LEAR Crystal Barrel Experiment, PS197
Z–Calibration of the JDC
JDC Calibration Version 1.50/00

Curtis A. Meyer
Carnegie Mellon University

22 July, 1995

Contents

1 Introduction

This note is a technical description on how the z–calibration of the JDC is performed. It is assumed throughout this manual that the user is familiar with the Crystal Barrel Reconstruction Software [1] and in particular the Chamber Reconstruction Software [2], [3]. It is also assumed that the user is familiar with the MINUIT program [4] as well as HBOOK [5] and HPLOT [6] Finally, it is assumed that the user is familar with CMZ. This manual is not intended for the general users; rather it is meant as a detailed guide for people wishing to perform a z–calibration of the JDC.

The z–coordinate of a wire in the JDC is obtained from the LEFT and RIGHT amplitudes through the formula,

$$z^i = z_0^i + z_l^i \cdot \frac{A_L - \alpha^i A_R}{A_L + \alpha^i A_R},$$

(see Calibration Data Banks in reference [2] for details). Each wire is then described by three parameters, z_0, z_l, and α. In the present version of the calibration, the first two can only assume discreet values, while the third is a continuous number.

- z_0 is the center in units of centimeters of the wire relative to the center of the JDC. It can assume three possible values, -0.92, 0.00 and $+0.92$. These correspond to the four possible crimp conditions on the wire.
- z_l is the electrical length of the wire in the JDC, (including the resistance of the preamplifiers). It could assume three possible values, but only two of these can be determined in this procedure, (the number of wires in the third case is quite small, and the effects of using the wrong length are probably negligible). The two values are 23.68cm and 22.76cm.
- α measures the relative gain between the two preamps. It is a continuous variable, but physically should be constrained between 0.91 and 1.10.

Before starting the calibration, it is necessary that the user have the correct version of LOCATER. The code should be generated using the *CALIB cradle as described in Building the LOCATER code in reference [2].

2 Z–Calibration Software

There are several different programs used to perform the z–calibration of the JDC. The first is a set of USER subroutines which is linked with the Crystal Barrel Offline software. This generates several output files which are then passed to later code. In addition to this USER code is a plotting package which examines the resolution seen in the chamber at every iteration. External to this USER code, is a set of programs to perform MINUIT fits to obtain the calibration, and to display the results of these fits. This section describes how each of these programs are built. The running of the software is then described in the next sections.

2.1 User Code for Z–Calibration

The USER code for z–calibration is distributed as a CMZ card file, USER_CALB.CAR. This should be made into a CMZ file, USER_CALB.CMZ by the user. (Note, the CMZ file contains all JDC calibration programs. In principle, all needed card files, kumac files, and command files are stored in the CMZ file, however much of that information will be repeated here. In order to perform the z–calibration, the user will need to create several programs from routines stored in the CMZ file. There is a program to determine the time offsets for the flash ADC crates known as CBRAWS, and a corresponding plotting program. There is also the z–calibration code known as CBZFIT, and its corresponding plot code.

In order to create an run these programs, the user will need a set of KUMAC files, and a set of command procedures. These can be extracted from the CMZ file for the user's machine by doing the following under CMZ.

```
CMZ> file user_calb
user_calb> cd shell_script
user_calb> select machine_flag (ALT,IBM,SUN,VAX)
user_calb> set user_calb.kumac -D
user_calb> ctot -Y kuma_cod
user_calb> set plot_calb.kumac -D
user_calb> ctot -Y kuma_plt
user_calb> set calb.exec -D
user_calb> ctot -Y exec_cod
user_calb> set plot.exec -D
user_calb> ctot -Y exec_plt
user_calb> exit
```

This will have created the following files:

- USER_CALB.KUMAC which is a generic KUMAC file for creating either the CBRAWS or CBZFIT program.
- PLOT_CALB.KUMAC which is a generic KUMAC file for creating either the PLOT_RAWS or PLOT_ZFIT programs.
- CALB.EXEC which is a command procedure for running either the CBRAWS or CBZFIT programs.
- PLOT.EXEC which is a command procedure for running either the PLOT_RAWS or PLOT_ZFIT programs.

In order to create the CBRAWS program, change the pilot to *RAWSCOD in the USER_CALB.KUMAC file. To create the CBZFIT code, then change the pilot to *ZFITCOD. To create the PLOT_RAWS code, change the pilot to *RAWPLT0 in the PLOT_CALB.KUMAC file, and to create the PLOT_ZFIT program, change the pilot to *ZFITPLT in the PLOT_CALB.KUMAC file. The programs are then created by excuting the KUMAC files from within CMZ

```
CMZ> exec user_calb.kumac
CMZ> exit
```

2.2 The External Calibration Code

In addition to the programs described in the previous section, there is a second file which is used for z–calibration. This is available as a CMZ card file, FITZCAL.CAR, which should be converted to a CMZ file, FITZCAL.CMZ. This file is used to generate several programs, all of which may be needed for the z–calibration. As before, these are selected using PILOTS in the files. The possibilities are given as follows:

- *FITZCAL. This generates a program which will read in the calibration file generated by the CBZFIT program, as well as the gain file used in running that program, and perform as MINUIT fit on every wire in the JDC. This program is referred to as FITZCAL.
- *INTERA. This is an interactive version of the previous program which allows the user to selectively fit individual wires, and to vary parameters of the fit. This program is referred to as FITZCAL_INT.

- *PLTZCAL This is a plotting program for comparing two different gain file. Normally a reference file, and the one generated by either of the two previous programs. This program is referred to as PLOTZCAL.
- *MKTABL This is a stand–alone program which will generate a initial gain file for calibration purpose. This program is referred to as MKTABL, and needs no external libraries in linking.
- *PRNTABL This is a stand–alone program for printing out calibration tables in a readable format. It needs no external libraries in linking.

In CMZ, do the following:

```
select  machine_flag
pilot *...
ctof -P
```

The resulting code is linked with the KERNLIB, PACKLIB, and GRAFLIB parts of the CERN library.

3 Running of the Calibration Code

The process of calibration is an iterative procedure which consists of running through the previous programs several times. In order to perform a calibration, a sample of field–off , ($B = 0$) data consisting of 70000 to 100000 events is needed. The first pass through the data is to identify which wires need to be turned off, (see the **TJWR** data bank and the /TJWIRE/ common block in reference [2]). These are then removed from the calibration by deselecting them in the first program, and the calibration begins in earnest.

This section will describe how to run each of the programs needed for calibration. It will not describe how the calibration is performed; that will be described in the next section. What is detailed is the input for each calibration program, the important output from each program, and a shell script for running each program. All shell scripts are written in C–shell; the user will need to provide similar scripts for other machines.

3.1 CBRAWS

The CBRAWS program runs like any normal CBOFF program. Use the following card file to direct the job.

```
WRIT 4                          !  Write to the log file.
LIST                            !  Make a copy on the log file.
FZIN 'XT' 0                     !  Initialize FZIN.
XTAL 'NONE'                     !  Turn off Crystal Reconstruction.
GLOB 'NONE'                     !  Turn off global tracking.
CHAM 'TRAK' 'RTRK' 'RAWS' 'PATT' !  Only do through pattern recognition.
USER 6=10000                    !  Set the total number of events.
STOP
STOP  <===== READING STOPS HERE ========
    USER(1)   Number of events per run to analyze.
    USER(2)   First event to analyze.
    USER(3)   Event to turn on Debug.
    USER(4)   First run to analyze.
    USER(5)   Last run to analyze.
    USER(6)   Total number of events t analyze.
```

Then use the following JDC card file to steer locater.

```
WRIT 4
LIST
DEBG   F F F F F F F F F
NDBG   0

PREC  'OLPT'             ! Select the old pattern recognition.
OJDC  F T                ! Select the new jdc

DELZ  23*0.90 24=1.
SY02     2.00E-4
SYDF     2.00E-4
ANGL    84.00    1.0
IAMP   400       1
BMAG   -15.0     1.0
ITFC   16*0      17=1

STOP  <======= READING STOPS HERE ====

 DEBG(1) : Overall debug control
 DEBG(2) : Debug unpacking raw data.
 DEBG(3) : Debug pattern recognition.
 DEBG(4) : Debug circle fits.
 DEBG(5) : Debug helix fits.
 DEBG(6) : Debug vertex fits.
```

Both of these card files can be extracted from the CMZ file by doing the following:

```
CMZ> file user_calb.cmz
user_calb> cd raws_instruct
user_calb> set card_raws.crd -D
user_calb> ctot card_crd
user_calb> set card_raws.jdc -D
user_calb> ctot card_jdc
user_calb> exit
```

3.2 PLOT_RAWS

The PLOT_RAWS program will read in the HBOOK file created by CBRAWS, and produce both pictures, and a starting calibration table for the CBZFIT program. The program is run on the Alliant FX/8 in Zürich using the following shell script.

```
#
#      Assign logical unit 003 to the histogram file.
#      Assign logical unit 081 to the output gain file.
#
    setenv FOR003 raws.hist
    setenv FOR081 jdc_gain_ref.tbl
#
#      Run the plotting program.
```

```
#
    plot_raws
#
#      Deassign the unit
#
    unsetenv FOR003
    unsetenv FOR081
#
    exit(0)
```

3.3 CBZFIT

In order to run CBZFIT as a calibration program, (and not as a general reconstruction program), one needs to tell the CBOFF code what is going on. This is done by using the following card file for CBOFF, (see Data Cards in reference [1]). (Note: you need to strip off the comments before using.)

```
WRIT 4                          !  Write to the log file.
LIST                            !  Make a copy on the log file.
FZIN 'XT' 0                     !  Initialize FZIN.
XTAL 'NONE'                     !  Turn off Crystal Reconstruction.
CHAM 'TRAK' 'RTRK' 'RAWS' 'PATT' !  Only do through pattern recognition.
USER 4=1403  5=1434             !  Select run range.
STOP
STOP  <===== READING STOPS HERE ========
    USER(1)   Number of events to analyze.
    USER(2)   First event to analyze.
    USER(3)   Event to turn on Debug.
    USER(4)   First run to analyze.
    USER(5)   Last run to analyze.
```

Next, one has to direct LOCATER to perform a calibration. This is done using the following card file, which is assigned to logical unit 81. (Note: you need to strip off the comments before using.)

```
WRIT 4                    ! Direct a copy to the log file.
LIST                      ! Make a list on the log file.
DEBG   F F F F F F F F F F  !
NDBG   0                  ! No event to debug.

PREC  'OLPT'              ! Select the old pattern recognition
OJDC  F T                 ! Selcet the new JDC
BXPR  716.0 1.0           ! Give the program a default pressure.

LGT2  T                   ! Allow bigger errors in r-phi.
SY02     6.25E-4
SYDF     9.00E-4
ANGL   84.00  84.00       ! Rotate the JDC into position.
BMAG   0.0  -15.0         ! Turn off the magnetic field.
ZOFF  -0.50  1.00         ! Set nominal z-vertex.

JDCL   T                  ! Turn on Calibration
ICAL   4                  ! Calibration mode 4, CJGLOZ.
```

```
LGWR   1                    ! Prevent the list of dead wires from being
                            ! erased by TJTIMI.

ITDF   250                  ! Loosen pattern recognition cuts
CLMB   12.0
XSQR   0.060
TMIN   0.020
JGAP   5

DELZ  23*1.50  24=1.0       ! 1-sigma z-errors.
LYVX   8                    ! Outermost inner layer of a track.
MTRK   3                    ! Minimum number of tracks.
MHIT   9                    ! Minimum hits per track.
ZPOS  20.00                 ! limit z to +/- 20cm.
ZCUT   6.00                 ! Tracks must project to this window.
ZCTT   3.00                 ! Fit vertex must be in this window.
ZRIN   0.000                ! r_xy must be larger than this.
ZROT   1.500                ! r_xy must be smaller than this.
ZXYR   0.50                 ! Maximum distance of track from x-y vertex.
ZPRB   0.02                 ! Fit probability cut.
ZMOV   3.50                 ! Maximum allowed shift in z on a wire.
ZAMP  1000.0                ! Minimum amplitude of a hit to be written.

STOP
STOP  <======= READING STOPS HERE ====
 DEBG(1) : Overall debug control
 DEBG(2) : Debug unpacking raw data.
 DEBG(3) : Debug pattern recognition.
 DEBG(4) : Debug circle fits.
 DEBG(5) : Debug helix fits.
 DEBG(6) : Debug vertex fits.
 DEBG(7) : Debug calibration code.
```

Both of these card files can be obtained from the CMZ file by the following:

```
CMZ> file user_calb.cmz
user_calb> cd raws_instruct
user_calb> set card_zfit.crd -D
user_calb> ctot card_crd
user_calb> set card_zfit.jdc -D
user_calb> ctot card_jdc
user_calb> exit
```

Finally, the user will need a starting guess to the z–calibration table. This should have been generated by the PLOT_RAWS program, (JDC_GAIN_REF.TBL).

The following C–shell script is used to run the z–calibration code on the Alliant FX/8 in Zürich.

```
#       Filename:       cbcalb_run
#       Language:       Berkeley Unix, C-shell
#       Author:         Curtis A. Meyer
#       Creation date: 10 February, 1990
```

```
#       References:
#       Description:   This file will run the cbzfit code.
#
#       Input:   card_zfit.crd   Steer CBOFF
#                card_zfit.jdc   Steer LOCATER
#                ref_gain.tbl    JDC Gain Table.
#                FOR021          Raw Data Tape
#
#       Output:  zfit.log        Status of the run.
#                zfit.dbg        Possible debug output.
#                zfit.err        Errors encountered in running.
#                zfit.hist       Histogram output, (input to plot_zfit).
#                CALGLOBZ.DAT    Calibration data written by the program,
#                                (input to fitzcal and fitzcal_inter).
#
#     ##########################################################
#
#      Assign the output files.
#
   setenv FOR004 zfit.log
   setenv FOR007 zfit.dbg
   setenv FOR008 zfit.err
   setenv FOR010 zfit.hist
#
#      Assign the input card files, (cf FFREAD).
#
   setenv FOR005 card_zfit.crd
   setenv FOR081 card_zfit.jdc
#
#      Assign the jdc_gain file.
#
   setenv FOR082 jdc_gain_ref.tbl
#
#      Assign the raw data tape.
#
   setenv FOR021 /dev/exabyte1
#
#      Run the program
#
   cbzfit
#
#      Deassign all units:
#
   unsetenv FOR003
   unsetenv FOR004
   unsetenv FOR007
   unsetenv FOR008
   unsetenv FOR010
   unsetenv FOR011
```

```
   unsetenv FOR020
   unsetenv FOR080
   unsetenv FOR081
   unsetenv FOR082
#
   exit(0)
```

During the calibration, it will probably be necessary to deselect wires in the JDC. At present, this has to be done by modifying the USER_ZFIT code. The user will have to edit the USINIT subroutine, and modify the section of code which looks like the following. Note that there are a group of wires in this code which should always be turned off. Leave them alone!

```
*        Turn off bad wires in the chamber:
*
      DO 70 ISEC = 1,30
         DO 60 ILYR = 1,23
            LGWIRE(ILYR,ISEC) = .FALSE.
 60      CONTINUE
 70   CONTINUE
*
*        The following wires are known to be bad from Hardware!
*        They should always be turned off.
*
      LGWIRE(21, 6) = .TRUE.
      LGWIRE(23, 6) = .TRUE.
      LGWIRE(16, 9) = .TRUE.
      LGWIRE(23,15) = .TRUE.
      LGWIRE(11,16) = .TRUE.
      LGWIRE(21,16) = .TRUE.
      LGWIRE( 6,17) = .TRUE.
      LGWIRE(21,20) = .TRUE.
      LGWIRE( 4,24) = .TRUE.
      LGWIRE( 6,27) = .TRUE.
      LGWIRE(16,27) = .TRUE.
*
*        Put the bad Wires for the present run period here.
*


*
```

The first index of LGWIRE is the layer number, and the second index is the sector number. A wire is deselected by setting LGWIRE to .TRUE. for the wire.

There are three important output files from CBZFIT. The first is the file CALGLOBZ.DAT which is used as input to the FITZCAL and FITZCAL_INT programs. This is a packed file containing the wire number, amplitudes, and fit z–coordinate for all accepted hits. The next file is ZFIT.LOG, the log file from this program. In particular, near the end of this file are statistics on every wire in the jdc. These can be used to decide which wires are not working. Finally, there is a file ZFIT.HIST, which is the output from HBOOK. This file is read in and examined by the PLOT_ZFIT program.

In running CBZFIT on the Alliant FX/8 in Zürich, a data sample triggered on two prongs takes about 125 ms per event. (NOTE: this is by no means the optimum data sample for z–calibration, rather the more tracks the better. If one has available a data sample triggered on more tracks, or

even minimum bias, then the calibration will produce much better results if only events with more than two tracks are used. This can be defined using the **MTRK** card.)

3.4 PLOT_ZFIT

The PLOT_ZFIT program reads in the ZFIT.HIST file as generated by CBZFIT, and fits a gaussian to the resolution histograms for every wire in the JDC, (690). The results of these fits are then printed to a file which can be used to decide which wires should be deselected. The program also draws pictures of many of the histograms, as well as the resolution as seen in one sector, (the sector can be selected with the parameter IISEC in the start of the code). The following shell script is used to run the program on the Alliant FX/8.

```
#
#      Assign logical unit 003 to the zfit.hist file.
#
    setenv FOR003 /users/cmeyer/CB/cboff/zfit.hist
#
#      Run the plotting program.
#
    plot_zfit
#
#      Deassign the unit
#
    unsetenv FOR003
#
    exit(0)
```

The program also produces three additional output files, *metafile.out*, *hist.log* and *hist.dat*. The first is a metafile containing the pictures plotted by the program. The second is an output file for HBOOK, and mostly contains stuff from MINUIT. The final file, *hist.dat* contains a summary of what happened on every wire, and in particular, which wires were poorly it. By looking for irregularities in the data in this file, wires can be deselected. An example output from this file is:

```
Sector  1 Layer  1 has  1695 entries.
Sector  1 Layer  2 has  2013 entries.
Sector  1 Layer  3 has  1982 entries.
  .     .  .     .  .    .    .
Sector  2 Layer 23 has   419 entries.
Mean   0.02978 Sigma   1.60150 Chisq  0.86515E+00
       0.08365         0.08281
Sector  3 Layer  1 has  1381 entries.
Mean  -0.60100 Sigma   0.88106 Chisq  0.58754E+01
       0.02887         0.03152
  .     .  .     .  .    .    .
Sector 30 Layer 22 has     0 entries.
Channel not fit due to insufficient data.
Sector 30 Layer 23 has   655 entries.
```

The quantity plotted as Δz is defined as:

$$\Delta z = z_{fit} - z_{measured},$$

where z_{fit} is the fit value of z on the wire, and $z_{measured}$ is the value of z computed with the two amplitudes and the present gain table.

3.5 FITZCAL

The FITZCAL program reads in the CALGLOBZ.DAT file written by CBZFIT, and performs a MINUIT fit on every wire to obtain the calibration constant α. During the fit, the program will produce a file *badwires.dat*, which is a list of poorly fitted wires, and a new calibration table, *new_gain.tbl*.

At the start of the program, the user is asked to enter the allowed range in α for a wire not to be written to the *badwires.dat* file. The nominal values are printed by the program; if they are acceptable, then the user should enter a negative number for the new value.

```
 Auto-update boundaries on alpha
       0.9100 <= ALPHA <=    1.10
 Please enter new bounds, (hold=negative) : -1.,1.08
```

The above input will cause the lower bound to be retained, and the upper bound to be changed to 1.08. If a value of α is found which is outside of this range, then the program will try to improve it by shifting the center and wire–length to other allowed values. The fact that the program has tried to shift things will also be printed in the *badwires.out* file. The following shell script is used to run the FITZCAL program on the Alliant FX/8 in Zürich.

```
#     Filename:      fit_z.sh
#     Language:      Berkeley Unix, C-shell
#     Author:        Curtis A Meyer
#     Creation date: 4 July,1989
#     References:
#     Description:   This shell script will run the non-interactive
#                    z calibration program.
#
#     Input:    CALGLOBZ.DAT   Data file written by CBZFIT.
#               ref_gain.tbl   The gain file used by CBZFIT.
#
#     Output:   fit.log        The logfile from the program.
#               fit.out        The output file from MINUIT.
#               debug.out      Output from debug prints.
#               badwires.out   List of all poorly fit wires.
#               new_gain.tbl   The new gain table.
#
#     ############################################################
#
#     Where are we?
#
   set Home=‘pwd‘
#
#     Set up the input files:
#
   setenv FOR015  /users/cmeyer/CB/cboff/ref_gain.tbl
   setenv FOR034   cbdata/CALGLOBZ.DAT
#
#     Set up the output files:
```

```
#
   setenv FOR003  fit.log
   setenv FOR004  fit.out
   setenv FOR013  $Home/debug.out
   setenv FOR014  $Home/badwires.out
   setenv FOR016  $Home/new_gain.tbl
#
#     Run the program
#
   /users/cmeyer/CB/cbcalb/fitzcal
#
   exit(0)
```

The following is an example of a *badwires.out* file. The *s/l/n* code refers to the sector, layer and number of data points. The *f* and *f/n* are the χ^2 and χ^2/n of the fit from MINUIT and *alp*, *z0* and *zl* are the three wire parameters, (recall that only α is fit). Note that the program successfully improved one of the bad fits, (on sector 3, wire 3), but that the other two poor fits were already at the limits of the center and length, so no improvement was tried. If the user would want to change any of these wires, or others that one considers poorly fit, then it is necessary to run the interactive version of this program.

```
 s/l/n  3  3   420 f   231.606 f/n    0.5514
 alp  1.1138 z0    0.000 zl   23.680
 Fixing s/l  3  3 to   -0.9200    22.76
 Sector  14 Layer  14 Data     0 NOT FIT !
 s/l/n 14 16   298 f   286.288 f/n    0.9607
 alp  1.1315 z0   -0.920 zl   22.760
 s/l/n 29 13   461 f   383.022 f/n    0.8309
 alp  1.1184 z0   -0.920 zl   22.760
 Sector  29 Layer  22 Data     0 NOT FIT !
```

3.6 FITZCAL_INT

The FITZCAL_INT program allows the user to try to improve the fits on those wires listed in the *badwires.out* file by changing the length and center of individual wires. In principle, it should not be necessary to run this program. The FITZCAL program tries to do what this program does; however, if the user feels that some of the data is poorly fit, or is unsatisfied with the results of the previous program, then this program could be used. The program is run using the following shell script.

```
#     Filename:      fit_z_i.sh
#     Language:      Berkeley Unix, C-shell
#     Author:        Curtis A Meyer
#     Creation date: 4 July,1989
#     References:
#     Description:   This shell script will run the non-interactive
#                    z calibration program.
#
#     Input:    CALGLOBZ.DAT   Data file written by CBZFIT.
#               ref_gain.tbl   The gain file used by CBZFIT.
#
```

```
#     Output:   fit.log        The logfile from the program.
#               fit.out        The output file from MINUIT.
#               debug.out      Output from debug prints.
#               badwires.out   List of all poorly fit wires.
#               new_gain.tbl   The new gain table.
#
#     ###############################################################
#
#     Where are we?
#
   set Home=‘pwd‘
   set cboff='/users/cmeyer/CB/cboff'
#
#     Set up the input files:
#
   setenv FOR015  new_gain.tbl
   setenv FOR034  cbdata/CALGLOBZ.DAT
#
#     Set up the output files:
#
   setenv FOR003  fit.log
   setenv FOR004  fit.out
   setenv FOR013  $Home/debug.out
   setenv FOR014  $Home/badwires.out
   setenv FOR016  $Home/fit_gain.tbl
#
#     Run the program
#
   /users/cmeyer/CB/cbcalb/fitzcal_int
#
   exit(0)
```

The program will start by asking the user which data set to fit. The two choices are *0* and *1*. Any other entry will cause the program to write out the gain file and stop. *0* is for sectors 1 to 15, while *1* is for sectors 16 to 30. The program will then read in the CALGLOBZ.DAT file, and store the data on the selected sectors. This normally takes a few minutes simply because the CALGLOBZ.DAT file is so big. After the data have been loaded, the program will ask for sector and layer numbers. If either of these is out of range, (sector 1–30, layer 1–23), the program will jump back and inquire on the data sets. Here the user enters the sector and layer number of a bad wire. In the example *badwires.out* file in the previous section, take sector 3, layer 3 with $\alpha = 1.1138$. Here enter:

```
3,3 <CR>
```

The program will then ask to make sure the numbers are correct. If so then it will ask for the new values of z_0 and z_l. The nominal set of data is $z_0 = 0.00$ and $z_l = 21.60$. However, if α is larger than 1.10, then we enter:

```
-0.667,23.934 <CR>
```

and if α is smaller than 0.910, we use:

```
+0.667,22.934 <CR>
```

The program will then refit this wire, and give the new value of α. If this value is better than the previous one, (in the range 0.91 to 1.10), then one tells the program to save it, otherwise the program forgets what it has done, and asks for another wire. After all wires in the selected data set have been refit, one should enter a sector and layer combination which is out of range, for example:

```
-1,-1 <CR>
```

The program will then inquire if it should read in a new data set. If this is not desired, then enter any integer except 0 or 1, and the program will write of the new gain file *fit_gain.tbl* and stop. This *fit_gain.tbl* file can then be used as the input for the next iteration of CBZFIT.

3.7 PLOTZCAL

The PLOTZCAL program is used to measure the change between calibration tables from adjacent iterations. It reads in two gain files, and makes a graphical comparison of the two. The pictures drawn are written to the file *metafile.out*. The following shell script runs the program:

```
#
#      Assign the new table to unit 003
#
   setenv FOR003 fit_gain.tbl
#
#      Assign the old table to unit 004.
#
   setenv FOR004 old_gain.tbl
#
#      Run the program
#
   plotzcal
#
#
   unsetenv FOR003
   unsetenv FOR004
#
   exit(0)
```

3.8 MKTABL

The MKTABL program is a stand–alone program which will generate a starting table for the calibration. Be sure to select real data and not Monte Carlo data before compiling the code (Two similar parameter statements in the code. If one forgets, then the program will not compile.) In order to do a calibration, one should always start with the table generated by this program. Starting from an old table can give bizarre results.

3.9 PRNTAB

The PRNTAB program will print out a gain table is a human readable for. With this output, one can look for deviations in one sector, or over several adjacent layers in the table.

4 Z–Calibration of the JDC

This section describes the steps involved in performing a z–calibration of the JDC. The section is divided into units called iterations, where each iteration will involve running some or all of the previous code.

4.1 Iteration 0

As the name implies, iteration zero is what has to be done before calibration can really begin. In this iteration, one wants to identify the bad wires in the JDC, and turn them off so they will not destroy later iterations.

Step 1:
Run the CBRAWS and PLOT_RAWS programs to generate a starting calibration table for the JDC.

Step 2:
Edit the USER_ZFIT code and modify the USINIT subroutine so that all wires are allowed. If any wires are known to be bad, then turn them off. Then create a version of the CBZFIT code.

Step 3:
Run the CBZFIT code with the gain file created in step one assigned to logical unit 82. This step takes about 125 ms per event on the Alliant FX/8, so with roughly 100000 events, expect about 3 to 4 cpu–hours. During this step, the user should write a DST tape, and the DST tape can then be used for all later calibration steps. This reduces by a factor of roughly four the CPU time needed in later steps.

Step 4:
Run the histogram file from CBZFIT through the PLOT_ZFIT program. The desired output here is the hist.dat file. In this file one wants to identify all wires which have either a very large σ, (larger than 2.5 cm), or whose centers are shifted by more than three centimeters. These wires are then turned off in step 2. One can also look at individual wires by looking at the PLOT_ZFIT program. Examples are given on how to plot the Δz distributions of any wire. It is advisable to repeat steps 2 through 4 at this point to make sure that no wires were missed. Note that all of these iterations should use the nominal gain table as generated in step 1. In figure 1 is shown what a poorly fit wire looks like, while figure 2 shows a well fit wire.

4.2 Iteration 1

This step produces the largest improvement in the calibration table. It essentially adjusts all of the α's such that all distributions will be centered at zero.

Step 1:
Create a version of the CBZFIT code with all bad wires as found in Iteration 0 turned off. Then run this code using the nominal gain table as generated in the previous iteration. This step will take between 3 and 4 cpu hours.

Step 2:
Run the CALGLOBZ.DAT file through the FITZCAL program. One should then examine the *bad-wires.out* file to make sure that all wires were shifted correctly. On this iteration, this should not be a problem. Also, when running the FITZCAL program, the nominal limits on α are the correct limits.

Step 3:
Run the PLOT_ZFIT program to see what the vertex and probability distributions look like. An example of the generated pictures are shown in figures 3 to 7.

Step 4:
Compare the new calibration table as generated by FITZCAL with the starting table by running the

PLOTZCAL program. Also, from this program, one should get an idea of what the limits on α should be for the next iteration. Example pictures are shown in figure 8.

4.3 Iteration n

This iteration can actually be several iterations, but all are now identical, so they are described as only one iteration. These iterations all use the CBZFIT code from iteration one.

Step 1:

As always, the first step is to run CBZFIT. However before starting, one should tighten up some of the cuts. The following cards, as well as any others deemed necessary, can be changed. Then the CBZFIT code is run.

```
DELZ   23*1.10 24=1.0  ! 1-sigma errors in z.
ZOFF                   ! It may be necessary to move the nominal z0.
ZCTT   1.50            ! Z-vertex window, (set from the z-vertex).
ZMOV   2.75            ! Don't allow the data to be shifted by as much
ZAMP   750.0           ! Take data down to lower amplitudes.
```

Step 2:

Run the CALGLOBZ.DAT file through the FITZCAL program to generate a new table. If there are problem wires, the user may have to employ FITZCAL_INT as well.

Step 3:

Compare the input and output tables using the PLOTZCAL to see if we have converged. Has α stopped varying by much? If so, then we are done, otherwise we need to repeat this iteration.

Figure Captions

Figure 1: The Δz, *fit error* and *pull* distributions of a poorly fit wire in the JDC. This wire will need to be turned off before calibration can proceed. The plotted quantity is the fit value of z minus the value of z computed from the two amplitudes, and the present gain table. The numbers at the top of each figure come from a gaussian fit to the distribution. The first is the mean of the fit, and the second is the sigma.
Figure 2: The Δz, *fit error* and *pull* distributions of a well fit wire in the JDC.
Figure 3: a: The distribution of all σ's and **b:** the center of each gaussian fit as per figures 1 and 2. This picture was made after iteration 1. The units are in centimeters.
Figure 4: a: and **b:** are as in the previous figure. This plot is from after iteration 3.
Figure 5: a: The x and **b:** y vertex distribution of the fit.
Figure 6: a: The z and **b:** r_{xy} vertex distribution of the fit. The z distribution plot is used to assign new values to the **ZOFF** and **ZCTT** cards.
Figure 7: a: The χ^2 of every fit, and **b:** the corresponding probability distribution. The input errors of 1.5cm are clearly too large, and can be lowered in the next iteration.
Figure 8: This figure shows Δz, $(z_{fit} - z_{meas})$, the *fit error* in z and the *pull* of the quantity for one layer in the JDC. This is made after iteration 1. The two numbers printed at the top of each figure are the mean and sigma of the fit gaussian. Note that the centers are not at zero, this iteration essentially will shift them there, and later iterations will make only small improvements.
Figure 9: The same as figure 8, except after iteration 3.
Figure 10: This figure shows the distribution in α, z_l and z_0, and the change in these quantities after the first iteration.
Figure 11: This is the same as figure 10, except it is after the third iteration. Note that α has stopped varying.

References

[1] Gunter Folger, CB–Note 121, **Offline Reconstruction Software**.

[2] Curtis A. Meyer, CB–Note 93, **Chamber Reconstruction Software**.

[3] Curtis A. Meyer, CB-Note 123, **Users Guide for Locater**.

[4] F. James and M. Roos, **MINUIT, Function Minimization and Error Analysis, Release 89.03**, *CERN Computer Center Program Library Long Write–Up D506.*

[5] R. Brun and D. Lienart, **HBOOK User Guide, Version 4**, *CERN Computer Center Program Library Long Write–Up Y250*

[6] R. Brun and N. Cremel Somon, **HPLOT User Guide, Version 5**, *CERN Computer Center Program Library Long Write–Up Y251*

[7] H. J. Klein and J. Zoll, **PATCHY Reference Manual**, *CERN Program Library.*

[8] M. Brun, R. Brun and F. Rademakers, **A Source Code Management System, CMZ, User's Guide & Reference Manual**.

[9] F. H. Heinsius and T. Kiel, CB–Note 92, **Crystal Data Reconstruction Software**.