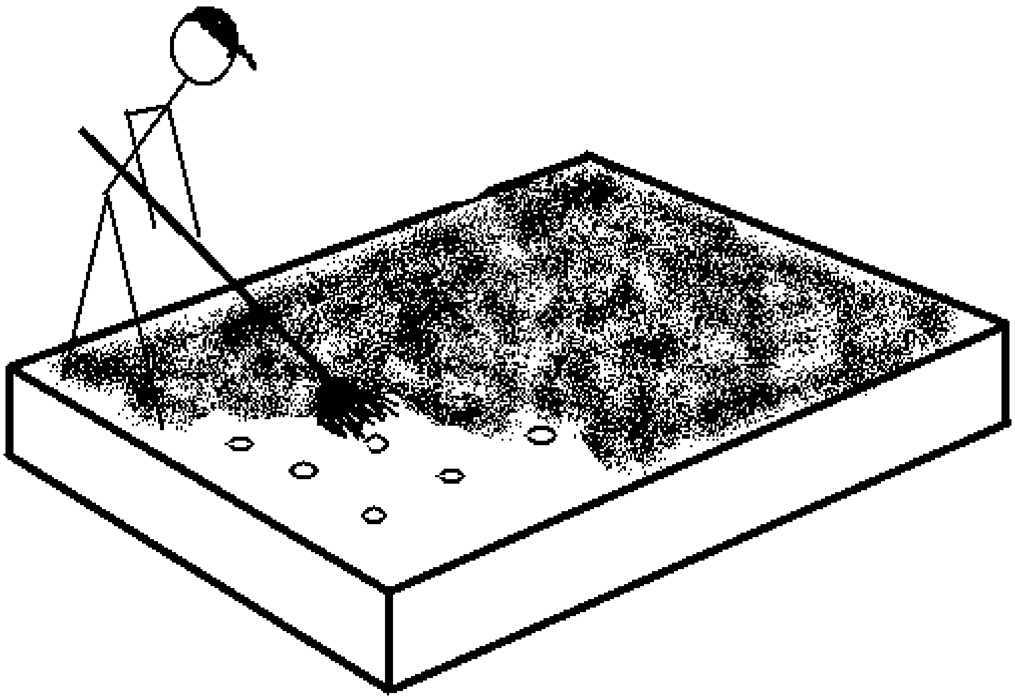
The Hall C Matrix Element Optimization Package (CMOP)

Dipangkar Dutta, Pat Welch

July 10, 1996



Abstract

An attempt to answer, in plain English (sic), the eternal questions What, Why, Where and How?

What and Why?

The hall C spectrometers HMS and SOS both use a pair of wire chambers for tracking, these chambers give us two positions (x, y) and two angles (ϕ, θ) on a plane exactly midway between the two chambers. This is our nominal focal plane, from this plane one can transform to the optical focal plane by simple rotation and drift. From the optical focal plane one can reconstruct the position and momentum of the event at the target via taylor expansion in terms of the four focal plane track quantities. The five target quantities being reconstructed are two positions (x and y, x is always taken as zero), two angles (ϕ, θ) and delta $(p - p_o/p_o)$. The coefficients of the taylor expansion are the so called matrix elements we want to optimize. The original set of coefficients are generated by modelling the spectrometers magnetic fields using the ever popular COSY. But since life is so full of misery and suffering, these matrix elements need to be optimized for the true magnetic fields, using actual data. The hardware used for this 'oh so noble' goal is a holy(pun intended) piece of densimat, -the sieve slits. To alleviate some of the extreme pain and tedium involved in optimizing the matrix elements, a simple least squares fitting package and a mapping package was interfaced with a graphics package (Paw++). Using data in standard ntuple format this program performs least squares fits to determine the optimum matrix elements.

Where?

The source code for this package has been split into two parts one for the SOS and one for the HMS. Although the functional form of both are exactly same they have been so divided to avoid confusion. You can copy the complete source code from ~ddutta/sos_ma for the SOS and ~ddutta/hms_ma for the HMS. The latest version is always kept in the topmost directory, and the other versions in subsequent directories.

How?

This is bound to be the longest section, so lets break it up.

Input files

All the input files are either in the form of database files [name].db, or in the form of macro files [name].kumac. The first macro you need to execute as soon as you begin is sstart.kumac for the SOS or hstart.kumac for the HMS. These read in the ntuple with data you wish to use, initialize the starting matrix elements, various offsets and other constants. It reads in three database files matrixelements.db, sieveslit.db and target.db. The first contains the starting values of all the matrixelements, all the detector offsets, the z position of the true focal plane and the transformation coefficients to the optical focal plane. sieveslit.db contains the position of each hole in the sieve slit and their diameters and the drift distance between the target and the slit. The last file contains various constants like the spectrometer angle, inclination of the slanted target, delta steps used in the delta scan, the total number of ntuples merged and number of events in each of the merged ntuples. The other important input files are the various fitting macros, eg. fittheta.kumac etc. These contain the list of matrixelements you wish to fit and those which you want to keep fixed. Since we fit the four sets of matrixelements seperately there are four such files for each spectrometer. fittheta.kumac, fitphi.kumac,fitytar.kumac and fitdel.kumac. It is advisable to browse through these input files in order to tune your self to the notations used, believe me its a truly Zen experience. The data to be used in fitting must be in ntuple form. Ntuples designed for optics studies can be obtained by running the engine with the following lines included in your REPLAY.PARM -

hms_sieve_ntuple = '[file name]'
sos_sieve_ntuple = '[file name]'

Mapping routines

Before we get to the fitting part let me tell you about an intermediate step which maps out each event from the focal plane to the slit, in order to determine whether that event would make it through any of the holes in the sieve slit. If it does pass through any of the holes, the position of the hole gives us the scattering angle ϕ(in HallC notation) and the out of plane angle θ(again in HallC notation) for that event. This mapping is done in three stages, first map_fp_ofp.f maps each event from the focal plane to the so called optical focal plane, which basically consists of adding the various offsets and drifting to the z position of the true focus. Next each event is mapped from the optical focal plane to the target by map_ofp_targ.f, here all the starting matrixelements are used to do this mapping. Lastly the events are mapped from the target to the slit, this involves just drifing from the target to the slit, the relevant routine is map_targ_slit.f.

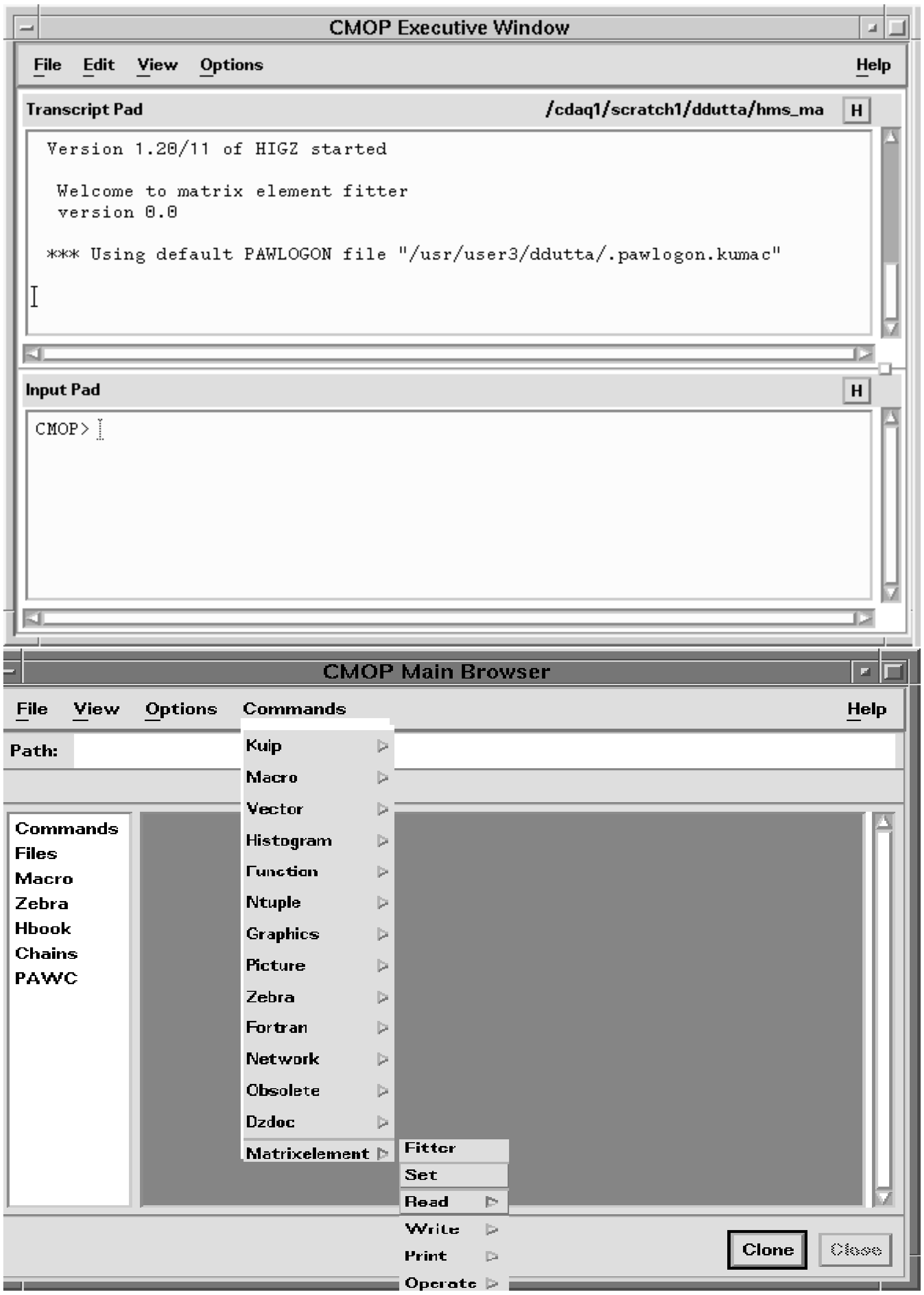
Pre fitting tune up

Before you plunge into the nether world of fitting, it is advisable to do this small tune up, which is basically to make sure you have all the offsets correct. Once you have the program running, at the CMOP> prompt (see figure below) execute the macro sstart or hstart. The sequence of commands is

CMOP> *exec sstart.kumec or hstart.kumac*

CMOP>*fitter [Ntuple ID]*

Note *:the last command 'fitter' can be accessed by clicking on the command button in the CMOP browser, and then selecting the appropriate buttons, just like any typical Paw++ command. Also all plotting is done exacly as in Paw++, see figure below.*


CMOP Executive Window
File Edit View Options Help
Transcript Pad /cdaq1/scratch1/ddutta/hms_ma H
Version 1.20/11 of HIGZ started
Welcome to matrix element fitter
version 0.0
*** Using default PAWLOGON file "/usr/user3/ddutta/.pawlogon.kumac"
Input Pad H
CMOP>
CMOP Main Browser
File View Options Commands Help
Path:
Commands
Files
Macro
Zebra
Hbook
Chains
PAWC
Kuip
Macro
Vector
Histogram
Function
Ntuple
Graphics
Picture
Zebra
Fortran
Network
Obsolete
Dzdoc
Matrixelement
Fitter
Set
Read
Write
Print
Operate
Clone
Close

This does a zero parameter fit ie. simply maps all the events from the focal plane to the slit, and creates a new ntuple (ID 200), with the focal plane quantities (x, y, xp, yp) transformed to the optical focal plane, the target and the slit. They all have obvious suffix fp, ofp, targ and sli respectively. Now one can plot various fp quantities to determine the offsets, or if you already know the offsets you can check if they had the desired effect by looking at the corresponding plots of the ofp quantities. Some examples are: plot x_fp vs y_fp in order to determine the detector offsets, the neck of the hour glass must be centered at $(0, 0)$. Next plot xp_fp and try to center it about zero, this gives you the angular offset ang_offset_x. Similarly one can find ang_offset_y with the plot of yp_fp. Now one can apply these offsets without quitting the program by editing the matrixelements.db file and changing the relevant offsets and coefficients. In order the check the effect of these changes just use the following command sequence

CMOP>*exec referesh* (reads in the changes in matrixelements.db)

CMOP>*fitter [Ntuple ID]* (redo the mapping)

Now one can look at 2d plots x_ofp vs y_ofp to see if the hour glass neck is centered. Then look at xp_ofp and yp_ofp to see if they are all centered. One can repeat this process till one is satisfied or passed out.

Fitting *(whew! at last)*

As I mentioned earlier the four sets of matrixelements are fitted seperately, so I will treat them seperately since each of them have their special quirks. Also the SOS has a special feature thet you can use matrixelements upto 6th order, but you can optimize only upto 5th order. For the HMS you can use and optimize only upto 4th order.

Theta and Phi

The individual matrix elements you want to fit are listed in fittheta.kumac and fitphi.kumac respectively. All the matrixelements under use must be listed under vary or fix (ie. ones being varied or ones being held fixed). The under lying method here is to map the events upto the slit and then find the nearest hole in the slit. From the position of the hole one can determine the angle $(\theta or \phi)$ and compare it with angle obtained using the starting matrixelements. Next they are tested to check if they are close enough, determined by the test *theta_test* or *phi_test*(assigned in target.db). These tests should nominally correspond to the radius of the sieve slit holes, but on the first pass they can be allowed to be larger to reflect the non optimized matrix elements. Next the events that pass the test are fitted using a linear least squares method to obtain a new optimized set of matrixelements. The sequence of commands is :

CMOP>*exec fittheta.kumac or fitphi.kumac*

CMOP>*fitter [Ntuple ID] [# of events] [element to fit(the or phi)]*

Note *: If any of these test parameters in target.db are changed while running one must remember to execute the macro refresh.kumac to read them in.*

For best results while fitting Phi, one has to merge the different slanted target runs into one ntuple and use it for fitting(always merge them in a series going either from +ymax to -ymax or vise a versa). This also means you have to update the relevant quantities like target inclination, spectrometer angle, target displacement, number of ntuples merged and the number of events in each of ntuples which were merged together in the file target.db . In the code these quantities are used to calulate the angle of the central hole (called Phizero) for each of the slanted target runs. Phizero always follows the sign of the target displacement (i.e. if target displacement is +ve phizero goes from +max to -max and vise a versa), hence one must choose the sign of target displacement carefully according to the order of the merged ntuples.

Note *: In case the target displacement was not uniform over the full slanted target scan, then the code needs to be hardwired to account for this nonuniform displacement. You must edit the subroutine fit_init_hms.f or fit_init_sos.f and use the example shown in these routines to make appropriate changes. You have to remake after all the changes are done.*

yTarget and Delta

These two have the special requirement of merging a number of ntuples from a slanted target scan or a delta scan. Farther in case of the delta scan one has to put cuts to include just the elastic or some well defined peak. It is important to note here that one has to be careful about the order in which the ntuples are merged and one must also record the number of events merged from each ntuple in input file target.db. The easiest way to do this is to start with an empty ntuple(obtained by running the engine with just one event) and then merging the required ntuples one by one, recording the number of events in each. An example can be seen in merge.kumac. In the code delta and y_targ follow the sign of delta step and the target displacement respectively (i.e. if delta step is positive delta goes from $+\delta$ max to the $-\delta$ max and vise a versa). So one must specify the sign of *delta* step and target displacement according to the the order in which the ntuples are merged. Some of the other quantities which need to be initialized are, the delta_peak, spectrometer angle, the inclination of the slanted target, number of ntuples merged and how many events were present in each of these ntuples. (all in the input file target.db). On executing the fitter command the *delta* or *y_tar* calculated from the starting matrixelements is compared with expected values and those events which are within the range specified by *delta_test* or *y_tar_test* are fitted using the least squares method. Again the sequence of commands is

CMOP>*exec fitdel.kumac or fitytar.kumac*

CMOP>*fitter [Ntuple ID] [# of events] [element to fit(del or y_t)]*

Note *: Max number of events that can be fitted is 50000. Once again if the delta scan steps or the target scan steps are not uniform the code needs to be hardwired as explained earlier.*

Saving the fruits of your labour

The command for writing out the optimized matrixelements is

CMOP>*ma/write/me [file name] [vary—fix—all]*

A genuine pain reliever

As you must have noticed by now the format of matrixelements.db is completely different from that of $hms/sos\_recon\_coeff.dat$ used by the engine. In the past going from one format to the other used to be a real pain in the neck and a major source of matrix mishaps (far from the matrix magic it was intended to perform). Well, being considerate human beings as we physicists are known to be (..I know, there are a few exceptions...), I did write a couple of conversion routines.

They are kept in $\sim ddutta/conversion$. The program convert takes $hms\_recon\_coeff.dat$ and converts it into the appropriate format and puts it in matrixelements.conv. While convertback takes matrixelements.new and puts it in $hms\_recon\_coeff.new$. Only thing thats needs to be done, before you start using $hms\_recon\_coeff.new$ in the engine, is to add the header, examples of which you can find in any self respecting $hms\_recon\_coeff.dat$.

Thats all for now, happy MOPing......

Note*: Abundant help is available on each of the commands, just as in Paw++*