Chapter 19.
Offline Analysis

Revised April 1994

19.1. Introduction

The goal of the offline group is to provide the g-2 collaboration with a set of tools for doing analysis, including easy access to the data and a common framework for developing managing and distributing analysis code. The offline code will not be as complicated as that required for the typical high energy physics experiment because of the limited number of physics goals and the relative simplicity of the cuts needed to form each dataset. g-2 is a minimum bias experiment in the truest sense. In order to conserve resources, however, it is imperative that as much of the offline package as possible be obtained from somewhere else, in particular from CERN. While not perfect, CERN code is coherent, well supported and runs on all the platforms needed by g-2: HP-UX, SGI IRIS, SUN SPARC DECstations and VAX/VMS. Furthermore, CERN software packages exist for nearly all our major software requirements - most of the offline work should involve the integration of this contributed code.

Listed below are the principal tasks of the offline group.

1 Set coding standards
2 Define data bank structure
3 Define analysis framework
4 Establish code management
5 Provide code distribution
6 Provide standard user interface
7 Set up database (geometric and calibration)
8 Provide histogramming and display capabillity: PAW/HBOOK/HPLOT/HIGZ etc.

The first item includes choosing the languages we will use and the coding standards we hope to enforce, selecting a databank manager for input and output, and setting up a code verification strategy. The analysis framework is a general means for integrating user code. With input and output standardized in terms of well-defined data banks (item 2) the user instructs the analysis manager to execute a set of routines on each data unit (presumably one spill in one calorimeter), each of which reads from standard input banks and writes to standard output banks.

The code manager should allow a community of users to develop code together, keep histories and restrict access. A related issue is code distribution, how to dessiminate this information to the far-flung users. Some of the code managers provide natural tools for distibution as well as an entire code development environment, including access to the editor, compiler and librarian. Of course someone must write the code itself, not only for final analyses but also a multitude of utility routines for developing calibrations and checking the basic health of the hardware, as well as simulation routines of various sorts.

g-2 will need a fairly modest database which should include both static information (where this is, how long that is, how many channels of those are there?) and time varying information like calibration constants and magnetic field maps. Finally there is the question of finding the correct hardware with which to perform all these tasks.

The formulation of the offline package is in its early stages. A basic framework will be established by the time data taking begins. By making wise choices at the design stages, we can minimize later difficulties. The following sections describe the steps that need to be taken.

19.2. Coding Standards

19.2.1. Programming guidelines

A set of basic standards must be established. For example:

1 Code should be written Fortran 77 (with the usual VaX extensions) or C.

2 IMPLICIT NONE is required. The IMPLICIT NONE statement is itself platform dependent but it must be implemented in some form, either in the code itself or in the compilation directive.

3 Parameters should be used whenever possible. These include physical constants such as the mass of the muon, the mean radius of the ring or the speed of light as well as offsets into data banks. The data bank specification should include a list of integer parameters which specify the offsets to various entries. Values of parameters should be stored in an include file or CMZ sequence.

4 Code should be commented. Particularly important is a statement of what the routine does and a specification of its input and output.

5 Use meaningful variable names but obey fortran typing conventions in doing so. Failing to observe the Fortran conventions requires the user to constantly refer back to the declaration headers. Describe the role of the variable in comments near its declaration.
6 No GOTOs. Use subroutines for repeated code. Indent DO/IF THEN bodies consistently and label endpoints of the longer ones.
7 Code should be device independent at least to the extent that it run on VMS and the standard UNIX workstations (DecStation, SPARC, HP, SGI). Device dependences arise mainly in file I/O and can be dealt with.
8 Limit the use of common blocks to control functions. Use of standard banks for input and output will make it easy to interface analysis subroutines. Any common blocks should be maintained by CMZ or some other automated code manager.

19.2.2. Code Verification

It will be the responsibility of the offline coordinator to ensure that production code meets the above standards. It is the job of the experts who write the code to ensure that it works. However, global code verification strategies will be developed to doublecheck the experts. One way is to exercise the analysis chain with simulated data and check to make sure that the answer is consistent with the input data. Successive releases of the offline code will be tested on the same sets of input data and any differences in the output histograms will be fully understood.

19.3. Data Banks and Bank Manager

As indicated above, the use of standard data banks facilitates input and output. However, Fortran 77's inability to allocate and deallocate memory for data storage at run time requires the help of a bank manager such as CERN's Zebra package. Using Fortran 90 would save us the trouble of having to deal with a data bank manager, but the code would be incompatible with many of the software packages of which we hope to take advantage and require an enormous education effort within the collaboration.

Zebra is the obvious choice for a bank manager but it is far from perfect. The bookkeeping overhead per bank is larger than much of the competition and the code which one ends up writing often ends up looking far from transparent. At the same time, it is CERN supported, it lies at the heart of HBOOK, CMZ and other CERN products, it does allow the user to transfer files between machines of different binary data format, and experiments in the HEP community *have* written utility routines to make it more user friendly.

Another possiblity is the use of TYPES, from the Supercomputer Computations Research Institute, which can be called from Fortran 77, runs on all the standard platforms and provides a much more effective, object-oriented solution to memory management than Zebra or YBOS(the CDF standard). In addition to built-in protection against memory overwrites, TYPES, through its data type definitions, provides encapsulation of the sort of linked list/pointer code which makes prospective Zebra users rightly shudder. While it runs on all the required platforms, TYPES, like Fortran 90 would require the offline group to do a considerable amount of public education.

Whatever the choice of bank manager, to encourage good coding practice, the offline group should provide template modules which demonstrate how to perform input and output in the g-2 analysis framework, as well as the good general programming style described above.

19.4. Data Banks

The standard I/O banks should be defined as quickly as possible in order that code now under development not have to be later transformed. The raw data banks should look much as the data comes from the hardware and the processed banks should contain information derived from the raw banks. For example, a raw bank might consist of all the WFD information and a set of processed banks derived from it might each contain (among other things) an electron energy and arrival time. The entries within each processed bank should be addressed with integer offsets. An include file of such parameters should be associated with each bank.

19.5. The analysis manager

The basic job of the analysis manager software package is to provide the user with the necessary hooks to interface a series of analysis modules. It should be possible for the user ability to include and remove modules in the execution path both at link and run time. User code provides entry points for the various stages in the analysis of an event or set of events (begin run,end run,define histos,event), which are called by the manager. The manager should also provide utilities for reading and writing (or NOT reading or NOT writing) groups of banks. Although basically intended for batch use, the analysis manager can also be useful in an interactive environment, where for example, the last routine in the execution path is some sort of graphical display.

We have not yet decided on a package for integrating the execution of the various modules. All the major HEP experiments have either written their own or borrowed from elsewehere and we have a wide range of choices. However we will avoid the use of PATCHY, which while powerful and flexible is stubbornly arcane and is porly documented. But again, whatever the choice of analysis manager, it should be promulgated within the collaboration as quickly and as vigorously as possible.

19.6. User Interface, Histogramming and Graphics

We should take KUIP as our user interface. KUIP runs on all kinds of machines, from dumb terminals to workstations running the Motif window manager. Its heierarchecal menu structure should be familiar to most physicists, as it provides the user interface to PAW. KUIP runs in both batch and interactive mode and can learn a set of instructions and save it as a macro. Menus are easy to write. The use of a menu-driven user interface with on-line help not only makes code more user-friendly, the standard presentation of choices allows skills acquired while mastering one package to be transferred to another.

We will use PAW/HBOOK/HPLOT/HIGZ package for histogramming, and basic data manipulation and presentation. There seems to be no serious competition.

19.7. The analysis code

Much of the code has already been written in some form or another, although none has been incorporated in the as yet unspecified analysis framework. Several analyses require minimal processing. The basic muon g-2 spectrum, the number of decay electrons observed vs. the time after injection, can be formed directly from the derandomized FSD channels of the MTDC, once the energy of the electron is obtained from the corresponding WFD data. Algorithms for forming the electron energy on line are now being written - presumably the same algorithms can be used offline for the pion injection data. Once the time histograms are filled, they can be fit with the help of G2MIN, a menu driven interface to Minuit, the standard CERN fitting package.

Other analyses concern themselves with the systematic corrections. Several programs designed to determine the distribution of muon equilibrium orbits (for the electric field correction) have already been written. The CERN III fast rotation code, which takes uses the evolution of the bunched structure of the beam at early times to extract the muon momentum distribution has been resuscitated and is now interfaced to the G2GEANT Monte Carlo. The code for a fourier transform analysis of the fast rotation has been distributed but has not been tested anywhere but Cornell.

The code for two other systematic analyses: the pitch and magnetic field corrections, is fairly mature. We have modified the CERN GEANE package to perform the electron traceback on simulated events and we find that it does a superb job. The same code will also be used to generate the acceptance correction required by the inevitable biases in r, z and E_e. As the chamber design is still incomplete, no code exists for finding tracks, rejecting background and solving other real-life experimental problems.

No code yet exists for the muon loss monitor. Several fairly simple studies have been performed and a few means for extracting muon signals from the calorimeters have been suggested but there is no convincing evidence that we will be able to monitor the muon population at the required level of 1 part in 10000 per lifetime. There is some code for the electric dipole moment analysis but it has yet to be incorporated into the standard analysis package.

The g-2 simulation still needs some work. Like all the analysis code, it does not conform to the standards listed above, although in addition to the standard Ntuple, it does produce data banks resemble the expected form of the MTDC and WFD banks. However, it must be modified to generate real g-2 data banks so that real and simulated data can be analyzed the same way.

There are a number of utility and production routines which need to be written before any of the the analysis code described above can be used. One obvious group consists of the routines which turn the raw data banks into electron time and energies. As this is the most critical part of the analysis, it will most likely have to be reworked many times. We also need routines for monitoring the basic health of the experiment. On line monitoring will do part of the job, but during commissioning we will undoubtedly want to dump a data tape and form a standard Ntuple (a PAW-readable direct access data file) and a large number of histograms to have a closer look.

19.8. Databases

The database requirements of g-2 are not extensive. As far as the offline is concerned, there are several general types of information that should be stored in a data base.

1 Static information: Basically the size and location of various things. This sort of information changes rarely, if ever. For example, the constants which describe our geometry should be stored in the database and should be read in by the simulation program as the defaults.

2 Calibration information: consists of data which is time dependent, although the frequency with which it is updated may vary. Presumably there will be a set of calibration constants for the detectors which will be written to the database every few runs. Raw data from the 300 fixed NMR probes, which can be read out on the order of seconds probably don't belong in the calibration database and should be written to tape. Summary data should go in the database, however. Data from the plunging probes and trolley, which come far less often, should definitely be in the database in some form.

In keeping with our general policy of using CERN software wherever possible we have decided to choose, on a tentative basis, HEPDB, a general High Energy Physics Database package, as our database manager. Although not yet widely tested, it is based on a database package designed by L3, whose requirements in terms of access time, flexibility and efficiency far exceed those of g-2. Like PAW, CMZ and other CERN packages HEPDB is based on the Zebra memory management system and will run on all the required platforms. HEPDB records can easily be transferred from machines with different binary represenations through the use of FZ sequential exchange mode files.

The more pressing issue is finding out what are the database needs of the various subsystems and how they should be implemented. We expect to gather this information by querying the relevant team leaders this spring and then to specify the structure of each subdatabase as part of an offline workshop planned for the summer of 1994.

19.9. Code Management and Distribution

Code management is concerned with the storage of code at a central repository, restricting access to the code for corrections and updates, keeping track of version numbers,maintaining code histories and releasing consistent code packages. Code distribution, on the other hand, is concerned with making sure that all member institutions have identical versions of the code, apart from possible machine dependencies. Code distribution should be far simpler than in the typical collider experiment. No one, for example, is familiar with every CDF production routine, but several of us on g-2 will probably know virtually every line of our own offline code.

CMZ is the logical choice as the code manager. It runs on all the necessary platforms and can perform all the necessary tasks. Some people object to CMZ, as it is a descendant of PATCHY and uses some PATCHY terminology. Nonetheless, it is easy to use, unlike PATCHY. The casual user need only learn a half dozen commands and online HELP is those who need it. Moreover, it includes many useful features such as a formatting tool for indenting loops, a utility which hunts for undefined variables and even a make facility for updating libraries. It also has direct interfaces to the C and Fortran compilers, the linker and various editors. It has been used extensively by HEP collaborations around the world, including our own collaborators at the Budker institute in Novisibirsk as well as H1 at DESY.

There is an appendix in the CMZ manual which explains how H1 has used CMZ as a code manager. We intend to use the H1 example as a guide for maintaining the G2GEANT simulation over the next few months and to tailor the framework they established to suit our own needs. This should involve little extra work, as the simulation is already maintained via CMZ. However, until recently, we have not had to face the problems created when multiple users try to work on one body of code. Our experience with the simulation should provide a good test of CMZ as a code development environment and help us work out the sorts of issues which will arise when we try to manage a much larger software package.

19.10. Hardware Requirements

The hardware requirements for the g-2 analysis are fairly simple, if demanding. As far as processing data tapes goes, the CPU requirements should not be great. Our online group has estimated that it will take on the order of 20 μsec to process a given pulse on a 68040 (25 MHz) CPU. This may be an optimistic estimate but even if it is off by an order of magnitude the time required to process 30 billion events is still only 6 hundred thousand seconds, or about 8 days. On a fast RISC processor this figure would be reduced by an order of magnitude. We might do well to wait until 1996 to purchase the processing power we'll require for the 1997 data.

Data production will probably be I/O bound. In that case, the time required to process all the data will be comparable to the time required to write it. Naturally, we can speed up the analysis by buying more exabyte drives and enlisting idle DAQ drives during down times. Extra drives will also be required to make copies of the 800 odd data tapes.

In addition to the production hardware, which would likely consist of a few big disks, a few fast processors and as many exabyte drives as we can muster, it is important that a disk and tape drive always be available for diagnostic analysis. Online monitoring should tell us a great deal about the basic health of the system but we will undoubtedly want immediate access to full data samples, particularly during commissioning.

Simulation remains a matter of some concern. In most high energy physics experiments, it is possible to simulate as many events (or more) as one has in the final data sample. In a minimum bias experiment like g-2, this is simply impossible. On a top of the line alpha, we estimate that it would take our full simulation (run with minimal detail) on the order of 100 years to generate 30 billion events. And yet, if there for unknown systematic effects which change ω_a by roughly 1ppm we may need to generate data sets of this size. One can search for effects analytically or exaggerate them and try to extrapolate from results drawn from smaller data sets, but when these more clever alternatives fail, we must turn to our fast Monte Carlo. If we demand that the fast Monte Carlo produce a data set in one day of CPU time, that's an approximate rate of 350 KHz or 1000 clock cycles/event on the fastest Alpha, a very demanding specification. We will do our utmost to make sure that this sort of supercomputing is not necessary.

19.11. Appendix: A more detailed summary of the simulation

19.11.1. Introduction

G2GEANT is Geant based Monte Carlo of g-2 storage ring. Elsewhere, we have outlined its functionality. Here, we give a little more detail.

19.11.2. Geometry

When the ring is fully simulated, the vacuum tank geometry is specified by BNL drawing D19-M-3443. This drawing is now fairly old, but remains adequate for our purposes. The other elements, the calorimeters, wire chambers, FSDs, Lightguides,PSDs, electrodes, etc., are described in terms of GEANT volumes and are then set in the storage ring. Here is the status of the more important elements:

1 Electrodes: The position and geometry of the electrodes are up-to-date as of 3/94. The support pieces are largely in place,

2 Collimators: The position and geometry of the collimators is up to date as of 3/94. There are 3 sets in each of the electrode sectors which are not just downstream of the inflector,

3 Yoke: Not modeled

4 Pole tips: Modeled crudely

5 Inflector: Not modeled

6 Calorimeter: Updated in 10/93 to reflect new Illinois design

7 SSD and FSD: Ditto

8 Lightguides: Reflect 10/93 design. FSD guides are reasonable approximations of true design. No SSD lightguides

9 PSD: No current design available. Model reflects status in 1991.

10 Bellows sections: Not modeled

11 Wire Chambers: A detailed straw tube design from P. Debevec was introduced in 2/94.

Design parameters are adjustable in most cases. Detailed geometries (like the calorimeter) can be replaced by approximations

19.11.3. Input

S. Mane has provided the detector group with the initial and final (typically at 22.5 microseconds) coordinates of large groups of stored muons. In either case, G2GEANT tracks the muons until their point of decay, using a parametrization of the muons' betatron motion. The decay electron is then handed over to GEANT itself for tracking and shower simulation. It is also possible to input muons or pions (also provided by S. Mane) which are useful for studies of the flash, the muon loss monitor or some of the muon phase space monitoring devices.

19.12. Magnetic field

The magnetic field is described in a giant 2 dimensional matrix, with indices corresponding to r and z, which contains the value of B_z at 1661 lattice points, 151 in r by 11 in z. It was obtained from a Poisson simulation of the magnet and will be updated with actual measurements when they become available. A pseudo-electric field option is also available. Although GEANT supports only magnetic field tracking, the electric field of the quadrupoles can be reasonably mimicked by an equivalent electric field, since the force on the muons and decay electrons is largely perpindicular to their motion.

19.13. Output

G2GEANT output is of two types, HBOOK and g-2. HBOOK output includes a standard Ntuple which is collected by G2GEANT and user defined histograms and Ntuples. g-2 style output consists of data banks which bear a strong resemblance to the raw data banks from the MTDC and waveform digitizer.

19.14. Appendix: A brief survey of the offline code

19.14.1. Traceback with GEANE

One of the issues left unaddressed in our first studies of the electron traceback was how to do the traceback in a non-uniform field. Because the traceback routine we used was fairly unsophisticated, we always kept the magnetic field uniform in our studies. Naively, one might expect that since one can use GEANT to track positrons forward in time, one should be able to use GEANT to trace them back. Once again, naive expectations are correct. A group at CERN has put together a package called GEANE which (with some slight modifications) will not only do the traceback, but calculate the errors involved as well. We have written a short program which employs the package and performs the traceback in a general non-uniform field.

The basic physics and geometry of GEANE comes from GEANT. For our purposes, the principal difference in operation is that because the particles travel backward in time, their charges and momenta must be reversed, and instead of losing energy through ionization and other physical processes, they gain energy. Furthermore, because we are also interested in the random errors introduced by, for example, multiple scattering, an error matrix is associated with each track, and is propagated with the track as it curves in the magnetic field and passes from one medium to the next.

The traceback is done in two steps. In the first, the positron is traced backwards through the three wire chambers. GEANE is given the position of the hits in the chambers and a trial momentum provided by the simple traceback, where the field is assumed constant. It then calculates a χ^2 for the trajectory and calculates a new starting momentum based on the track residuals. This process is repeated (3 iterations is generally sufficient) until the results converge. At that point, the track is propagated outward until its radial component of motion vanishes.

19.14.2. FASROF - The CERN III fast rotation code

19.14.3. Introduction

The fast rotation analysis, which seeks to measure the distribution of muon momenta by examining the beam debunching early in the spill, is based directly on copies of the original CERN III Fortran 66 code and includes two separate packages: BOBAN and REVFR. BOBAN looks at the pattern of hits before overlap and calculates the mean momentum of the bunch and the injection time. REVFR looks at the spreading of the bunch over time and calculates the equilibrium orbit distribution or, equivalently, the muon momentum distribution. Both have been rewritten in Fortran-77 for the g-2 experiment but are substantially unchanged. For complete details please consult Peter Hattersley's (and others') original notes on the two programs.

FASROF accepts as input HBOOK histograms of electron arrival times. These histograms can be made from the standard G2GEANT ntuple with a standard kuip macro. The output of both packages is very simple. REVFR prints out the radial distribution at each of its 5 iterations and then plots the last one. It also prints out the average field and the mean and rms of the radial distribution. BOBAN prints out its best guess as to the injection time and rotation frequency and plots the arrival time of the bunch and its width vs. time.

A mentioned earlier, Y. Orlov has written a program which attempts to measure the muon phase space distribution using Fourier analysis. It will soon be integrated in the offline package.

19.14.4. G2MIN - A g-2 interface to MINUIT

It will be some time before we will be spending a lot of time fitting g-2 decay spectra. However, for the purposes of evaluating the effect of pileup and other easily modeled distortions of the canonical spectrum, it's handy to have a program which will do such fits. Naturally, g2min does no minimization itself. Rather, it is an interface to Minuit (the CERN minimization package, also used by CERN III) which can be modified to suit individual needs.

The code performs a six parameter fit. Five are the traditional N, τ, ω, A and ϕ. The sixth is an overlap parameter, call it α, which modifies the spectrum as follows. Instead of the traditional form:

$$N_0\left(t\right) = Ne^{\frac{-t}{\tau}}\left(1 + A\cos\left(\omega t + \phi\right)\right) \tag{19.14.1}$$

we have

$$N\left(t\right) = \frac{N_0\left(t\right)}{1+\alpha N_0\left(t\right)} \tag{19.14.2}$$

Fitting to other functional forms is easily done.

G2MIN's input is a formatted data file with g-2 spectrum data. The format is 10(1x,I11) but can be adjusted. Running the code requires the following steps. **READ** the data. Set the starting parameters with **PARAM**. Use **COMP** to eyeball the result. MINUIT's **STEP** sizes and BOUNDs can then be set. **LIST** each to see if they need adjustment. The time interval over which to do the fit can be set with **INTER(VAL)**. All numbers are single precision but are translated into double precision for MINUIT. To do the fit, use **CHISQ** which turns control over to MINUIT and type MINI. When MINUIT is done, it will return control to g2min and print out the final fit values and errors.

The control knobs are as follows

1 READ_DATA: Read in a formatted data file which contains the g-2 spectrum. The bin width, which determines the time, should be set first.
2 COMPARE: Plot the fitted spectrum on the workstation screen
3 INTERVAL: set the period in microseconds over which to do the fit
4 LIST_DATA: print the first microsecond of data.
5 PARAMETERS: Set initial parameters of fit
6 LIST_PARAM: Print out initial parameters of fit
7 STEP_SIZE: Set initial step size for Minuit search through parameter space
8 LIST_STEP: List initial step sizes
9 BOUNDS: Set bounds on parameters
10 LIST_B: List parameter bounds
11 CHISQ: Do the fit
12 EXIT: Exit the program

5 ns is the default bin width. Other defaults are reasonable and need not be changed, with the exception of N which depends on the number of events. Bounds and step sizes are automatically scaled when guesses are changed but can be reset with BOUNDS and STEP. CHISQ turns control over to MINUIT. Simply type MINI at the prompt and MINUIT will search out the minimum, spewing out possibly useful information as it goes. If it finds a minimum, it returns control to the user menu, and g2min prints out the location of the minimum and the statistical errors on each parameter.

19.14.5. Other programs

One other piece of code now integrated with the simulation is E. Hazen's simulation of data flow in the MTDC. The rate of data flow is critical to the MTDC design for its front end buffers are small and must be flushed often early in the spill if data is not to be lost. Eric's code runs on the MTDC data buffers produced by G2GEANT.