CB-Note 138

LEAR Crystal Barrel Experiment, PS197
Kinematic Fitting Software Ver. 3.11/00

Pal Hidas, Gyorgy Pinter
KFKI , Budapest

April 16^{th}, 1997

Contents

1 Introduction

CBKFIT is a kinematic fitting package with automatic hypothesis handling and generation of combinatoric cases of identical particles in the input which reads the Global Tracking Particle Banks. The user can call it (CALL CFDOFI) from the USER routine of CBOFF several times in an event. To select the measured particles, suppress some combinations, make mass preselection and take out the result. There are nontrivial but rewritable user routines. For old DST-s the user must call the new BCTTKS routine to each PED-s which imposes energy dependent errors on fit parameters.

The cmz file consists of four patches. CFCOMMON contains the sequence definitions, see it for the description of variables, CBKFIT contains the fit routines, CBHYPO does the hypothesis handling and CFUSER yields some examples how to write the user routines. See the cmz file for the description of routines as well.

This package can handle 25 particles in the hypothesis 20, 5 and 5 of which are allowed to be photons, positive and negative tracks respectively.

The output is written in the KRES top bank and its KSUB sub-banks. It is highly recommanded to use the reference links of both banks to access to the hypothesis words and particles respectively, because then you can avoid rewriting your code when the leading part of the banks are changed. These changes are sometimes necessary, e.g. when extra vertices are introduced.

2 Initialization

To initialize a run one has to

- call CFINIT(n) from USINIT
- include +SEQ,CFCOMS. in all the user routines, no other commons are needed
- add COMMON /CFCSTR/ ICMBPR(n) to USINIT
- put hypothesis input cards on LUN=LHYP (LHYP=32 recently)
- overwrite defaults and set logical variables according to the different cases (see sections 3, 4, 5 and 6) of the fit in USER or after the CFINIT call in USINIT
- for deuterium target set the logical variable CFDEUT to be true in USINIT (after CFINIT)
- event by event you should fill the integer array MEASCO with the TTKS ids of particles which you want to fit (see subsection 7.1)
- call the logical function CFMPRE in the CFUSCO routine as it is also given in the appendix (the example)

To get a compound statistics one has to

- call CFLAST from USLAST

Here n should be greater than the number of particles times the number of combinatoric cases of all the hypotheses of one event type. It allows to create a lookup table of combinatorics at the beginning of a run. For example if you analyze 5 gamma events and try 3 hypotheses as following

```
ETYP 0 0 0 0 5 0
HYPO G G G G G
HYPO PI0 PI0 G
...
HYPO ETA PI0 G
...
```

then you have 1 combinatorial case of the first hypothesis, 15 cases of the second one and 30 cases of the third one. You have 46 combinatorial cases and 5 particles so you must set n to be greater than 230.

2.1 Hypothesis input

There are four types of hypothesis input cards

ETYP event type card :

ETYP i1 i2 i3 i4 i5 i6

where i2, i4 and i6 are not yet used, but must be set to 0 and i1 is the number of positive tracks, i3 is the number of negative tracks and i5 is the number of photons in the final state to be investigated. One has to start writing one's input with one of these cards and can study several parallel channels repeating it later with different arguments. i1-i6 are of type CHARACTER*1 in the range 1-9,A,B,C where A=10,...,C=12.

HYPO hypothesis card :

HYPO par1 par2 ...

where par1, par2, ... are names of particles in the intermediate state (see routine CFLKUP for definitions). You can also use Subroutine CFGTUP(IGEANT,CNAME) to receive the character variable CNAME used as par1, par2, ... where the input parameter IGEANT is the integer Geant particle code. Particles can be stable particles or resonances which decays through several steps into the final state. If you have resonances on your hypothesis card (see array RBUF in routine CFLKUP) then resolve them by RES1 resonance cards in the order of appearance.

RES1 first level resonance card

```
RES1 par1 -> par2 par3 ...
```

which resolves resonance par1 appeared in the previous hypothesis card.

RESn nth level resonance card

```
RESn par1 -> par2 par3 ...
```

if you still have resonances on a certain level of resonance cards you have to resolve them in the order of appearance but not mixing different levels until you get to the final state.

END end card must be the last one of the hypothesis input.

You can write several event type cards on your input and you can also write several hypothesis cards after each of them but the first one must be the phase space (the final state without resonances). Do not separate identical particles by another one, if possible, otherwise you can get combinatorical multiplications. If you want to comment out a line, use 'C' as the first character of the line followed by a blank one.

Example :

```
ETYP 0 0 0 0 5 0
 HYPO G G G G G
 HYPO ETA PI0 G
  RES1 ETA -> G G
  RES1 PI0 -> G G
 HYPO ETA OM
  RES1 ETA -> G G
  RES1 OM  -> PI0 G
   RES2 PI0 -> G G
ETYP 1 0 1 0 4 0
 HYPO PI+ PI- G G G G
END
```

3 Missing particles, bad energy, merged particles

In the basic case (CASE=1) charged and neutral particles are fitted without vertex fit and missing particles or energies. Resonance (mass) constraints are allowed.

3.1 Missing photon

In this case (CASE=2) both the momentum and the energy is missing. To do a fit like this one has to

- set the logical variable MISGAM to be true
- include hypothesis cards of the new event type on the hypothesis input

3.2 Missing K long

In this case (CASE=2) both the momentum and the energy is missing. To do a fit like this one has to

- set the logical variable MISSKL to be true
- include hypothesis cards of the new event type on the hypothesis input

The K long (KL) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards. For the event type K long is regarded as photon.

3.3 Missing neutron

In this case (CASE=2) both the momentum and the energy is missing. To do a fit like this one has to

- set the logical variable MISNEU be true
- include hypothesis cards of the new event type on the hypothesis input

The neutron (N) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards. For the event type neutron is regarded as photon.

3.4 Missing proton

In this case (CASE=2) both the momentum and the energy is missing. To do a fit like this one has to

- set the logical variable MISPRO be true
- include hypothesis cards of the new event type on the hypothesis input

The proton (P) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards.

3.5 Missing neutral particle with mass reset (old)

This is a generalization to do fits with missing neutral particle with a mass set by the user. Please be careful with the mass set and always check the error message file. The mass should be different from any of the particle masses in CFLKUP. If not so just change the value of sixth digit. In this case (CASE=2) both the momentum and the energy is missing. To do a fit like this one has to

- set the real variable CFMISM to have the required mass what should be different from zero
- to name the particle as X on the hypothesis cards

- include hypothesis cards of the new event type on the hypothesis input

The particle (X) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards.

3.6 Missing neutral particle with mass reset (new)

This is a generalization to do fits with missing neutral particle with a mass set by the user. Please be careful with the mass set and always check the error message file. The mass should be different from any of the particle masses in CFLKUP. If not so just change the value of sixth digit. In this case (CASE=11) both the momentum and the energy is missing. To do a fit like this one has to

- set the real variable CFMISX to have the required mass what should be different from zero
- set the real array CFMIER(1/3/6) to be the "error squared of the missing particle parameters" the defaults are (1.e-3,1.e-3,3.) for (Φ,Θ,$\sqrt{(E)}$)
- to name the particle as X on the hypothesis cards
- include hypothesis cards of the new event type on the hypothesis input

The particle (X) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards.

3.7 Neutron with bad energy

In this case only the energy is missing (CASE=3) but you know the direction of the momentum. To do a fit like this one has to

- set the integer variable BADNEU to the TTKS id of the neutron (typically 1 or 2 crystal PED with low energy)
- include hypothesis cards of the new event type on the hypothesis input

The neutron (N) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards. For the event type neutron is regarded as photon. Also in MEASCO the TTKS id of the neutron should be the last.

3.8 Klong with bad energy

In this case only the energy is missing (CASE=3) but you know the direction of the momentum. To do a fit like this one has to

- set the integer variable BADKL to the TTKS id of the Klong
- include hypothesis cards of the new event type on the hypothesis input

The Klong (KL) must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards. For the event type Klong is regarded as photon. Also in MEASCO the TTKS id of the Klong should be the last.

3.9 Neutral particle with bad energy and mass reset

In this case only the energy is missing (CASE=12) but you know the direction of the momentum. To do a fit like this one has to

- set the integer variable BADXCF to the TTKS id of the badly measured particle
- set the real variable CFMISX to have the required mass what should be different from zero
- to name the particle as X on the hypothesis cards
- include hypothesis cards of the new event type on the hypothesis input

The particle X must be the last particle on the hypothesis card. Furthermore it must not be included in any resonance cards. For the event type X is regarded as photon. Also in MEASCO the TTKS id of the X should be the last.

3.10 Merged pions and etas

You can directly fit π°s and ηs found by PI0FND. To do this one has to

- set the integer variable NPI0CF to the number of merged π°s and ηs used
- fill the TTKS ids of merged π°s and ηs to the integer array MERPI0(NRMAX)
- include these ids also in MEASCO as all other measured particles.
- exclude from MEASCO the ids of photons coming from these pions

The merged π°s and ηs should be the last particles on the hypothesis input, also in MEASCO with the same ordering as in MERPI0. You can include these particles also in resonances. Combinatorics is not yielded for them by the fitter, the user is supposed to do it by repeating hypotheses on the hypothesis input or simply permutating MEASCO and MERPI0.

3.11 Using photon covariances

Originally covariances of photons were not used in the fitter, but you can use them now in CASE=1,2,3,6. To do this

- set the logical variable CFGCOV be true

4 Vertex fit

In version 3.00/00 new main and secondary vertex fits were introduced. Formerly in CASE=4 and 5 the vertex coordinates were regarded as unmeasured parameters (e.g. their error was infinite) and only the square root of energy of the particles was fitted. This resulted in an incorrect vertex distribution because the vertex was also corrected when the angular variables should have been done. In the z-vertex fit the problem even appeared in the confidence level, because the fitter could not improve the bad Φ measurement by the vertex move along the z-axis. So from the version mentioned above Φ also used in the CASE=4 z-vertex fit. This improves the confidence level a lot.

Introducing Θ in this method leads to singularity, because a vertex shift or a simultaneous Θ shift has the same result on the constraints. This problem can be avoided using the vertex coordinates as measured variables (e.g. introducing them into the covariance matrix). For the x and y coordinates we have a real measurement, because we set the beam (of course with an error) at x=y=0. For z the fitter does first a Newton- iteration, which "eats" one (the P_z) constraint, e.g. decreases the number of constraints by one. The CASE=6 and 7 vertex fits uses then all measured parameters and handles the vertex coordinates as measured. The result is a realistic vertex distribution and a somewhat more smooth confidence level distribution.

The users are recommended to avoid using CASE=5 (use CASE=7 instead) and to prefer CASE=6 instead of CASE=4, but in this case do a comparison test first.

4.1 Main vertex fit - vertex is unmeasured parameter

For CASE=4 only the z coordinate, for CASE=5 all x, y and z coordinates are fitted and resonances are not allowed, but subsequent high constraint fits are done with the vertex found by this fit. The vertex coordinates are parameters with infinit error and one or three momentum constraints are used to calculate them so these fits are 3C and 1C fits respectively. For CASE=5 only the square root of energy is used as fit parameter, so angular pulls are unavailable. For CASE=4 phi also used. Resonance (mass) constraints and charged particles are not allowed in neither CASE=4 nor CASE=5. To do a fit one has to

- set the logical variable CFVRTZ be true for CASE=4
- set the logical variable CFVERT be true for CASE=5

4.2 Main vertex fit - vertex is measured parameter

For CASE=6 only the z-coordinate, for CASE=7 the full vertex is fitted. The vertex parameters are regarded as measured parameters. The x and y coordinates are really measured variables, because we know where the beam is. In addition a newton iteration is made to find an initial value for the vertex z-coordinate. This uses the really measured momenta, so the z-vertex measurement is not independent from the measurement of the

particle parameters, but uses the z-momentum constraint to "measure". That is why both CASE=6 and 7 are 3C fits. All the measured particle parameters (phi, theta, square root of energy) are fitted. The angular parameters counted with respect to the origin of the coordinate system, but not to the fitted vertex. Resonance (mass) constraints are allowed, but charged particles are not. To do a fit one has to

- set the logical variable CFVERZ to be true for the z-vertex fit, or
- set the logical variable CFVERA to be true for the full vertex fit
- set the real array CFVTER(1/2/3) to be the error squared of the "vertex measurement", the defaults are 0.03 for all of them

4.3 Neutral K°_{short} vertex fit

This is CASE=8 which fits the $K^{\circ}_{short} \rightarrow \pi^0\pi^0$ hypothesis with the secondary vertex parameters and any other particles coming from the fixed main vertex. Before the fit a 3 parameter newton iteration is made to find an initial value for the vertex coordinates using the momentum conservation for the K°_{short} decay vertex. All the measured particle parameters (phi, theta, square root of energy) are fitted. The angular parameters counted with respect to the origin of the coordinate system, but not to the fitted vertex. Resonance (mass) constraints and charged particles are allowed, but missing particles are not. The K°_{short} which decays in the neutral mode should be the last particle on the hypothesis card. Keeping e.g. $\pi^0 K^{\circ}_{short} K^{\circ}_{short}$ in mind, where the first kaon decays into two charged pions correction of the momenta of these two particles is made with respect to the vertex found by LOCATER. To do a fit one has to

- set the logical variable CFKS00 to be true
- set the real array CFVTER(1/2/3) to be the error squared of the "vertex measurement", the defaults are 1.0 for all of them
- set the logical variable CFCHCR to be true if you want to switch the charged correction on
- write a hypothesis card with a last KSH decaying into the neutral PI0 PI0 mode (another "charged KSH" is allowed

4.4 Neutral K°_{short} vertex fit - 2 vertices

This is CASE=13 which based on CASE=8 The two K°_{short}s should be the last particles on the hypothesis card. To do a fit one has to

- set the logical variable CFKSKS to be true
- set the real array CFVTER(1/2/3) to be the error squared of the "vertex measurement", the defaults are 1.0 for all of them

- set the logical variable CFCHCR to be true if you want to switch the charged correction on (for the fixed main vertex)
- write a hypothesis card with two last KSHs decaying into the neutral PI0 PI0 mode

4.5 Charged K°_{short} vertex fit

Well this is not really true. Charged vertices always should be fitted by LOCATER. Parallel with this version a new TCVERT routine is written for LOCATER to fit not only the main, but also secondary vertices. Untill it is not an official part of LOCATER, you can find it in //CBKFIT/CFSTOR, from which you can copy it in your analysis file, but please never compile this patch when you create an object library (e.g. you should not include this patch name in your kumac file). For CBKFIT this is a normal CASE=1 fit, but it uses the information yielded by LOCATER if you use the CFCHCR option. You can fit here X K°_{short} K°_{short}, where X can be any group of particles and resonances. To do a fit one has to

- set the logical variable CFKSCC to be true
- set the logical variable CFCHCR to be true if you want to switch the charged correction on (this you can set parallel with any options and cases of CBKFIT, not only for the CFKSCC case)
- fill the integer IDRVCF array with the TCVX numbers of (maximum of 10) vertices, which you want to drop, e.g. do not want a charged correction for them (this is necessary when there are several vertices found which contain the same tracks)

4.6 Neutral K°_{short} vertex fit with missing K_{long}

This is CASE=9 which combines the CASE=8 secondary vertex fit with the CASE=11 missing particle fit. The K°_{short} which decays in the neutral mode should be the next to last particle on the hypothesis card, and the K_{long} should be the last one. To do a fit one has to

- set the logical variable CFKSKM to be true
- set the real array CFVTER(1/2/3) to be the error squared of the "vertex measurement", the defaults are 1.0 for all of them
- write an appropriate ... KSH KL hypothesis card
- set the real array CFMIER(1/3/6) to be the "error squared of the missing particle parameters" the defaults are (1.e-3,1.e-3,3.) for (Φ,Θ,$\sqrt{(}E$))

4.7 Neutral K°_{short} vertex fit with bad K_{long}

This is CASE=10 which combines the CASE=8 secondary vertex fit with the CASE=12 missing energy particle fit. The K°_{short} which decays in the neutral mode should be the next to last particle on the hypothesis card, and the K_{long} should be the last one. To do a fit one has to

- set the integer variable KSKBCF to be the TTKS id of the K_{long}, in MEASCO this id should be the last one
- set the real array CFVTER(1/2/3) to be the error squared of the "vertex measurement", the defaults are 1.0 for all of them
- write an appropriate ... KSH KL hypothesis card
- set the real array element CFMIER(6) to be the "error squared of $\sqrt{(}E)$ of the K_{long} ", the default is 3.

4.8 Neutral K°_{short} vertex scan

This is a CASE=1 basic fit which includes some tuning of the data to the presumed neutral secondary vertex. No missing particle is allowed here, i.e. there must be another charged K°_{short} in the event, whose charged pion paparmeters you can also tune to that second secondary vertex (called charged correction in section 4.5). To do a fit like this one has to

- set the logical variable CFSCAN to be true
- set the real array VRTKCF(1/2/3) to be the presumed
- put the neutral KSH as the last particle on the hypo card

I also propose to supress the combinatorics (see section 5.2).

5 Application of the user

Following the instructions of this section you may refill the commons in the same event. If so you must

- set the logical variable CFORCE to be .TRUE.

which forces CBKFIT to refill the commons from the TTKS bank for every CFDOFI call. The default is to fill them once for each event.

5.1 Suppression of hypothesis looping

If you do not want the fitter to loop over all the hypotheses you can select an individual hypothesis

- setting the integer variable CFTAKH to the hypothesis number according to the ordering on the hypothesis input.

This is the hypothesis number of the actual event type. If CFTAKH=0 (default) then the fitter loops over all the hypotheses. You can change the value of CFTAKH and call CFDOFI several times in an event.

5.2 Suppression of combinatorics

If a hypothesis has plenty of combinatoric cases then it may be economic to preselect them by the user. To select individual ordering of particles and so suppress the automatic generation of the combinatoric cases one has to

- set the logical variable CFSUPC to .TRUE.
- set the integer array NXTCMB to the expected order of particles

See subsections 7.1 and 7.2 of this manual for the meaning of the NXTCMB, BASEC and MEASCO arrays.

6 Setting beam and target conditions

The default is a beam stopped in hydrogen e.g. neutrons missing from the target.

6.1 Fit of reactions in flight

If you want to fit events with a beam of nonzero momentum then you have to

- set the real variable CFBEAM to the beam momentum

You can use all the other options as you like.

6.2 Fit of reactions with deuterium target

If this case you must

- set the logical variable CFDEUT to be .TRUE.

You can use all the other options as you like but you need either MISNEU or MISPRO which restrict this freedom.

6.3 Fit of events coming from the collider option

- set the logical variable CFCOLL to be .TRUE.
- set the logical variable CFBEAM to the momentum of the proton and antiproton beam (if it is zero you have annihilations at rest)

You can use all the options of CBKFIT.

7 User Routines

You can call the main fit program CFDOFI several times for an event which calls user routines to allow for control. Simple examples are given the CMZ file. It loops over all the hypotheses of the event type set by the user particle selection routine CFUSRD. If you want to reselect particles then call CFDOFI again and in CFUSRD change MEASCO, the event type, etc.

cfusrd particle selection

cfusco combinatorics control, mass preselection

cfusou take output after a succesful fit

7.1 Subroutine CFUSRD

Arguments: IRET
This routine is called once for each CFDOFI call by the routine CFREA which takes particles from CBBANK to fit common blocks. You can select particles filling up the array MEASCO with the accepted particle numbers of the TTKS bank. You may reorder the particles if you wish. Set NPOS, NNEG and NNE to the number of positive, negative and neutral particles accepted from the TTKS bank. NPOS should be equal to NNEG or in the case of annihilation with neutron NNEG-1. Set IRET to be negative if you do not want to analyse this event and zero if yes.

For example if TTKS contains 8 particles of particle ids 1 to 8 and you want to drop 2 particles namely particle 3 and 7 then you have to fill the following numbers to MEASCO : 1, 2, 4, 5, 6, 8, and the number of particles will be set to 6 automatically.

7.2 Logical function CFUSCO

Arguments: IHY
This routine is called once for each combinatorial case generated automatically by the fitter from the hypothesis input. IHY is the hypothesis sequence number of the actual event type in the input. Set CFUSCO to false if you want to drop the combination which appears in the array NXTCMB.

It is recommended to do here a mass preselection calling the function CFMPRE. See the example in appendix A. If you do not want to be an exceptional expert of CBKFIT, you can skip the rest of this subsection.

The particle ids are reordered according to the actual hypothesis what basic combinations are stored in the array BASEC(I,IHY) in order of appearance on the hypothesis card. BASEC refers to the content of MEASCO. A resonance is replaced by the resonance card but the order of particles remaines the same. Nontrivialities occur because of the different charge of particles. This array is filled up by CBKFIT but you must know the method if you want to switch off the combinatorics. First it takes the first particle of the final state, say it is a negative particle. Then it looks for the first negative particle in MEASCO, say it is the 4th. So the first element of BASEC is 4. Then it takes the second particle of the final state, say it is a photon. Then it looks for the first photon in MEASCO, say it is the first. So the second element of BASEC is 1. Then it takes the third particle of the final state, say it is a negative particle again. Then it looks for the second negative particle in MEASCO, say it is the 5th. So the third element of BASEC is 5. And so on. Follow this method to set up NXTCMB if you switch off combinatorics as described in 5.2 .

Example 1: if the hypothesis is

```
HYPO G G G G G
```

then BASEC contains 1, 2, 3, 4, 5

Example 2: if the hypothesis is

```
HYPO OM OM
 RES1 OM -> PI0 G
 RES1 OM -> PI0 G
  RES2 PI0 -> G G
  RES2 PI0 -> G G
```

then BASEC contains 1, 2, 3, 4, 5, 6

Example 3: if the hypothesis is

```
HYPO RO+ RO-
 RES1 RO+ -> PI+ PI0
 RES1 RO- -> PI- PI0
  RES2 PI0 -> G G
  RES2 PI0 -> G G
```

and MEASCO contains TTKS ids which have charges of +, -, 0, 0, 0, 0 then BASEC contains 1, 3, 4, 2, 5, 6 because the hypothesis in the final state is PI+ G G PI- G G

The array NXTCMB(I) contains the next combination of the BASEC content so if you want to reach TTKS ids then take MEASCO(NXTCMB(I)). In the third example MEASCO(NXTCMB(2)) and MEASCO(NXTCMB(3)) give the TTKS id of photons coming from the π° of the ρ^+ .

The arrays FITMAS(I,IHY) and FITCHA(I,IHY) contain the masses and the charges of particles used by the fit in the same order as they are stored in BASEC and permutated by NXTCMB.

The array RESMAS(I,IHY) containes the masses of resonances in order of appearance on the hypothesis card. In this sense an ω resonance starts first itself then the π° coming from it so the order of resonances in the second example will be ω π° ω π° .

The array RDAUGH contains the daughter particles of resonances. RDAUGH(0,I,IHY) gives the number of elements of the *ith* resonance in RESMAS. RDAUGH(I,J,IHY) gives the NXTCMB/BASEC index of the *jth* element of the *ith* resonance that is the TTKS ids of the daughter particles of a resonance are MEASCO(NXTCMB(RDAUGH(I,J,IHY))) .

7.3 Subroutine CFUSOU

Arguments: IHY,GODNES
This routine is called once for each combinatoric case of each hypothesis, regardless whether fit was tried or not, because of preselection, or the fit was succesful or not, to allow to take the results. The integer variable CFCODE(100) in the common area CFTEST describes the termination of the fit :

element :

```
1  =  1 : succesful fit, 0 : fit not tried, -1 : unsuccesful
2  =                     0 :               , -1 : GODNES < CUTCL
3  =                     0 :               , -1 : dropped by CFMPRE
4  =  last iteration step started
5  =  iteration stop parameter * 10^9  ( should be < 1000 )
6  =  chisquared * 1000
...
long term parameters
14  =  number of combinations generated
15  =  number of high probability fits (with respect to CUTCL)
16  =  number of low probability fits (with respect to CUTCL)
17  =  number of unsuccesful fits
18  =  number of fits tried
19  =  0 : track errors are taken from TTKS   else : not
20  =  0 : PED   errors are taken from TTKS   else : not
21  =  number of low  c.l. fits 1st hyp. no missing particle
22  =  number of high c.l. fits 1st hyp. no missing particle
23  =  number of low  c.l. fits 2nd hyp. no missing particle
24  =  number of high c.l. fits 2nd hyp. no missing particle
25  =  number of low  c.l. fits 3rd hyp. no missing particle
26  =  number of high c.l. fits 3rd hyp. no missing particle
27  =  number of low  c.l. fits 4th hyp. no missing particle
28  =  number of high c.l. fits 4th hyp. no missing particle
```

```
29  =  number of low  c.l. fits 1st hyp.    missing particle
30  =  number of high c.l. fits 1st hyp.    missing particle
31  =  number of low  c.l. fits 2nd hyp.    missing particle
32  =  number of high c.l. fits 2nd hyp.    missing particle
33  =  number of low  c.l. fits 3rd hyp.    missing particle
34  =  number of high c.l. fits 3rd hyp.    missing particle
35  =  number of low  c.l. fits 4th hyp.    missing particle
36  =  number of high c.l. fits 4th hyp.    missing particle
```

the rest has not been filled yet.

IHY is the hypothesis sequence number at the actual event type. GODNES is the goodness of the fit. All the other parameters are found in the common blocks. See CFUSCO for the description of arrays. If you want to save this succesful fit into a KRES subbank then set FPUTZ=.TRUE. in this routine. If you want to reject fits having a confidence level less than a certain value then set CUTCL to this value.

8 Description of routines

8.1 Calling sequence of the software

See figure 1.

9 Kinematic fit output bank format

The results of the kinematic fitting are stored in a ZEBRA structure. The name of the top bank is KRES, the link pointing to it is LKRES. The different hypotheses are stored in the data part of the header bank. Each hypothesis has a reference link pointing to it. If the logical variable FPUTZ is set to be .TRUE. in the user output routine CFUSOU then the actual fit results are stored as KSUB subbanks to KRES, i.e. structural link 1 points to the the fit results for hypothesis 1. If there are several good fits for one hypothesis, the banks for these fits form a linear chain.

9.1 The KRES Bank

This bank stores all hypotheses in the data part, and links to the results as structural links. Reference links point to the different hypotheses within the data part of this bank.

The hypotheses are stored in a compact but still readable format. There is one word per particle, including decaying particles. Each particle is identified by its GEANT particle id. Decaying particles create a 'vertex' with an id. Also each particle is created at a vertex with a given id. The first vertex is called 0. The word describing one particle has these three numbers added together

$$\text{id} + 100 \times (\text{ vertex id}) + 10,000 \times (\text{ id of decay vertex})$$

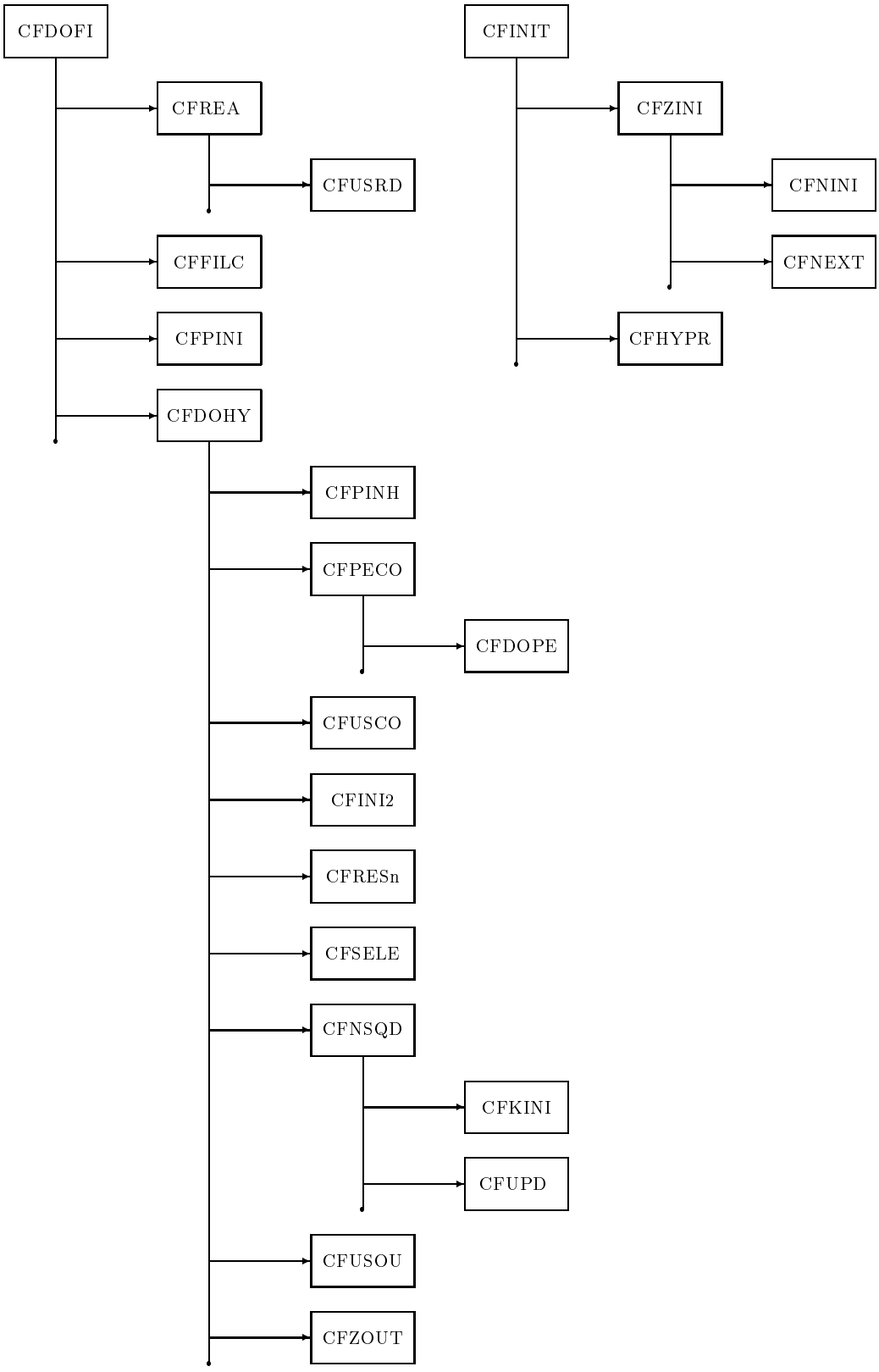

Figure 1: Calling sequence of the kinematic fitting software

Using this convention, stable particles are easily identified by having a hypothesis word less than 1000. For example:

$$\bar{p}p \rightarrow \pi^+\pi^-\omega \rightarrow \pi^+\pi^-\pi^\circ\gamma \rightarrow \pi^+\pi^-\gamma\gamma\gamma$$

would have for the 6C fit (π°, ω used as constraints) the following hypothesis :

8	π^+
9	π^-
10060	ω
20107	π°, from ω
101	γ, from ω
201	γ, from π°
201	γ, from π°

Where the ω has id 60, comes from vertex 0 and created vertex 1, the π° has id 7, was creted at vertex 1, and creates vertex 2, the γ has id 1, the first one originates from vertex 1, the next two from vertex 2, finally π^+ has id 8, and π^- has id 9, and they both come from vertex 0.

The order of hypothesis words so corresponds to the order of appearance on the HYPO and RES1,RES2,... cards, i.e. resonances and stable particles are mixed, but "follow time evolution".

The KRES bank has a leading part and a trailing part (see the ZEBRA manual for definitions). You can acces to the Ith word of the leading part in the usual IQ(LKRES+I) way, then the trailing part is repeated according to the number of hypotheses. To make it easy to acces to the trailing part, i.e. the hypothesis words, use the reference links. There are as many reference links as the number of hypotheses you wrote on the hypothesis input (number of HYPO cards).

You can acces to the Jth hypothesis, i.e. the first word of the Jth cycle of the trailing part as IKRES=IQ(LQ(LKRES-IQ(LKRES-2)-J)), and then for this hypothesis :

- IQ(IKRES) is the number of degrees of freedom
- IQ(IKRES+1) is the number of stored fits
- Q(IKRES+2) is the confidence level cutoff
- IQ(IKRES+3) is the number of particles (including resonances and missing particles i.e. it is NPART+NRES), i.e. the number of hypothesis words
- IQ(IKRES+4) is the first hypothesis word
- IQ(IKRES+3+I) is the Ith ...

The KRES bank has then the following as given in table I.

9.2 The Sub-banks to KRES

The subbanks of KRES (called KSUB) store the results for the fits. There is exactly one bank per fit. The confidence level CL and the χ^2 are available for every fit. For each final state particle the four momentum and the pulls for the measured variables are kept. For resonances pulls are set to be zero. The ordering of the particles is identical to that is the hypothesis description in the bank KRES. The details of the bank contents are given in Table II.

The address of the sub-bank chain containing the results of the Jth hypothesis is LQ(LKRES-J), the number of the elements in the linear chain (e.g. the number of combinatorial cases which yield good CL for this hypothesis) is IQ(LQ(LKRES-IQ(LKRES-2)-J)+1). If there are more than one succesful fits for this hypothesis, then the next link of the first bank (LQ(LQ(LKRES-J))) differs from zero.

You can loop over the good fits of the Jth hypothesis with the following code :

```
      NGF(J)=IQ(LQ(LKRES-IQ(LKRES-2)-J)+1)
      JKRES=LQ(LKRES-J)
      DO 10 K=1,NGF(J)
         .
         .
         .
```

here the address of the bank containig the fit results is JKRES

```
         .
         .
         .
      JKRES=LQ(JKRES)
10    CONTINUE
```

and you can check that after the DO loop JKRES=0 .

The KSUB bank has a leading and a trailing part (see the ZEBRA manual for definitions. You can access the Ith word of the leading part in the usual Q(JKRES+I) way, i.e.

- Q(JKRES+1) is the confidence level of this fit
- Q(JKRES+2) is the chisquared
- Q(JKRES+3) is the fitted vertex x-coordinate
- Q(JKRES+4) ...
- Q(JKRES+9) is number of particles (identical to IQ(IKRES+3) of the previous paragraph describing the KRES top bank)

Inside the KSUB bank you can loop over the particles with the help of the reference links of this bank. There are no structural links in this bank (because there are no sub-banks to it) so you can acces to the Ith particle (to the word preceeding its TTKS id) as KKRES=IQ(LQ(JTTKS)-I), therefore

- IQ(KKRES+1) is the TTKS id of the particle
- Q(KKRES+2) is the mass
- Q(KKRES+3) is the energy
- Q(KKRES+4) is p_x
- Q(KKRES+5) ...

The Ith particle here corresponds to the Ith hypothesis word of the previous subsection (i.e. to IQ(IKRES+3+I)) describing the KRES top bank.

10 Particle IDs

I give here a list of the particles id's, which are used to describe the hypothesis for the fits. These follow the GEANT convention including Crystal Barrel extensions.

1	γ	G
2	e^+	$E+$
3	e^-	$E-$
4	ν	
5	μ^+	$MU+$
6	μ^-	$MU-$
7	π°	$PI0$
8	π^+	$PI+$
9	π^-	$PI-$
10	K°_{long}	KL
11	K^+	$K+$
12	K^-	$K-$
13	n	N
14	p	P
15	$\bar{p}$	PB
16	K°_{short}	KSH
17	η	ETA
57	$\rho^\circ(770)$	$RO0$
58	$\rho^+(770)$	$RO+$
59	$\rho^-(770)$	$RO-$
60	$\omega(785)$	OM
61	$\eta'(957)$	$ETAP$
?	$\Phi(1020)$	FI

11 Debugging

You can set some logical variables to be true anywhere in your user program, globally or selectively and you get some more information about the happenings is the fitter

- CFTRSU yields one printed line per succesful fit
- CFTRHY traces the fait of all hypotheses and combinatoric cases
- CFDBUG yields a very detailed printout

A An Example Using CBKFIT

After many many questions to the Pal Hidas regarding CBKFIT, I finally got it running. I would be the first to admit that the startup was rather painful, but the rewards are well

worth the trouble. The package is a very powerful tool, and in the end quite easy to use. However, as my start up effort was so large, I felt it would be useful to append this to the CBKIFT manual to save Pal the difficulty to responding to my questions again and again. Any questions about this appendix should be addressed to Curtis, as I would be responsible for any misinformation herin.

What I am going to describe here is a detailed example for fitting a mixed charged and neutral final state. I am interested in the final state $\pi^+\pi^-6\gamma$, and in particular in $\pi^+\pi^-3\pi^\circ$. The following will be a description of exactly how I use CBKFIT.

First and formost, one needs the hypothesis cards, which should be assigned to logical unit 32, (defined as LYHP in CBUNIT). For my case I first ask for the $\pi^+\pi^-3\pi^\circ$ hypothesis, however, as a background I consider $\pi^+\pi^-2\pi^\circ\eta$. Finally, I am interested in the results of the simple 4–C fit to $\pi^+\pi^-6\gamma$. To examine the three hypothesis, I use the following cards:

```
ETYP 1 0 1 0 6 0
HYPO PI+ PI- PI0 PI0 PI0
RES1 PI0 -> G G
RES1 PI0 -> G G
RES1 PI0 -> G G
HYPO PI+ PI- PI0 PI0 ETA
RES1 PI0 -> G G
RES1 PI0 -> G G
RES1 ETA -> G G
HYPO PI+ PI- G G G G G G
END
```

Following the input cards, it is necesary to properly initialize CBKFIT. This is done by calling the CFINIT routine from my USINIT routine. There is also **one** CMZ KEEP which is interesting at this point, +CDE,CFCOMS, and it is necessary to provide sufficient memory in the /CFCSTR/ common block. Sufficient memory is defined as the number of combinatorical cases times the number of particles. I have eight particles, ($\pi^+\pi^-6\gamma$). Also, for the three–π° hypothesis, there are 15 possible ways to form this. For the second, there are 30 ways to form this, and for the third, there is exactly one. This gives 46 possible combinations. This means we need to have at least 368 words in the common block. One other thing which can be set here is the value of CUTCL in the CFCOMS KEEP. This is a probability cutoff that determines which combinatorical cases get sent to the CFUSOU routine. As I would like to examine the confidence level of all fits, I set this to zero. I will then make my confidence level decesion entirely in the CFUSOU routine.

```
      SUBROUTINE USINIT
         ...
      INTEGER         NWRD
      PARAMETER      (NWRD=1000)
      COMMON /CFCSTR/ ICMBPR(NWD)
+CDE,CFCOMS.
         ...
      CALL CFINIT(NWRD)
```

```
      CUTCL = 0.0
         ...
      END
```

The next thing which I need to provide is the CFUSRD routine. If I always wanted to use all the particles in the **TTKS** banks, then this routine could be a dummy. However, in my case I have some apriori information that certain of the *unmatched* photons should really be treated as charged split–offs, and ignored by CBKFIT. We will also assume that the list of valid global tracking numbers is passed through a common block in the variable LISTUS(8), (I always should have eight particles I need to use.)

These global tracking numbers need to be copied into the MEASCO array. However, as it is not true that I always use all the particles in global tracking, I first need to zero the MEASCO array. Also, because I have changed MEASCO, it is also necessary to make sure that the number of positive, negative and neutral particels in MEASCO are recorded correctly in the NPOS, NNEG and NNE variables.

```
      SUBROUTINE CFUSRD(IRET)
      ...
      COMMON /USJUNK/ LISTUS(8)
+CDE,CFCOMS.
      ...
      CALL VZERO(MEASCO,NPMAX)
      ...
      DO 100 I = 1,8
         MEASCO(I) = LISTUS(I)
 100  CONTINUE
      ...
      NPOS = 1
      NNEG = 1
      NNE  = 6
      ...
      END
```

The next routine that I could provide is CFUSCO. However, as I do not want to supress any of the combinatorical cases, I just have a dummy routine.

```
      LOGICAL FUNCTION CFUSCO(IHY)
+SEQ,CFCOMS.
      LOGICAL CFMPRE
      CFUSCO = .TRUE.
      IF(.NOT.CFMPRE(IHY)) THEN
       CFUSCO=.FALSE.
       RETURN
      ENDIF
      RETURN
      END
```

Finally, I need to provide a CFUSOU routine to identify which combinatorical cases should be saved in a kinematic fit bank. In my case, I want to save all cases of hypothesis one and two which are larger than a cutoff P7CTUS. I never want to save the third hypothesis, but I want to histogram it. For every combination I want to save, I must set FPUTZ .TRUE., while for those that I want to reject, I set FPUTZ .FALSE..

```
       SUBROUITNE CFUSOU(IHY,CL)
       ...
       COMMON /USCUTS/ P7CTUS
+CDE,CFCOMS.
       ...
       FPUTZ = .FALSE.
       ...
       IF(IHY .EQ. 1) THEN
          CALL HF1( 1,CL,1.)
          IF(CL .GE. P7CTUS) FPUTZ = .TRUE.
       IF(IHY .EQ. 2) THEN
          CALL HF1( 2,CL,1.)
          IF(CL .GE. P7CTUS) FPUTZ = .TRUE.
       ELSE
          CALL HF1( 3,CL,1.)
       ENDIF
       END
```

I am now ready to use CBKFIT. To do this, I will need to simply call CFDOFI from my USER routine. However, before I do this, it may be desireable to modify the errors on track or crystal quantities. The CBKFIT routine takes its information directly out of the **TTKS** bank. As such, If I want to change anything, I must do it in the **TTKS** before I call CFDOFI. I should also restore that information after I have called CBKFIT so I don't get into the nasty situation of applying the same correction twice.

In my case, it is known that the errors on the tracks need to be expanded. I will not explain precisely what all the factors mean, but instead just expalin how this correction needs to be done. I also want to say at this time that the first element in the LISTUS array, (see CFUSRD) always points to the π^+ and the second element always point to the π^-. I will then provide a dummy loop for extracting the best combination of hypothesis one, just to show how the data can be accessed.

```
      SUBROUTINE USER
       ...
*         ALPHUS : Scale the error in alpha :
*         TGLMUS : Scale the error in tan{lambda} :
*         PSIFUS : Scale the error in psi :
*
       ...
      REAL ALPHUS,TGLMUS,PSIFUS
      PARAMETER (ALPHUS = 2.00)
```

```
      PARAMETER (TGLMUS = 1.50)
      PARAMETER (PSIFUS = 3.00)
       ...
      COMMON /USJUNK/ LISTUS(8)
+CDE,CFCOMS.
+CDE,CBLINK.
       ...
*
*---Fix the errors on the charged tracks.
*
      DO 100 I = 1,2
*
         JTTKS = LQ(LTTKS - IQ(LTTKS-2) - LISTUS(I) )
*
         Q(JTTKS+46) = Q(JTTKS+46) * ( PSIFUS * PSIFUS )
         Q(JTTKS+47) = Q(JTTKS+47) * ( PSIFUS * ALPHUS )
         Q(JTTKS+48) = Q(JTTKS+48) * ( ALPHUS * ALPHUS )
         Q(JTTKS+49) = Q(JTTKS+49) * ( PSIFUS * TGLMUS )
         Q(JTTKS+50) = Q(JTTKS+50) * ( ALPHUS * TGLMUS )
         Q(JTTKS+51) = Q(JTTKS+51) * ( TGLMUS * TGLMUS )
*
 100  CONTINUE
```

Now I will perform the kinematic fits. I then want to restore the **TTKS** banks to the state they were in before I did my fits.

```
      CALL CFDOFI
*
*---Unfix the errors on the charged tracks.
*
      DO 200 I = 1,2
*
         JTTKS = LQ(LTTKS - IQ(LTTKS-2) - LISTUS(I) )
*
         Q(JTTKS+46) = Q(JTTKS+46) / ( PSIFUS * PSIFUS )
         Q(JTTKS+47) = Q(JTTKS+47) / ( PSIFUS * ALPHUS )
         Q(JTTKS+48) = Q(JTTKS+48) / ( ALPHUS * ALPHUS )
         Q(JTTKS+49) = Q(JTTKS+49) / ( PSIFUS * TGLMUS )
         Q(JTTKS+50) = Q(JTTKS+50) / ( ALPHUS * TGLMUS )
         Q(JTTKS+51) = Q(JTTKS+51) / ( TGLMUS * TGLMUS )
*
 200  CONTINUE
*
```

Now I can examine exactly what came back from CBKFIT. Initially I want to make sure that there is a bank containing fit data. I then would like to know how many hypothesis one and hypothesis two combinations were saved.

```
      IF(LKRES .LE. 0) RETURN
*
*---Find out how many good fits for hypothesis one are stores.
*
      JKRES = LQ(LKRES - IQ(LKRES-2) - 1)
      N3PI  = IQ(JKRES + 1)
*
*---Repeat for hypothesis two.
*
      JKRES = LQ(LKRES - IQ(LKRES-2) - 2)
      N2PIE = IQ(JKRES + 1)
*
```

At this point, I am only interested in the best combination for hypotheis number one. In order to figure out which one this is, I will need to sort all the stored combinations for hypothesis one by their confidence levels.

```
      KKRES = LQ(LKRES - 1)
*
      DO 600 I = 1,N3PI
*
         PRBLA(I) = Q(KKRES + 1)
         CHSQR(I) = Q(KKRES + 2)
         INDX(I) = I
         KKRES = LQ(KKRES)
*
 600  CONTINUE
*
      IF(N3PI .GT. 1) CALL SORTFL(PRBLA,INDX,N3PI)
      CHSQ  = CHSQR(INDX(N3PI))
      PROBA = PRBLA(INDX(N3PI))
```

At this point I have INDX(N3PI) pointing to the best confidence level. I now need the structural link to the corresponding subbank of **KRES**.

```
      JKRES = LQ(LKRES-1)
      IF(INDX(N3PI) .GT. 1) THEN
         DO 900 I = 2,INDX(N3PI)
            JKRES = LQ(JKRES)
 900     CONTINUE
      ENDIF
```

Because ot the form of my hypothesis cards, I know that there should be 11 particles stored in this subbank. They are π^+, π^-, then $3\pi^\circ$'s, and finally the 6γ's. The γ's will be sorted in such a way that the first two belong to the first π°, the third and fourth will belong to the second π°, while the last two will belong to the third π°. I can now loop through the eleven particles in this bank, and extract the needed quantities.

```
      KKRES = JKRES + 15
```

```
      DO 1500 I = 1,IQ(JKRES+15)
         IF(IQ(KKRES+1).LE.0 ) THEN        ! Check for pi-0
            EPI0  = Q(KKRES + 3)
            PXPI0 = Q(KKRES + 4)
            PYPI0 = Q(KKRES + 5)
            PZPI0 = Q(KKRES + 6)
         ELSE
            IF(Q(KKRES+2) .LT. 10.0) THEN  ! Check for Photon.
               EGAM  = Q(KKRES + 3)
               PXGAM = Q(KKRES + 4)
               PYGAM = Q(KKRES + 5)
               PZGAM = Q(KKRES + 6)
            ELSE                           ! Charged pion.
               EPIQ  = Q(KKRES + 3)
               PXPIQ = Q(KKRES + 4)
               PYPIQ = Q(KKRES + 5)
               PZPIQ = Q(KKRES + 6)
            ENDIF
            PULL1 = Q(KKRES + 7)
            PULL1 = Q(KKRES + 8)
            PULL1 = Q(KKRES + 9)
         ENDIF
         ...
         KKRES = KKRES + 9
 1500 CONTINUE
      END
```

Offset	TYPE	*Quantity*
reference links		
⋮	⋮	⋮
−2	LINK	pointer to second hypothesis, ie. data word i
−1	LINK	pointer to first hypothesis, ie. data word 17
structural links		
⋮	⋮	⋮
−2	LINK	link to bank for fit results for second hypothesis
−1	LINK	link to bank for fit results for first hypothesis
data part of bank		
+1	INTEGER	Version number of CBKFIT
+2	INTEGER	Number of words per particle in subbanks
+3	INTEGER	Number of hypotheses
+4	REAL	Charged vertex x from locater (or 0)
+5	REAL	Charged vertex y from locater (or 0)
+6	REAL	Charged vertex z from locater (or 0)
+7	REAL	Charged vertex x-momentum from locater (or 0)
+8	REAL	Charged vertex y-momentum from locater (or 0)
+9	REAL	Charged vertex z-momentum from locater (or 0)
+10	REAL	Charged vertex x from locater (or 0)
+11	REAL	Charged vertex y from locater (or 0)
+12	REAL	Charged vertex z from locater (or 0)
+13	REAL	Charged vertex x-momentum from locater (or 0)
+14	REAL	Charged vertex y-momentum from locater (or 0)
+15	REAL	Charged vertex z-momentum from locater (or 0)
+16	REAL	Main vertex z coordinate (from Newton iteration)
repeat the next words for every hypothesis		
+17	INTEGER	Number of degrees of freedom for this hypothesis, zero if no succesful fit
+18	INTEGER	Number of good fits for this hypothesis = number of hypothesis words
+19	REAL	Confidence level cutoff for this hypothesis, zero if no succesful fit
+20	INTEGER	Number of particles first hypothesis
+21	INTEGER	hypothesis words
⋮	⋮	⋮
$+i$	INTEGER	Number of degrees of freedom for the second hypothesis, zero if no succesful fit
⋮	⋮	⋮

Table 1: The data stored in the **KRES** top bank.

Offset	TYPE	*Quantity*
reference links		
⋮	⋮	⋮
$-i$	LINK	pointer to the word preceeding the TTKS id of particle i
⋮	⋮	⋮
data part of bank		
+1	REAL	Confidence level CL
+2	REAL	χ^2
+3	REAL	Vertex x
+4	REAL	Vertex y
+5	REAL	Vertex z
+6	REAL	Pull of x
+7	REAL	Pull of y
+8	REAL	Pull of z
+9	REAL	Vertex x
+10	REAL	Vertex y
+11	REAL	Vertex z
+12	REAL	Pull of x
+13	REAL	Pull of y
+14	REAL	Pull of z
+15	INTEGER	N, *number of particles*
repeat the next words for every particle		
+16	INTEGER	$TTKS$ *particle id*, 0 *if resonance*, -1 *if missingparticle*
+17	REAL	$Mass$ *used in fit*
+18	REAL	$Energy$ *from fit*
+19	REAL	p_x *momentum from fit*
+20	REAL	p_y *momentum from fit*
+21	REAL	p_z *momentum from fit*
+22	REAL	$Pull$ *for* ψ *for charged tracks,* ϕ *for showers,* 0 *if resonance or missing particle*
+23	REAL	$Pull$ *for* $1/P_{xy}$ *for charged tracks,* θ *for showers,* 0 *if resonance or missing particle*
+24	REAL	$Pull$ *for* $\tan(\lambda)$ *for charged tracks,* $\sqrt{(}E)$ *for showers,* 0 *if resonance or missing particle*
⋮	⋮	⋮

Table 2: The data stored in the subbank of the **KRES** bank. Words 16 to 24 are repeated for every particle