
                                params[0]/=scan[x].Sample[0].position*scan[x].Sample[0].position;
                                params[1]=slope;
                                params[2]=tmax;*/
                                params[2]=scan[x].Sample[cmn->NStep/2].average;
                                params[0]=(params[2]-
scan[x].Sample[imax].average)/(scan[x].Sample[imax].position*scan[x].Sample[imax].position);
                                if (params[0]>0) params[0]=-params[0];
                                params[1]=-2*params[0]*scan[x].Sample[imax].position;
                                npar=3;
                                res=NonLinearFit(off,tpi,sig,cmn-
>NStep,params,eparams,npar,MAX_ITER,&chi,fparabola);
                                fittedoffset=-(params[1]/(2*params[0]));

        eparams[0]=(eparams[1]*eparams[1])/(params[1]*params[1])+(eparams[0]*eparams[0])/(params[0]*params[0]);
                                eparams[0]/=fittedoffset*fittedoffset;
                                eparams[0]=sqrt(eparams[0]);
                                /*params[2]+=cnst;
                                params[1]+=slope;*/
                                params[0]=fittedoffset;
                        }
                        display_fit(scan[x].AxisName,params,eparams,chi,res,gaussfit);
                        if ((fittedoffset>=scan[x].Sample[0].position) && (fittedoffset<=scan[x].Sample[cmn-
>NStep-1].position) && (res>0)) {
                                setAxe(&(NewOffset[x]),fittedoffset,scan[x].Axe);
                                diff=fittedoffset;
                                scan[x].FitRes=res;
                        }
                        else {
                                setAxe(&(NewOffset[x]),0,scan[x].Axe);
                                diff=0;
                                if (res>0) scan[x].FitRes=-3;
                                else scan[x].FitRes=res;
                        }
                        scan[x].Diff=diff;
                        sumAxis(&(scan[x].NewOffset),&(NewOffset[x]),&(scan[x].POffset),scan[x].Axe);
                        //  use new offset
                        if (go_scu(scan[x].Axe,&(scan[x].NewOffset),scan[x].Mode,ierr)) goto Error;
```

```
                        if ((antennamoved) && (diff!=0.0)) {
                                if (scan[x].Axe==AXIS_YP) {
                                        elevationstep=diff*ELOFF_4_AXISYP;
                                        if
(go_off(scan[x].AzOff*DEG2RAD,(scan[x].ElOff+elevationstep)*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto Error;
                                }
                                else if (scan[x].Axe==AXIS_YS) {
                                        elevationstep=diff*ELOFF_4_AXISYS;
                                        if
(go_off(scan[x].AzOff*DEG2RAD,(scan[x].ElOff+elevationstep)*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto Error;
                                }
                                else if (scan[x].Axe==AXIS_XS) {
                                        ctmp=cos(scan[x].Elevation*DEG2RAD);
                                        azimutstep=diff*(AZOFF_4_AXISXS/ctmp);
                                        if
(go_off((scan[x].AzOff+azimutstep)*DEG2RAD,scan[x].ElOff*DEG2RAD,MAX_ANTENNA_WAIT,ierr)) goto Error;
                                }
                        }
                }
                display_offset(scan,count);

        }
        app=0;
        goto End;
Error:
        for (i=0;i<count;i++) {
                uploadAxe(st->Sofs,&(Offset[i]),axis[i],mode);
        }
        if ((antennamoved) && (*ierr!=ERR_ANTENNA)) {
                go_off(scan[x].AzOff*DEG2RAD,scan[x].ElOff*DEG2RAD,MAX_ANTENNA_WAIT,ierr);
        }
        app=-1;
End:
        if (Offset!=NULL) free(Offset);
        if (NewOffset!=NULL) free(NewOffset);
        return app;
}
```

# c.7 Export.h

```
#if !defined PEAKF_EXPORT
#define PEAKF_EXPORT

// program name

#define PROGRAM_NAME "peakf"
#define PROGRAM_NICK "pf"

#define PEAKF_PARAM_NUMBER 6
#define PEAKF_MAX_NSTEP 19

// default value definition

#define PEAKF_AXIS 2
#define PEAKF_REPETITIONS 2
#define PEAKF_NSTEP 11
#define PEAKF_DELTASTEP 24.0
#define PEAKF_NSAMPLES 5
#define PEAKF_DEVICE DEV_VE

// type definitions

typedef enum {
        DEV_V1=1,
        DEV_V2=2,
        DEV_V3=3,
        DEV_V4=4,
        DEV_V5=5,
        DEV_V6=6,
        DEV_V7=7,
        DEV_V8=8,
        DEV_V9=9,
        DEV_VA=10,
        DEV_VB=11,
        DEV_VC=12,
        DEV_VD=13,
        DEV_VE=14,
        DEV_VF=15,
        DEV_I1=16,
        DEV_I2=17,
        DEV_I3=18
} T_Devices;

typedef struct {
        int Axis;
        int Repetitions;
```

```
        int NStep;
        float DeltaStep;
        int NSamples;
        T_Devices Device;
} peakf_cmn;
```

## c.8 Fgauss.c

```
#include <math.h>

void fgauss(float x,float a[],float *y,float dyda[],int na)
{
        float e,w;

        e=(x-a[2])/a[3];
        w=exp(-2.7725887*e*e);
        *y=a[1]*w+a[4]*x+a[5];
        dyda[1]=w;
        dyda[2]=5.5451774*a[1]*e*w/a[3];
        dyda[3]=5.5451774*a[1]*e*e*w/a[3];
        dyda[4]=x;
        dyda[5]=1.0;
}
```

## c.9 Fparabola.c

```
#include <math.h>

void fparabola(float x,float a[],float *y,float dyda[],int na)
{
        *y=a[1]*x*x+a[2]*x+a[3];
        dyda[1]=x*x;
        dyda[2]=x;
        dyda[3]=1.0;

}
```

## c.10 Get_device.c

```
#include "peakf.h"
#include "private.h"

#include "../../fs/include/dpi.h"
#include "../../fs/include/params.h"
#include "../../fs/include/fs_types.h"
#include "../../fs/include/fscom.h"         /* shared memory definition */
#include "../../fs/include/shm_addr.h"      /* shared memory pointer */


int get_device(device,ifchain,center,ierr)
T_Devices device;
int *ifchain;
double *center;
int *ierr;
{
        float vcf,vcbw,extbw,upper;
        int i;
        float bw[]={0.0,0.125,0.250,0.50,1.0,2.0,4.0};
        float bw4[ ]={0.0,0.125,16.0,0.50,8.0,2.0,4.0};
        i=device-1;
        *ierr=0;
        if(shm_addr->equip.rack==MK3||shm_addr->equip.rack==MK4) {
                if (device<DEV_VF) {
                        i=device-1;
                        *ifchain=abs(shm_addr->ifp2vc[i]);
                        if((*ifchain)<1||(*ifchain)>3) *ifchain=0;
                        if(*ifchain!=0) {
                                vcf=shm_addr->freqvc[i];
                        if(shm_addr->ibwvc[i]==0) vcbw=shm_addr->extbwvc[i];
                        else if(shm_addr->equip.rack==MK3) vcbw=bw[shm_addr->ibwvc[i]];
                        else if(shm_addr->equip.rack==MK4) vcbw=bw4[shm_addr->ibwvc[i]];
                        switch (shm_addr->ITPIVC[i]&0x7) {
                                        case 0: /* dual */
                                                vcf=vcf;
                                                break;
                                        case 1: /* lsb */
                                                vcf=vcf-vcbw*0.5;
                                                break;
                                        case 2: /* usb */
```

```
                                        vcf=vcf-vcbw*0.5;
                                        break;
                                default:
                                        *ierr=ERR_NTRNL;
                                        return -1;
                                        break;
                        }
                        switch(shm_addr->lo.sideband[(*ifchain)-1]) {
                                case 1:
                                        *center=shm_addr->lo.lo[(*ifchain)-1]+vcf;
                                        break;
                                case 2:
                                        (*center)=shm_addr->lo.lo[(*ifchain)-1]-vcf;
                                        break;
                                default:
                                        *ierr=ERR_NTRNL;
                                        return -1;
                                        break;
                        }
                        } else {
                                *ierr=ERR_NTRNL;
                                return -1;
                        }
                }
                else if ((device==DEV_I1) || (device==DEV_I2)) {
                        *ifchain=device-15;
                        switch (shm_addr->lo.sideband[(*ifchain)-1]) {
                                case 1:
                                        *center=shm_addr->lo.lo[(*ifchain)-1]+(500.+100.);
                                        break;
                                case 2:
                                        *center= shm_addr->lo.lo[(*ifchain)-1]-(500.+100.)*0.5;
                                        break;
                                default:
                                        *ierr=ERR_NTRNL;
                                        return -1;
                                        break;
                        }
                }
                else if (device==DEV_I3) {
                        *ifchain=device-15;
                        if(shm_addr->imixif3==1) upper=400.0;
                        else upper=500.0;
                        switch (shm_addr->lo.sideband[(*ifchain)-1]) {
                                case 1:
                                        *center=shm_addr->lo.lo[(*ifchain)-1]+(upper+100.)*0.5;
                                        break;
                                case 2:
                                        *center=shm_addr->lo.lo[(*ifchain)-1]-(upper+100.)*0.5;
                                        break;
                                default:
                                        *ierr=ERR_NTRNL;
                                        return -1;
                                        break;
                        }
                }
                else {
                        *ierr=ERR_NTRNL;
                        return -1;
                }
        }
        else {
                *ierr=ERR_NTRNL;
                return -1;
        }
        return 0;
}
```

# c.11 Get_gain_par.c

```
#include <stdio.h>
#include <sys/types.h>
#include <math.h>

#include "../../fs/include/dpi.h"
#include "../../fs/include/params.h"
#include "../../fs/include/fs_types.h"
#include "../../fs/include/fscom.h"         /* shared memory definition */
#include "../../fs/include/shm_addr.h"      /* shared memory pointer */

void get_gain_par(ifchain,center,fwhm,dpfu,gain,tcal)
     int ifchain;
     double center;
     float *fwhm, *tcal, *dpfu, *gain;
{
  int i, ir, it[6], ifirst, ilast;
```

```
double az,el, arg;

*fwhm=0.0;
*dpfu=0.0;
*tcal=0.0;
if(gain!=NULL)
  *gain=0.0;

if(ifchain <1 || 4 < ifchain)
  return;

ir=-1;
for(i=0;i<MAX_RXGAIN;i++) {
  if(shm_addr->rxgain[i].type=='f'
     && ((fabs(shm_addr->lo.lo[ifchain-1]-shm_addr->rxgain[i].lo[0])
          < 0.001)
         ||(shm_addr->rxgain[i].lo[1] > 0.0
            && fabs(shm_addr->lo.lo[ifchain-1]-shm_addr->rxgain[i].lo[1])
            < 0.001))
     ) {
    ir=i;
  }
}
if(ir==-1)
  for(i=0;i<MAX_RXGAIN;i++) {
    if(shm_addr->rxgain[i].type=='r'
       && shm_addr->lo.lo[ifchain-1]>shm_addr->rxgain[i].lo[0]-0.001
       && shm_addr->lo.lo[ifchain-1]<shm_addr->rxgain[i].lo[1]+0.001) {
      ir=i;
    }
  }

if(ir==-1)
  return;

if(shm_addr->rxgain[ir].fwhm.model=='c')
  *fwhm=shm_addr->rxgain[ir].fwhm.coeff;
else if(shm_addr->rxgain[ir].fwhm.model=='f'
        && center*1e6*shm_addr->diaman > 1e-12) {
  *fwhm=1.22*299792458.0e0/(center*1e6*shm_addr->diaman);
}

if(gain != NULL) {
  rte_time(it,it+5);
  cnvrt2(1,shm_addr->radat,shm_addr->decdat,&az,&el,it,0.0,shm_addr->alat,
         shm_addr->wlong);

  arg=el*RAD2DEG;
  if(shm_addr->rxgain[ir].gain.form=='a')
    arg=90.0-arg;

  *gain=0.0;
  for(i=shm_addr->rxgain[ir].gain.ncoeff-1;i>-1;i--) {
    *gain=*gain*arg+shm_addr->rxgain[ir].gain.coeff[i];
  }
}


switch(shm_addr->lo.pol[ifchain-1]) {
case 1:
  /* ifchain is r */
  if(shm_addr->rxgain[ir].pol[0]=='r') {
    *dpfu=shm_addr->rxgain[ir].dpfu[0];
  } else if(shm_addr->rxgain[ir].pol[1]=='r') {
    *dpfu=shm_addr->rxgain[ir].dpfu[1];
  }
  ifirst=-1;
  ilast=-1;
  for(i=0;i<shm_addr->rxgain[ir].tcal_ntable;i++)
    if(shm_addr->rxgain[ir].tcal[i].pol=='r' &&
       shm_addr->rxgain[ir].tcal[i].freq < center)
      ifirst=i;
    else if(shm_addr->rxgain[ir].tcal[i].pol=='r' &&
            center < shm_addr->rxgain[ir].tcal[i].freq) {
      ilast=i;
      goto interpr;
    }
interpr:
  if(ifirst==-1 && ilast != -1)
    *tcal=shm_addr->rxgain[ir].tcal[ilast].tcal;
  else if(ifirst!=-1 && ilast ==-1)
    *tcal=shm_addr->rxgain[ir].tcal[ifirst].tcal;
  else if(ifirst!=-1 && ilast!=-1)
    *tcal=shm_addr->rxgain[ir].tcal[ifirst].tcal+
      (center-shm_addr->rxgain[ir].tcal[ifirst].freq)*
      (shm_addr->rxgain[ir].tcal[ilast].tcal-
       shm_addr->rxgain[ir].tcal[ifirst].tcal)/
      (shm_addr->rxgain[ir].tcal[ilast].freq-
       shm_addr->rxgain[ir].tcal[ifirst].freq);
  break;
```

```
  case 2:
    if(shm_addr->rxgain[ir].pol[0]=='l') {
      *dpfu=shm_addr->rxgain[ir].dpfu[0];
    } else if(shm_addr->rxgain[ir].pol[1]=='l') {
      *dpfu=shm_addr->rxgain[ir].dpfu[1];
    }
    ifirst=-1;
    ilast=-1;
    for(i=0;i<shm_addr->rxgain[ir].tcal_ntable;i++)
      if(shm_addr->rxgain[ir].tcal[i].pol=='l' &&
          shm_addr->rxgain[ir].tcal[i].freq < center)
        ifirst=i;
      else if(shm_addr->rxgain[ir].tcal[i].pol=='l' &&
              center < shm_addr->rxgain[ir].tcal[i].freq) {
        ilast=i;
        goto interpl;
      }
  interpl:
    if(ifirst==-1 && ilast != -1)
      *tcal=shm_addr->rxgain[ir].tcal[ilast].tcal;
    else if(ifirst!=-1 && ilast ==-1)
      *tcal=shm_addr->rxgain[ir].tcal[ifirst].tcal;
    else if(ifirst!=-1 && ilast!=-1)
      *tcal=shm_addr->rxgain[ir].tcal[ifirst].tcal+
         (center-shm_addr->rxgain[ir].tcal[ifirst].freq)*
         (shm_addr->rxgain[ir].tcal[ilast].tcal-
          shm_addr->rxgain[ir].tcal[ifirst].tcal)/
         (shm_addr->rxgain[ir].tcal[ilast].freq-
          shm_addr->rxgain[ir].tcal[ifirst].freq);
    break;
  default:
    break;
  }
}
```

# c.12 Get_samples.c

```
/*get_samples */
/*gets N samples of tpi from the specified device*/
/*returns -1 if an error occured*/
/*ip : class */
/*device : used device */
/*samples : numebr of samples */
/*ierr : error code*/
/*sample : average, sigma obtained by sampling samples times the device*/
/*rut : time (in seconds) at the begging of sampling  */

#include "../../fs/include/fs_types.h"
#include <math.h>

#include "peakf.h"
#include "private.h"

// {}

int get_samples(device,samples,ierr,sample,rut)
T_Devices device;
int samples,*ierr;
T_Sample *sample;
float rut;
{
        int now[6],time[6],elapsed;
        int i,retry;
        long ip[5];
        int pwr,dummy;
        float tpi,app,stm;
        if (brk_chk(PROGRAM_NAME)!=0) {
                *ierr=ERR_BRK_DTCTD;
                return -1;
        }
        rte_sleep(101);
        rte_time(now,now+5);
        sample->average=sample->sigma=sample->stm=0.0;
        retry=0;
        for(i=0;i<samples;i++) {
                if (brk_chk(PROGRAM_NAME)!=0) {
                        *ierr=ERR_BRK_DTCTD;
                        return -1;
                }
                rte_time(time,time+5);
                elapsed=(time[1]-now[1])*100+time[0]-now[0]+101;
                if (elapsed<0) elapsed+=6000;
                if (elapsed>0) rte_sleep(elapsed);
                // an error has occured in Matcn, unrecoverable, then exit
                if (tpi_request(ip,device,ierr)) return -1;
                rte_time(now,now+5);
```

```
                stm=now[3]*3600.0+now[2]*60.0+now[1]+((float)now[0])/100.0;
                if (stm<rut) stm+=86400.0;
                if ((dummy=tpi_get(device,ip,&pwr))<0) {  // something went wrong
                        if (retry<MAX_RETRY) {
                                // repeat measure !!!!
                                i--;
                                retry++;
                        }
                        else { //if we tried enough...give up and declare error
                                *ierr=ERR_MTCN_FLR;
                                return -1;
                        }
                }
                else {

                        retry=0;
                        if (dummy==1) { //device is in overflow
                                tpi=1e9;
                        }
                        else {
                                tpi=(float)pwr;
                        }
                        sample->average+=tpi/(float)samples;
                        sample->sigma+=(tpi*tpi)/(float)samples;
                        sample->stm+=(stm-rut)/(float)samples;
                }
        }
        if (samples>1) {
                tpi=sample->sigma-(sample->average*sample->average);
                if (tpi<0.0) {
                        sample->sigma=0.0;
                }
                else {
                        app=((float)samples)/((float)(samples-1));
                        sample->sigma=sqrt(fabs(tpi)*app)/sqrt((float)(samples-1));
                }
        }
        else {
                sample->sigma=0.33;
        }
        return 0;
}
```

## c.13 Go_off.c

```
/* go_off */
/* this function drives the antenna to specified offsets */
/* return 0 if no error was found, otherwise -1 */
/* azoff : azimuth  offset */
/* eloff : elevation offset */
/* nwait : second to wait for the antenna to arrive */
/* ierr : error code */

// {}

#include "../../fs/include/dpi.h"
#include "../../fs/include/params.h"       /* FS parameters            */
#include "../../fs/include/fs_types.h" /* FS header files        */
#include "../../fs/include/fscom.h"        /* FS shared mem. structure */
#include "peakf.h"
#include "private.h"

extern struct fscom *shm_addr;

int go_off(azoff,eloff,nwait,ierr)
double azoff,eloff;
int nwait,*ierr;
{
        shm_addr->AZOFF=azoff;
        shm_addr->ELOFF=eloff;
        if(antcn(2,ierr))
                return ERR_ANTENNA;

        if(onsor(nwait,ierr))
                return ERR_ANTENNA;

        return 0;

}
```

## c.14 Go_scu.c

```
/* go_scu */
```

```
/* send one subreflector axis toward a specified position by selecting an offset */
/* Also it waits for a time which is supposed to be the time the subreflector will take to reach commanded
posistion */
/* This functions performs soma check of the scu status, if everything was ok returned value is 0*/
/* Ax : subreflector axis */
/* offset : offset to apply */

#include "../../fs/include/fs_types.h"
#include "../../fs/include/pmodel.h"
#include "../include/stparams.h"
#include "../include/stcom.h"

#include <math.h>

#include "peakf.h"
#include "private.h"
#include "common.h"

// {}

int go_scu(Ax,offset,mode,ierr)
T_Axis Ax;
T_AxisPosition *offset;
T_Mode mode;
int *ierr;
{
        long ip[5] = {0,0,0,0,0};
        double app;
        long wait;
        int i,j;
        int check;
        T_AxisPosition currentoff;
        if (brk_chk(PROGRAM_NAME)!=0) {
                *ierr=ERR_BRK_DTCTD;
                return -1;
        }
        if (!st->sculink) {   // check for link
                *ierr=ERR_NO_SC_LNK;
                return -1;
        }
        if (st->scust.bit0==0) { // if scu is not ready
                check=1;                // check position after command
        }
        else {
                check=0;
        }
        *ierr=0;
        loadAxe(&currentoff,st->Sofs,Ax,mode);
        uploadAxe(st->Sofs,offset,Ax,mode);

        if (Ax==AXIS_ZS) {
                wait=(fabs(currentoff.ZAxis[0]-offset->ZAxis[0])*(double)SecondaryDelays[AXIS_Z1S-mode]);
                app=(fabs(currentoff.ZAxis[1]-offset->ZAxis[1])*(double)SecondaryDelays[AXIS_Z2S-mode]);
                if (wait<app) wait=app;
                app=(fabs(currentoff.ZAxis[2]-offset->ZAxis[2])*(double)SecondaryDelays[AXIS_Z3S-mode]);
                if (wait<app) wait=app;
                wait+=150;
        }
        else {
                if (mode==MODE_PRIMARY) {
                        wait=(long)(fabs(currentoff.AllAxis-offset->AllAxis)*(double)PrimaryDelays[Ax-mode]);
                }
                else {
                        wait=(long)(fabs(currentoff.AllAxis-offset->AllAxis)*(double)SecondaryDelays[Ax-mode]);
                }
        }
        wait*=2;
        wait+=150;
        ip[1]=803;
        skd_run("stqkr",'w',ip);
        rte_sleep(wait);          // wait for the scu to arrive
        if (check) {
                for (i=0;i<MAX_RETRY;i++) {
                        if (brk_chk(PROGRAM_NAME)!=0) {
                                *ierr=ERR_BRK_DTCTD;
                                return -1;
                        }
                        rte_sleep(wait);    // wait
                        for(j=0;j<5;j++) ip[j]=0;
                        ip[1]=803;
                        skd_run("stqkr",'w',ip);   // read status
                        if (st->scust.bit0==1) {
                                check=0;
                                break;
                        }
                }
                if (check) {
                        *ierr=ERR_SC_FLR;
                        uploadAxe(st->Sofs,&currentoff,Ax,mode);
                        return -1;
```

```
                }
        }
        return 0;
}
```

## c.15 Local.c

```
/* local */
/* get actual antenna elevation and azimuth */
/* az, el : antenna coordinate azimut and elevation in degree */
/* azoff, eloff : antenna offsets */
/* ierr : error code */

#include <math.h>

//#include "../../fs/include/dpi.h"
//#include "../../fs/include/params.h"     /* FS parameters            */
//#include "../../fs/include/fs_types.h" /* FS header files           */
//#include "../../fs/include/fscom.h"      /* FS shared mem. structure */

#include "../../fs/include/pmodel.h"
#include "../include/stparams.h"
#include "../include/stcom.h"

//extern struct fscom *shm_addr;
#include "common.h"

//double refrw();
// {}

void local(azimut,elevation,azoff,eloff)
double *azimut,*elevation;
double *azoff,*eloff;
{
        //int it[6];
        //double az,el,x,y;
        //double tmp;

        //rte_time(it,it+5);
        // convert from Ra/Dec to Az/El
        //cnvrt2(1,shm_addr->radat,shm_addr->decdat,&az,&el,it,0.0,shm_addr->alat,shm_addr->wlong);
        // add refraction model
        //el+=DEG2RAD*refrw(el,20.0,50.0,950.0);
        //cnvrt2(5,az,el,&x,&y,it,0.0,shm_addr->alat,shm_addr->wlong);
        //cnvrt2(4,x,y,azimut,elevation,it,0.0,shm_addr->alat,shm_addr->wlong);
        *elevation=(st->elq/10000.0);
        *azimut=(st->azq/10000.0);
        *azoff=st->azoff*RAD2DEG;
        *eloff=st->eloff*RAD2DEG;
        //*azoff=shm_addr->AZOFF;
        //*eloff=shm_addr->ELOFF;
}
```

## c.16 Log.c

```
/* log */
/* Put into the field system log a buffer of characters */
/* buff : message to display into log */

#include "peakf.h"
#include "private.h"


void loglog(buff)
char *buff;
{
        char sb[MAX_BUF];
        //strcpy(sb,"/");
        //strcat(sb,PROGRAM_NAME);
        //strcat(sb,"/");
        strcpy(sb,buff);
        logita(sb,0,PROGRAM_NICK,"");
}
```

## c.17 Mat.c

```
/* Mat */
/* this function send to  matcn a buffer with a request. return 0 if no error*/
/* occured */
```

```
/* ip : class returned by Matcn as aswer*/
/* iclass : class numeber */
/* nrec : numebr of record in the class */
/* ierr : error code, 0 means no error */

#include <stdio.h>
#include "peakf.h"
#include "private.h"

int mat(ip,iclass,nrec,ierr)
long ip[5];
int iclass, nrec, *ierr;
{
        // check if a break has been issued
        if (brk_chk(PROGRAM_NAME)!=0) {
                *ierr=ERR_BRK_DTCTD;
                return -1;
        }
        *ierr=0;
        ip[0]=iclass;
        ip[1]=nrec;
        skd_run("matcn",'w',ip);
        skd_par(ip);
        if (ip[2]>=0) return 0;
        // error occured in Matcn
        if (ip[1]!=0) {
                cls_clr(ip[0]);
                ip[0]=ip[1]=0;
        }
        *ierr=ERR_MTCN_FLR;
        return -1;

}
```

# c.18 Onsor.c

```
/* This function wait for the telescope to come on source /*
/* It will fail if the onsource condition wouldn't hapen within nwait seconds */
/* return -1 if it fails 0 otherwise */
/* Thaks you, Ed for this function */

#include <signal.h>
#include <math.h>
#include <stdio.h>

#include "../../fs/include/dpi.h"
#include "../../fs/include/params.h"
#include "../../fs/include/fs_types.h"
#include "../../fs/include/fscom.h"

#include "peakf.h"
#include "private.h"

extern struct fscom *shm_addr;

int onsor(nwait,ierr)
     int nwait,*ierr;
{
        int it[6];
        double tim,tim2;

        rte_time(it,it+5);
        tim=it[3]*3600.0+it[2]*60.0+it[1]+((double)it[0])/100.;

        rte_time(it,it+5);
        tim2=it[3]*3600.0+it[2]*60.0+it[1]+((double)it[0])/100.;
        if(tim2<tim) tim2+=86400.0;

        while(tim+nwait>tim2) {
                if (brk_chk(PROGRAM_NAME)!=0) {
                *ierr=ERR_BRK_DTCTD;
                return -1;
        }

                if(antcn(5,ierr)) return -1;
                if(shm_addr->ionsor!=0) return 0;

                rte_time(it,it+5);
                tim2=it[3]*3600.0+it[2]*60.0+it[1]+((double)it[0])/100.;
                if(tim2<tim) tim2+=86400.0;
        }

        *ierr=ERR_SRC_TM;
        return -1;
}
```

## c.19 Peakf.c

```
/* peakf.c */

/* Include files */

#include <stdio.h>
#include "../../fs/include/params.h"       /* FS parameters            */
#include "../../fs/include/fs_types.h" /* FS header files        */
#include "../../fs/include/fscom.h"        /* FS shared mem. structure */
#include "../../fs/include/shm_addr.h" /* FS shared mem. pointer */
#include "../../fs/include/pmodel.h"
#include "../include/stm_addr.h"
#include "../include/stparams.h"
#include "../include/stcom.h"

#include "peakf.h"
#include "private.h"


// {}

struct fscom *fs;
struct stcom *st;
peakf_cmn *cmn;

main()
{
        long ip[5];
        char buf[MAX_BUF];
        int ierr,i;
        int time[6];
        double rut;
        T_Mode mode;
        T_Axis axe[3];  //max logical axis
        T_AxisPosition OldOffset[3];
        T_AxisPosition OldPosition[3];
        int axecounter,len;
        char *axisname[MAX_SAMETIME_AXIS];

/* Put our program name where logit can find it. */
        putpname(PROGRAM_NAME);
        setup_ids();
        setup_st();

Cont:
        // Return to this point to wait until we are called again
        skd_wait(PROGRAM_NAME,ip,(unsigned)0);

        // clear class
        // Set up IDs for shared memory, then assign the pointer to
        // fs, for readability.
        fs=shm_addr;
        st=stm_addr;

        if (nsem_take(PROGRAM_NAME,1)==1) {
                // peakf can't get semaphore
                ierr=ERR_PRG_LRD_RNNNG;
                goto Error;
        }
        cmn=&(st->peakfcommon);

        //wait for telescope to reach source
        if (onsor(MAX_ANTENNA_WAIT,&ierr)) goto Error;

        //*********************************************************************
        // controlla che il subriflettore sia fermo in una posizione consistente.

        for (i=0;i<3;i++) axisname[i]=(char *)malloc(3*sizeof(char));
        if (st->Sactmode==0) mode=MODE_PRIMARY;
        else mode=MODE_SECONDARY;

        //check which axis must be used
        axecounter=0;
        len=strlen(AxisConfig[cmn->Axis]);
        if (mode==MODE_PRIMARY) {
                for(i=0;i<len;i++) {
                        //if (strstr(AxisConfig[cmn->Axis],"y")!=0) {
                        if (AxisConfig[cmn->Axis][i]=='y') {
                                axe[axecounter]=AXIS_YP;
                                strcpy(axisname[axecounter],"yP");
                                axecounter++;
                        }
                        else {
                        //if (strstr(AxisConfig[cmn->Axis],"z")!=0) {
                                axe[axecounter]=AXIS_ZP;
                                strcpy(axisname[axecounter],"zP");
                                axecounter++;
```

```
                        }
                }
        }
        else {
                for(i=0;i<len;i++) {
                        if (AxisConfig[cmn->Axis][i]=='x') {
                                axe[axecounter]=AXIS_XS;
                                strcpy(axisname[axecounter],"xS");
                                axecounter++;
                        }
                        else if (AxisConfig[cmn->Axis][i]=='y') {
                                axe[axecounter]=AXIS_YS;
                                strcpy(axisname[axecounter],"yS");
                                axecounter++;
                        }
                        else if (AxisConfig[cmn->Axis][i]=='z') {
                                axe[axecounter]=AXIS_ZS;
                                strcpy(axisname[axecounter],"zS");
                                axecounter++;
                        }
                }
                // z1,z2,z3 can't be used together other axis
                if (strstr(AxisConfig[cmn->Axis],"z1")!=0) {
                        axe[axecounter]=AXIS_Z1S;
                        strcpy(axisname[axecounter],"z1S");
                        axecounter++;
                }
                else if (strstr(AxisConfig[cmn->Axis],"z2")!=0) {
                        axe[axecounter]=AXIS_Z2S;
                        strcpy(axisname[axecounter],"z2S");
                        axecounter++;
                }
                else if (strstr(AxisConfig[cmn->Axis],"z3")!=0) {
                        axe[axecounter]=AXIS_Z3S;
                        strcpy(axisname[axecounter],"z3S");
                        axecounter++;
                }
        }
        if (axecounter==0) {
                ierr=ERR_WRNG_PRMTR;
                goto Error;
        }


        // get start time in seconds from the begging of the day
        source_record();
        peakf_record(cmn);
        if (do_scan(axe,axisname,axecounter,cmn,mode,&ierr)) goto Error;
        for (i=0;i<3;i++) free(axisname[i]);

        goto End;
Error:
        logita(NULL,ierr,PROGRAM_NICK,"");
End:
        nsem_put(PROGRAM_NAME);
        goto Cont;
}
```

# c.20 Peakf.h

```
#if !defined PEAKF_HEADER
#define PEAKF_HEADER

#include "export.h"

// symbol

#define PEAKF_DEVICENAME_SIZE 19
#define PEAKF_AXISCONFIG_SIZE 30

static char *DeviceName[] =
{"","v1","v2","v3","v4","v5","v6","v7","v8","v9","va","vb",
"vc","vd","ve","vf","i1","i2","i3"};

// allowed axis configurations
#define MAX_SAMETIME_AXIS 3
static char *AxisConfig[] =
{"","y","z","yz","zy","","","","","","",                    // primary allowed modes
 "","x","y","z","xy","yx","xz","zx","yz","zy","xyz","xzy",
    "yxz","yzx","zxy","zyx","z1","z2","z3" };               // secondary allowed modes

#endif
```

## c.21 Peakf_record.c

```
/* peakf_record */

// put into log peakf running parameters

// {}

#include "peakf.h"
#include "private.h"


void  peakf_record(peakf_cmn *c)
{
        char buff[MAX_BUF];
        sprintf(buff,"parameters  %3s  %2d  %2d  %3.2f  %2d  %2s",AxisConfig[c->Axis],c->Repetitions,
          c->NStep,c->DeltaStep,c->NSamples,DeviceName[c->Device]);
        loglog(buff);

}
```

## c.22 Private.h

```
#if !defined PEAKF_PRIVATE
#define  PEAKF_PRIVATE

/* Error book */
#define ERR_PRG_LRD_RNNNG -1001 //program is already runninng
#define ERR_BRK_DTCTD -1002       // break detected on peakf
#define ERR_MTCN_FLR -1003      // failure in Matcn
#define ERR_NTCN_FLR -1004        // failure in Antcn
#define ERR_SRC_TM  -1005 // didn't reach source in allotted time
#define ERR_WRNG_PRMTR -1006    // paramter is wrong or out of range
#define ERR_NO_SC_LNK -1007       // no scu link
#define ERR_SC_FLR -1008  // failure in Scu
#define ERR_NTRNL -1009           // error in program
#define ERR_ANTENNA -1010 // error in Antenna
/* Constant definition */

#define MAX_BUF 128
#define MAX_RETRY 2
#define MAX_ANTENNA_WAIT 450              //second to wait for Antenna
#define CUTOFF_ELEVATION 75               // cutoff elevation;
#define OFFSOURCE_OFFSET 0.017453       // radians, offest to move antenna offsource
#define ELOFF_4_AXISYP 0.0044             // degree per millimeters elevation offset for YP axe offset
#define ELOFF_4_AXISYS -0.0037            // degree per millimeters elevation offset for YS axe offset
#define AZOFF_4_AXISXS -0.0037            // degree(*cos el) per millimeters azimuth offset for XS axe offset
#define MAX_ITER 50
#define MAX_AXISNAMES_LENGTH 3            // used for strings that contain axis names : zP, zS, z1S,...

typedef enum {
        MODE_PRIMARY=0,
        MODE_SECONDARY=10
} T_Mode;

typedef enum {
        AXIS_YP=1,
        AXIS_ZP=2,
        AXIS_XS=11,
        AXIS_YS=12,
        AXIS_Z1S=13,
        AXIS_Z2S=14,
        AXIS_Z3S=15,
        AXIS_ZS=16
} T_Axis;

typedef union {
        float AllAxis;
        float ZAxis[3];
} T_AxisPosition;

typedef struct {
        float average;
        float sigma;
        float stm;
        float position;
} T_Sample;
```

## c.23 scmds.c

```
/* scmds */
```

```
/* This function permits to inject a snap command into boss command stack */
/* It also wait for the snap procedure to finish, then the called snap procedure must call a sy=go peakf & */
#include <signal.h>
#include <math.h>
#include <stdio.h>

#include "../../fs/include/dpi.h"
#include "../../fs/include/params.h"
#include "../../fs/include/fs_types.h"
#include "../../fs/include/fscom.h"
#include "peakf.h"

extern struct fscom *shm_addr;

void scmds(mess)
     char *mess;
{
  long ip[5];

  cls_snd( &(shm_addr->iclopr), mess, strlen(mess) , 0, 0);
  skd_run("boss ",'n',ip);

  go_take(PROGRAM_NAME,0);

  return;

}
```

# c.24 Scu_record.c

```
/* scu_record */
// log the scu data before starting peakf process

// {}

#include "peakf.h"
#include "private.h"


void scu_record(scan)
T_Scan *scan;
{
         char buff[MAX_BUF];
         if (scan->Axe==AXIS_ZS) {
                  sprintf(buff,"scu%3s  %.3f  %.3f  %.3f  %.3f  %.3f  %.3f",
                  scan->AxisName,scan->PPosition.ZAxis[0],scan->POffset.ZAxis[0],
                  scan->PPosition.ZAxis[1],scan->POffset.ZAxis[1],
                  scan->PPosition.ZAxis[2],scan->POffset.ZAxis[2]);
         }
         else {
                  sprintf(buff,"scu  %3s  %.3f  %.3f",scan->AxisName,scan->PPosition.AllAxis,scan-
>POffset.AllAxis);
         }
         loglog(buff);

}
```

# c.25 Source_record.c

```
/* source_record /*
// write source name and coordinates */

// {}

#include <math.h>

#include "../../fs/include/dpi.h"
#include "../../fs/include/params.h"       /* FS parameters            */
#include "../../fs/include/fs_types.h" /* FS header files           */
#include "../../fs/include/fscom.h"        /* FS shared mem. structure */

#include "peakf.h"
#include "private.h"

extern struct fscom *fs;

void source_record()
{
         double Ra,Dec;
         long h,m,s,ss;
         double ra1,ra2,ra3;
         char buff[MAX_BUF],ras[MAX_BUF],decs[MAX_BUF],buff1[MAX_BUF],sign;
         Ra=fs->ra50*RAD2SEC*10.0;
```

```
        Dec=fabs(fs->dec50)*RAD2DEG*3600.0;
        if (fs->dec50<0) sign='-';
        else sign='+';
        // convert ra
        h=(int)Ra/36000.0;
        ra1=Ra-36000.0*(double)h;
        m=(int)ra1/600.0;
        ra2=ra1-600.0*(double)m;
        s=(int)ra2/10.0;
        ra3=ra2-10.0*(double)s;
        ss=(int)ra3+0.5;
        if (ss<0) ss=0;
        if (ss>9) ss=9;
        sprintf(ras,"%02d:%02d:%02d.%01d  ",h,m,s,ss);
        //convert dec
        h=(int)Dec/3600.0;
        ra1=Dec-3600.0*(double)h;
        m=(int)ra1/60.0;
        ra2=ra1-60.0*(double)m;
        s=(int)ra2+0.5;
        if (s<0) s=0;
        if (s>59) s=59;
        sprintf(decs,"%c%02d:%02d:%02d  ",sign,h,m,s);
        memcpy(buff,fs->lsorna,10);
        buff[10]=0;
        strcat(buff,"  ");
        strcat(buff,ras);
        strcat(buff,decs);
        sprintf(ras,"%4.1f",fs->ep1950);
        strcat(buff,ras);
        strcpy(buff1,"source  ");
        strcat(buff1,buff);
        loglog(buff1);
}
```

# c.26 Tpi_get.c

```
/* tpi_get */
/* This function parses a Matcn response buffer and return the tpi measure */
/* device : device to read */
/* ip : class received from Matcn */
/* pwr : tpi */
/* if the pwr value is consistent this function return 0, if 1 overflow was */ /*found, if a negative error is
returned thi value is not correct */

#include "../../fs/include/params.h"

#include "peakf.h"
#include "private.h"

// {}

int tpi_get(device,ip,pwr)
T_Devices device;
long ip[5];
int* pwr;
{
        int i,nrec,iclass;
        int rd;
        int ierr,dummy,nchar;
        char buf[MAX_BUF];
        nrec=ip[1];   // number of record in the class
        iclass=ip[0];  // class number
        *pwr=0;
        i=0;
        while (i<nrec) {
                cls_rcv(iclass,&ierr,sizeof(ierr),&dummy,&dummy);
                nchar=cls_rcv(iclass,buf,MAX_BUF,&dummy,&dummy);
                if (ierr>=0) {
                        if (sscanf(buf+(device!=DEV_I2?6:2),"%4x",&rd)==1) {
                                if (rd>=0xffff) {
                                        cls_clr(ip[0]);
                                        ip[0]=ip[1]=0;
                                        return 1; // overflow
                                }
                                else {
                                        *pwr=rd;
                                        cls_clr(ip[0]);
                                        ip[0]=ip[1]=0;
                                        return 0;
                                }
                        }
                        else {
                                cls_clr(ip[0]);
                                ip[0]=ip[1]=0;
                                return -200; //cannot parse received buffer
```

```
                        }
                }
                else {
                        cls_clr(ip[0]);
                        ip[0]=ip[1]=0;
                        return ierr;  // some error occured in Matcn
                }
                i+=2;
        }
        cls_clr(ip[0]);
        ip[0]=ip[1]=0;
        return -100;
}
```

## c.27 Tpi_request.c

```
/* tpi_reqest */
/* sends a request ot matcn  to retrive the tpi of the specified device */
/* ip : class from Matcn */
/* Device : specified device */
/* ierr : error code */


#include "../../fs/include/params.h"

#include "peakf.h"
#include "private.h"


int tpi_request(ip,Device,ierr)
long ip[5];
T_Devices Device;
int *ierr;
{
        int iclass;
        short int Buff[MAX_BUF];
        iclass=0;
        if (Device<=DEV_VF) {
                Buff[0]=-22;
                memcpy(Buff+1,DeviceName[Device],2);
        }
        else if ((Device==DEV_I1) || (Device==DEV_I2)) {
                memcpy(Buff+1,"if",2);
                Buff[0]=-21;
        }
        else {
                memcpy(Buff+1,DeviceName[DEV_I3],2);
                Buff[0]=-22;
        }
        cls_snd(&iclass,Buff,4,0,0);
        if (mat(ip,iclass,1,ierr)) return -1;
        else return 0;
}
```

## c.28 Types.h

```
#if !defined PEAKF_TYPES
#define PEAKF_TYPES

typedef enum {
        DEV_V1=1,
        DEV_V2=2,
        DEV_V3=3,
        DEV_V4=4,
        DEV_V5=5,
        DEV_V6=6,
        DEV_V7=7,
        DEV_V8=8,
        DEV_V9=9,
        DEV_VA=10,
        DEV_VB=11,
        DEV_VC=12,
        DEV_VD=13,
        DEV_VE=14,
        DEV_VF=15,
        DEV_I1=16,
        DEV_I2=17,
```

```
        DEV_I3=18
} T_Devices;

typedef enum {
        AXIS_X=1,
        AXIS_Y=2,
        AXIS_Z=4,
        AXIS_Z1=8,
        AXIS_Z2=16,
        AXIS_Z3=32
} T_Axis;

typedef struct {
        char Axis[7];
        int Repetitions;
        int NStep;
        float DeltaStep;
        int NSamples;
        T_Devices Device;
} peakf_cmn;

#endif
```

# c.29 Tzero.c

```
/* tzero */
/* This function get tpi measure when the attenuators are all on */
/*ip : class */
/*ifchain : if chain of the used device */
/*device : used device */
/*samples : numebr of samples */
/*ierr : error code*/
/*sample : average, sigma obtained by sampling samples times the device*/
/*rut : time (in seconds) at the begging of sampling  */
/* Thanks Ed for this function */
#include <stdio.h>

#include "../../fs/include/params.h"
#include "../../fs/include/fs_types.h"
#include "../../fs/include/fscom.h"

#include "peakf.h"
#include "private.h"

extern struct fscom *shm_addr;

int tzero(ifchain,device,samples,sample,rut,ierr)
T_Devices device;
int ifchain,samples;
T_Sample *sample;
float rut;
int *ierr;
{
        int iclass, nrec, i,ifc[3],ierr2;
        short int buf2[80];
        long ip[5];
        int att1,att2;
        ifc[0]=ifc[1]=ifc[2]=0;
        ierr2=0;

        ifc[ifchain-1]=1;

        if(shm_addr->equip.rack==MK3||shm_addr->equip.rack==MK4) {
                iclass=0;
                nrec=0;
                buf2[0]=0;
                if(ifc[0]||ifc[1]) {
                        memcpy(buf2+1,"if",2);
                        memcpy(buf2+2,"00003F3F",8);
                        // questo serve per settare la modalita', se inp2if e' zero significa ch siamo in NOR
altrimenti se e'
                        // 1 siamo in ALT
                        if(shm_addr->inp2if!=0) memcpy(((char *)(buf2+2))+2,"8",1);
                        if(shm_addr->inp1if!=0) memcpy(((char *)(buf2+2))+3,"8",1);
                        if(!ifc[1]) att2=(shm_addr->iat2if)&0x3F;
                        else att2=0x3F;
                        if(!ifc[0]) att1=(shm_addr->iat1if)&0x3F;
                        else att1=0x3F;
                        sprintf(((char *)(buf2+2))+4,"%2.2X",att2);
                        sprintf(((char *)(buf2+2))+6,"%2.2X",att1);
                        cls_snd(&iclass,buf2,12,0,0); nrec++;
                }
                if (ifc[2]) {
                        memcpy(buf2+1,"i3",2);
                        memcpy(buf2+2,"00000F3F",8);
                        if(shm_addr->ipcalif3==0) memcpy(((char *)(buf2+2))+4,"2",1);
```

```
                        sprintf(((char *)(buf2+2))+5,"%1.1X",0xF&((2-shm_addr->iswif3_fs[0])+
                          (2-shm_addr->iswif3_fs[1])*2+(2-shm_addr->iswif3_fs[2])*4+(2-shm_addr-
>iswif3_fs[3])*8));
                        sprintf(((char *)(buf2+2))+6,"%2.2X",0x7F&(((2-shm_addr->imixif3)<<6)+0x3F));
                        cls_snd(&iclass,buf2,12,0,0); nrec++;
                }
                if(mat(ip,iclass,nrec,&ierr2)) {
                        goto restore;
                }
                if(ip[1]!=0) {
                        cls_clr(ip[0]);
                }
        }
        get_samples(device,samples,&ierr2,sample,rut);

restore:
        if(shm_addr->equip.rack==MK3||shm_addr->equip.rack==MK4) {
                iclass=0;
                nrec=0;
                buf2[0]=0;
                if(ifc[0]||ifc[1]) {
                        memcpy(buf2+1,"if",2);
                        memcpy(buf2+2,"00003F3F",8);
                        if(shm_addr->inp2if!=0) memcpy(((char *)(buf2+2))+2,"8",1);
                        if(shm_addr->inp1if!=0) memcpy(((char *)(buf2+2))+3,"8",1);
                        sprintf(((char *)(buf2+2))+4,"%2.2X",(shm_addr->iat2if)&0x3F);
                        sprintf(((char *)(buf2+2))+6,"%2.2X",(shm_addr->iat1if)&0x3F);
                        cls_snd(&iclass,buf2,12,0,0); nrec++;
                }
                if(ifc[2]) {
                        memcpy(buf2+1,"i3",2);
                        memcpy(buf2+2,"00000F3F",8);
                        if(shm_addr->ipcalif3==0) memcpy(((char *)(buf2+2))+4,"2",1);
                        sprintf(((char *)(buf2+2))+5,"%1.1X",0xF&((2-shm_addr->iswif3_fs[0])+
                          (2-shm_addr->iswif3_fs[1])*2+(2-shm_addr->iswif3_fs[2])*4+(2-shm_addr-
>iswif3_fs[3])*8));
                        sprintf(((char *)(buf2+2))+6,"%2.2X",0x7F&(((2-shm_addr->imixif3)<<6)+shm_addr->iat3if));
                        cls_snd(&iclass,buf2,12,0,0); nrec++;
                }
                if(mat(ip,iclass,nrec,ierr)) goto end;
                if(ip[1]!=0) {
                        cls_clr(ip[0]);
                }
        }
end:
        if (ierr2!=0) {
                *ierr=ierr2;
                return -1;
        }
        else if (*ierr!=0) {
                return -1;
        }
        return 0;
}
```

# c.30 Peakflib

```
/* Peakf Utility functions*/

// {}
#include <math.h>

#include "../peakf.h"
#include "../private.h"

// this function load an axe position with specified values
void loadAxe(dest,values,axe,mode)
T_AxisPosition *dest;
float *values;
T_Axis axe;
T_Mode mode;
{
        if (axe==AXIS_ZS) {
                dest->ZAxis[0]=values[AXIS_Z1S-mode];
                dest->ZAxis[1]=values[AXIS_Z2S-mode];
                dest->ZAxis[2]=values[AXIS_Z3S-mode];
        }
        else {
                dest->AllAxis=values[axe-mode];
        }
}

// this function fill a vector of float with the axe posistions
void uploadAxe(dest,values,axe,mode)
float *dest;
T_AxisPosition *values;
```

```
T_Axis axe;
T_Mode mode;
{
        if (axe==AXIS_ZS) {
                dest[AXIS_Z1S-mode]=values->ZAxis[0];
                dest[AXIS_Z2S-mode]=values->ZAxis[1];
                dest[AXIS_Z3S-mode]=values->ZAxis[2];
        }
        else {
                dest[axe-mode]=values->AllAxis;
        }
}

// This function copy an axe position from Orig to Dest
void copyAxe(dest,orig,axe)
T_AxisPosition *orig,*dest;
T_Axis axe;
{
        int i;
        if (axe==AXIS_ZS) {
                for (i=0;i<3;i++) {
                        dest->ZAxis[i]=orig->ZAxis[i];
                }
        }
        else {
                dest->AllAxis=orig->AllAxis;
        }
}

// This function sums two axis
void sumAxis(res,add1,add2,axe)
T_AxisPosition *add1,*add2,*res;
T_Axis axe;
{
        int i;
        if (axe==AXIS_ZS) {
                for (i=0;i<3;i++) {
                        res->ZAxis[i]=add1->ZAxis[i]+add2->ZAxis[i];
                }
        }
        else {
                res->AllAxis=add1->AllAxis+add2->AllAxis;
        }
}

void diffAxis(res,add1,add2,axe)
T_AxisPosition *add1,*add2,*res;
T_Axis axe;
{
        int i;
        if (axe==AXIS_ZS) {
                for (i=0;i<3;i++) {
                        res->ZAxis[i]=add1->ZAxis[i]-add2->ZAxis[i];
                }
        }
        else {
                        res->AllAxis=add1->AllAxis-add2->AllAxis;
        }
}

// this function load an axe with specifeid value
void setAxe(dest,value,axe)
T_AxisPosition *dest;
float value;
T_Axis axe;
{
        int i;
        if (axe==AXIS_ZS) {
                for (i=0;i<3;i++) {
                        dest->ZAxis[i]=value;
                }
        }
        else {
                dest->AllAxis=value;
        }
}

// this function add an offset to an axe position
void addAxe(dest,offs,axe)
T_AxisPosition *dest;
T_Axis axe;
float offs;
{
        int i;
        if (axe==AXIS_ZS) {
                for (i=0;i<3;i++) {
                        dest->ZAxis[i]+=offs;
                }
        }
        else {
```

```
                dest->AllAxis+=offs;
        }
}

/* remove a linear slope from data. */
void unslope(p1,p2,data,npts,slope,q)
T_Sample *p1,*p2,*data;
int npts;
float *slope,*q;
{
        int i;
        *slope=(p2->average-p1->average)/(p2->position-p1->position);
        *q=p1->average-(*slope)*(p1->position);
        for (i=0;i<npts;i++) {
                data[i].average-=data[i].position*(*slope)+(*q);
        }
}
```

## c.31 Fit library

Segueno ora i sorgenti della libreria per l'interpolazione non lineare[1].

## c.31.1 Covsrt.c

```
#define SWAP(a,b) {swap=(a);(a)=(b);(b)=swap;}

void covsrt(float **covar, int ma, int ia[], int mfit)
{
        int i,j,k;
        float swap;

        for (i=mfit+1;i<=ma;i++)
                for (j=1;j<=i;j++) covar[i][j]=covar[j][i]=0.0;
        k=mfit;
        for (j=ma;j>=1;j--) {
                if (ia[j]) {
                        for (i=1;i<=ma;i++) SWAP(covar[i][k],covar[i][j])
                        for (i=1;i<=ma;i++) SWAP(covar[k][i],covar[j][i])
                        k--;
                }
        }
}

#undef SWAP
```

## c.31.2 Fitlib.h

```
extern int NonLinearFit(float *x,float *y,float *sig,int datasize,float *params,float *eparams,int paramsize,int
maxiter,float *chi,void (*funcs)(float,float [],float *,float [],int));
```

## c.31.3 Gaussj.c

```
#include <math.h>
#define NRANSI
#include "nrutil.h"
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}

int gaussj(float **a, int n, float **b, int m)
{
        int *indxc,*indxr,*ipiv;
        int i,icol,irow,j,k,l,ll;
        float big,dum,pivinv,temp;

        indxc=ivector(1,n);
        indxr=ivector(1,n);
        ipiv=ivector(1,n);
        for (j=1;j<=n;j++) ipiv[j]=0;
        for (i=1;i<=n;i++) {
                big=0.0;
                for (j=1;j<=n;j++)
```

```
                        if (ipiv[j] != 1)
                                for (k=1;k<=n;k++) {
                                        if (ipiv[k] == 0) {
                                                if (fabs(a[j][k]) >= big) {
                                                        big=fabs(a[j][k]);
                                                        irow=j;
                                                        icol=k;
                                                }
                                        } else if (ipiv[k] > 1) { return -1; }
                                }
                ++(ipiv[icol]);
                if (irow != icol) {
                        for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
                        for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
                }
                indxr[i]=irow;
                indxc[i]=icol;
                if (a[icol][icol] == 0.0) { return -1; }
                pivinv=1.0/a[icol][icol];
                a[icol][icol]=1.0;
                for (l=1;l<=n;l++) a[icol][l] *= pivinv;
                for (l=1;l<=m;l++) b[icol][l] *= pivinv;
                for (ll=1;ll<=n;ll++)
                        if (ll != icol) {
                                dum=a[ll][icol];
                                a[ll][icol]=0.0;
                                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
                                for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
                        }
        }
        for (l=n;l>=1;l--) {
                if (indxr[l] != indxc[l])
                        for (k=1;k<=n;k++)
                                SWAP(a[k][indxr[l]],a[k][indxc[l]]);
        }
        free_ivector(ipiv,1,n);
        free_ivector(indxr,1,n);
        free_ivector(indxc,1,n);
        return 0;
}
#undef SWAP
#undef NRANSI
```

## c.31.4 Mrqcof.c

```
#define NRANSI
#include "nrutil.h"

void mrqcof(float x[], float y[], float sig[], int ndata, float a[], int ia[],
        int ma, float **alpha, float beta[], float *chisq,
        void (*funcs)(float, float [], float *, float [], int))
{
        int i,j,k,l,m,mfit=0;
        float ymod,wt,sig2i,dy,*dyda;

        dyda=vector(1,ma);
        for (j=1;j<=ma;j++)
                if (ia[j]) mfit++;
        for (j=1;j<=mfit;j++) {
                for (k=1;k<=j;k++) alpha[j][k]=0.0;
                beta[j]=0.0;
        }
        *chisq=0.0;
        for (i=1;i<=ndata;i++) {
                (*funcs)(x[i],a,&ymod,dyda,ma);
                sig2i=1.0/(sig[i]*sig[i]);
                dy=y[i]-ymod;
                for (j=0,l=1;l<=ma;l++) {
                        if (ia[l]) {
                                wt=dyda[l]*sig2i;
                                for (j++,k=0,m=1;m<=l;m++)
                                        if (ia[m]) alpha[j][++k] += wt*dyda[m];
                                beta[j] += dy*wt;
                        }
                }
                *chisq += dy*dy*sig2i;
        }
        for (j=2;j<=mfit;j++)
                for (k=1;k<j;k++) alpha[k][j]=alpha[j][k];
        free_vector(dyda,1,ma);
}

#undef NRANSI
```

## c.31.5 Mrqmin.c

```
#define NRANSI
#include "nrutil.h"

int mrqmin(float x[], float y[], float sig[], int ndata, float a[], int ia[],
        int ma, float **covar, float **alpha, float *chisq,
        void (*funcs)(float, float [], float *, float [], int), float *alamda)
{
        void covsrt(float **covar, int ma, int ia[], int mfit);
        int gaussj(float **a, int n, float **b, int m);
        void mrqcof(float x[], float y[], float sig[], int ndata, float a[],
                int ia[], int ma, float **alpha, float beta[], float *chisq,
                void (*funcs)(float, float [], float *, float [], int));
        int j,k,l;
        static int mfit;
        static float ochisq,*atry,*beta,*da,**oneda;

        if (*alamda < 0.0) {
                atry=vector(1,ma);
                beta=vector(1,ma);
                da=vector(1,ma);
                for (mfit=0,j=1;j<=ma;j++)
                        if (ia[j]) mfit++;
                oneda=matrix(1,mfit,1,1);
                *alamda=0.001;
                mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,funcs);
                ochisq=(*chisq);
                for (j=1;j<=ma;j++) atry[j]=a[j];
        }
        for (j=1;j<=mfit;j++) {
                for (k=1;k<=mfit;k++) covar[j][k]=alpha[j][k];
                covar[j][j]=alpha[j][j]*(1.0+(*alamda));
                oneda[j][1]=beta[j];
        }
        if (gaussj(covar,mfit,oneda,1)==-1) return -1;
        for (j=1;j<=mfit;j++) da[j]=oneda[j][1];
        if (*alamda == 0.0) {
                covsrt(covar,ma,ia,mfit);
                free_matrix(oneda,1,mfit,1,1);
                free_vector(da,1,ma);
                free_vector(beta,1,ma);
                free_vector(atry,1,ma);
                return 0;
        }
        for (j=0,l=1;l<=ma;l++)
                if (ia[l]) atry[l]=a[l]+da[++j];
        mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,funcs);
        if (*chisq < ochisq) {
                *alamda *= 0.1;
                ochisq=(*chisq);
                for (j=1;j<=mfit;j++) {
                        for (k=1;k<=mfit;k++) alpha[j][k]=covar[j][k];
                        beta[j]=da[j];
                }
                for (l=1;l<=ma;l++) a[l]=atry[l];
        } else {
                *alamda *= 10.0;
                *chisq=ochisq;
        }
        return 0;
}
#undef NRANSI
```

## c.31.6 Nonlinearfit.c

```
#include <stdio.h>
#include <math.h>
#define NRANSI
#include "nrutil.h"

int NonLinearFit(float *x,float *y,float *sig,int datasize,float *params,float *eparams,int paramsize,int maxiter,
    float *chi,void (*funcs)(float,float [],float *,float [],int))
{
        float *X;
        float *Y;
        float *SIG;
        float *A;
        int *IA;
        float **COVAR;
        float **ALPHA;
        int i;
        float alamda,chisq,ochisq;
        int itst,ok,suc;
```

```
        X=vector(1,datasize);
        Y=vector(1,datasize);
        SIG=vector(1,datasize);
        A=vector(1,paramsize);
        IA=ivector(1,datasize);
        COVAR=matrix(1,paramsize,1,paramsize);
        ALPHA=matrix(1,paramsize,1,paramsize);
        for (i=1;i<datasize+1;i++) {
                Y[i]=y[i-1];
                X[i]=x[i-1];
                SIG[i]=sig[i-1];
        }
        for (i=1;i<paramsize+1;i++) {
                A[i]=params[i-1];
                IA[i]=1;
                eparams[i-1]=0.0;
        }
        alamda=-1.0;
        ochisq=0.0;
        itst=ok=suc=0;
        while ((itst<=maxiter) && (!ok)) {
                if (mrqmin(X,Y,SIG,datasize,A,IA,paramsize,COVAR,ALPHA,&chisq,funcs,&alamda)) {
                        // matrix is singular !!
                        free_vector(X,1,datasize);
                        free_vector(Y,1,datasize);
                        free_vector(SIG,1,datasize);
                        free_vector(A,1,paramsize);
                        free_ivector(IA,1,paramsize);
                        free_matrix(ALPHA,1,paramsize,1,paramsize);
                        free_matrix(COVAR,1,paramsize,1,paramsize);
                        return -1;
                }
                if (fabs(ochisq-chisq)<0.01) suc++;
                else suc=0;
                if (suc>=4) ok=1;
                ochisq=chisq;
                itst++;
        }
        *chi=chisq;
        for (i=1;i<paramsize+1;i++) {
                params[i-1]=A[i];
        }
        if (ok) {
                alamda=0.0;
                mrqmin(X,Y,SIG,datasize,A,IA,paramsize,COVAR,ALPHA,&chisq,funcs,&alamda);
                for (i=1;i<paramsize+1;i++) {
                        eparams[i-1]=sqrt(COVAR[i][i]);
                        params[i-1]=A[i];
                }
        }
        else {
                // method doesn't converge.
                free_vector(X,1,datasize);
                free_vector(Y,1,datasize);
                free_vector(SIG,1,datasize);
                free_vector(A,1,paramsize);
                free_ivector(IA,1,paramsize);
                free_matrix(ALPHA,1,paramsize,1,paramsize);
                free_matrix(COVAR,1,paramsize,1,paramsize);
                return -2;
        }
        free_vector(X,1,datasize);
        free_vector(Y,1,datasize);
        free_vector(SIG,1,datasize);
        free_vector(A,1,paramsize);
        free_ivector(IA,1,paramsize);
        free_matrix(ALPHA,1,paramsize,1,paramsize);
        free_matrix(COVAR,1,paramsize,1,paramsize);
        return itst;
}

#undef NRANSI
```

## c.31.7 Nrutil.h

```
#ifndef _NR_UTILS_H_
#define _NR_UTILS_H_

static float sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)

static double dsqrarg;
#define DSQR(a) ((dsqrarg=(a)) == 0.0 ? 0.0 : dsqrarg*dsqrarg)

static double dmaxarg1,dmaxarg2;
```

```
#define DMAX(a,b) (dmaxarg1=(a),dmaxarg2=(b),(dmaxarg1) > (dmaxarg2) ?\
        (dmaxarg1) : (dmaxarg2))

static double dminarg1,dminarg2;
#define DMIN(a,b) (dminarg1=(a),dminarg2=(b),(dminarg1) < (dminarg2) ?\
        (dminarg1) : (dminarg2))

static float maxarg1,maxarg2;
#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
        (maxarg1) : (maxarg2))

static float minarg1,minarg2;
#define FMIN(a,b) (minarg1=(a),minarg2=(b),(minarg1) < (minarg2) ?\
        (minarg1) : (minarg2))

static long lmaxarg1,lmaxarg2;
#define LMAX(a,b) (lmaxarg1=(a),lmaxarg2=(b),(lmaxarg1) > (lmaxarg2) ?\
        (lmaxarg1) : (lmaxarg2))

static long lminarg1,lminarg2;
#define LMIN(a,b) (lminarg1=(a),lminarg2=(b),(lminarg1) < (lminarg2) ?\
        (lminarg1) : (lminarg2))

static int imaxarg1,imaxarg2;
#define IMAX(a,b) (imaxarg1=(a),imaxarg2=(b),(imaxarg1) > (imaxarg2) ?\
        (imaxarg1) : (imaxarg2))

static int iminarg1,iminarg2;
#define IMIN(a,b) (iminarg1=(a),iminarg2=(b),(iminarg1) < (iminarg2) ?\
        (iminarg1) : (iminarg2))

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

void nrerror(char error_text[]);
float *vector(long nl, long nh);
int *ivector(long nl, long nh);
unsigned char *cvector(long nl, long nh);
unsigned long *lvector(long nl, long nh);
double *dvector(long nl, long nh);
float **matrix(long nrl, long nrh, long ncl, long nch);
double **dmatrix(long nrl, long nrh, long ncl, long nch);
int **imatrix(long nrl, long nrh, long ncl, long nch);
float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,
        long newrl, long newcl);
float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch);
float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh);
void free_vector(float *v, long nl, long nh);
void free_ivector(int *v, long nl, long nh);
void free_cvector(unsigned char *v, long nl, long nh);
void free_lvector(unsigned long *v, long nl, long nh);
void free_dvector(double *v, long nl, long nh);
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch);
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch);
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch);
void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch);
void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch);
void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
        long ndl, long ndh);

#else /* ANSI */
/* traditional - K&R */

void nrerror();
float *vector();
float **matrix();
float **submatrix();
float **convert_matrix();
float ***f3tensor();
double *dvector();
double **dmatrix();
int *ivector();
int **imatrix();
unsigned char *cvector();
unsigned long *lvector();
void free_vector();
void free_dvector();
void free_ivector();
void free_cvector();
void free_lvector();
void free_matrix();
void free_submatrix();
void free_convert_matrix();
void free_dmatrix();
void free_imatrix();
void free_f3tensor();

#endif /* ANSI */
```

```
#endif /* _NR_UTILS_H_ */
```

## c.32 Makefiles

```
#
LIBPATH=../../fs/
CFLAGS= -g
LIBES =  $(LIBPATH)poclb/poclb.a $(LIBPATH)clib/clib.a $(LIBPATH)rtelb/rtelb.a\
        ../medlib/medlib.a ../stlib/stlib.a ./lib/peakflib.lib ./fitlib/fitlib.lib -lf2c -lm
#
OBJECTS = peakf.o mat.o antcn.o onsor.o tpi_request.o tpi_get.o get_samples.o fparabola.o fgauss.o source_record.o
scu_record.o\
          peakf_record.o go_scu.o do_scan.o log.o display_sample.o go_off.o local.o display_fit.o display_offset.o
tzero.o\
          get_device.o get_gain_par.o scmds.o
#
peakf : $(OBJECTS)
         cc  -o ../bin/peakf $(OBJECTS) $(LIBES)
#
clean:
         touch ../bin/peakf
         rm -f *.o
cleanbck:
         rm -f *.c%
         rm -f *.h%



# Makefile for library routines
#
OBJECTS = peakflib.o
#
peakflib.a: $(OBJECTS)
         ar -qc peakflib.lib $(OBJECTS)
         ar -s peakflib.lib
#
clean:
         rm peakflib.lib
         touch *.c
         rm *.o



# Makefile for library routines
#
OBJECTS = covsrt.o gaussj.o mrqcof.o mrqmin.o nrutil.o nonlinearfit.o
#
fitlib.lib: $(OBJECTS)
         ar -qc fitlib.lib $(OBJECTS)
         ar -s fitlib.lib
#
clean:
         rm fitlib.lib
         touch *.c
         rm *.o
```

# Riferimenti


[1] **Numerical Recipes in C.** W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery. Cambridge University Press.

[2] **Prestazioni della nuova meccanica subriflettore: sensibilità, ripetibilità e stabilità di puntamento**. A. Orfei, M. Morsiani, G. Zacchiroli, G. Maccaferri. IRA-CNR 236/97

[3] **The Field System Manual Vol 1 & 2.**

[4] **Calibrazioni di guadagno dei ricevitori VLBI di Medicina.** A. Orlati, G. Maccaferri IRA-CNR 341/03

[5] **Data reduction and error analysis for the physical sciences**. P.R. Bevington, D.K. Robinson. McGraw-Hill, Inc.

[6] **http://192.167.189.40/polinomial.html**