AN OBJECT ORIENTED APPROACH TO PROCESSING ACCELERATOR ALIGNMENT MEASUREMENTS

M. Jones, CERN, Geneva, Switzerland

Abstract

Over a long period of time the least squares compensation program at CERN, LGC, has been gradually re-written in C++. Part of the program has been integrated into a software library, which is also used in a number of other programs, with the functionality specific to LGC built on top. The geodetic transformations used within the program have been updated and a sparse matrix representation added for the least squares matrix, this latter change contributing, in part, to a significant increase in the calculation speed.

A new version is now nearing completion, with a necessary change to the input data format allowing for the inclusion of measurements from a non-levelled theodolite or laser tracker. The observation equations used within the program have also been revised where appropriate, to increase the programs flexibility, and these new formulations have added unknown lines and planes to the parameters to be determined. Some of these new observation equations will be presented, together with the anticipated advantages they provide.

INTRODUCTION

The LGC program was originally written in Fortran77 in the 1980's, at the time of the construction of LEP. It was designed to handle all the different measurement types that were used by the Survey Group at CERN, and to process the data in a 3D coordinate system either locally defined, or, more rigorously, taking into account the horizontal and vertical geodetic reference surfaces that were established during the same period for the CERN site. The program passed through the hands of numerous "stagiaires" (trainee surveyors) over the next 10-15 years and various new observations and analysis tools were introduced.

In view of the tendency at CERN, and elsewhere, to migrate from software written in F77 to other more modern languages, and taking into account the anticipated requirement to add further functionality, analysis tools and observation types, the last 10 years have been spent, when time allowed, translating and developing this program in C++. The program went through several revisions, and again passed through the hands of numerous students and fellows, but the translation is now complete.

In fact we have now found time to start on new developments in the program, inspired in part by other software which had been developed by the experiment collaborations, such as Simulgeo [1] and ARAMyS [2], and also programs such as LTOP (Swiss Federal Office of Topography) and TRINET+ (heig-vd) which benefit from a more detailed stochastic model than that found in LGC.

With an initial request to process measurements from laser trackers, or off level total stations, a proposal to change the data file format was picked up again, and changes were made to simultaneously allow an increased flexibility in the presentation of certain measurements. This then led inevitably to the introduction of some simple 3D shapes, and the requirement to determine their parameters, in addition to the set of unknown point coordinates. Another new version of LGC is now nearing completion, and the door has been opened towards further new developments.

CHANGES IN THE C++ VERSION

The primary goal in re-writing LGC was to make it easier to modify and add to the program with future developments. An analysis of the FORTRAN version of the program using Logiscope had confirmed that the majority of the routines were relatively easy to understand, and could therefore be modified and tested efficiently. Unfortunately one routine that was flagged as needing to be changed was the MAIN routine. This routine included many jump statements into and out of loops and at 3000 lines represented nearly a third of the code, see Fig. 1. The author, having previously been one of the trainee surveyors whose hands this program had passed through, also had first hand knowledge of the difficulty of assuring that no unexpected results were introduced by a seemingly innocuous change in the code.

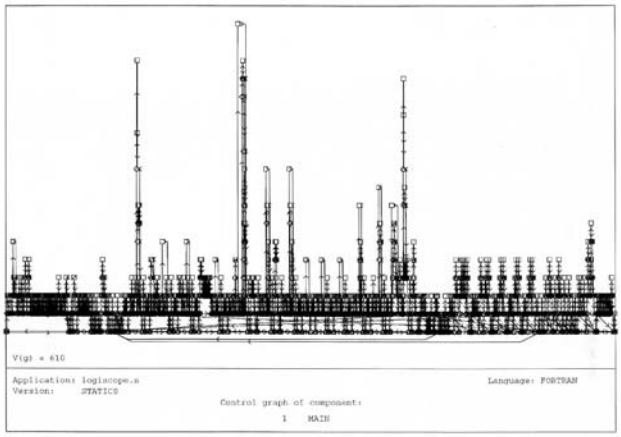

Figure 1: Control Graph of the LGC F77 Main routine

A lot of the new code for the LHC era at CERN was being written in C++, and as the future of FORTRAN was uncertain it was decided to re-write LGC in C++.

Reference Frames and Transformations

Whilst re-creating a CERN reference frame transformation program CSGEO, it had been shown that

the transformation algorithms developed for the LEP era were not as rigorous as they could be and could lead to small errors. Although not significant for the day to day survey and alignment work it was decided to update the transformation algorithms, and extend the list of reference frames available, in order to maintain the highest possible precision. This required the introduction of local astronomical reference frames into the LGC processing algorithm. The change to C++, and object oriented code, provided certain advantages in the representation of objects such as points and reference frames, and this was now fully exploited.

For the purposes of the transformation algorithm the reference frames are represented as nodes of a graph and the transformations as links between the nodes. The sequence of transformations to pass between any pair of reference frames is determined at run time by simply finding the shortest path between them on the graph [3]. The spatial position of a given point is then changed as it is transformed from one reference frame to another using the "State" design pattern [4]. This same pattern is used in the transformation of the coordinates of a position vector when it is changed from one coordinate system to another.

A selection of these reference frames and transformations were also used in a number of different applications, including Geode - the SURVEY database GUI application - and some field software. To avoid duplication of code, this whole section of code was extracted from LGC and now forms the basis of a software library called SurveyLib.

Variables and Objects

Although 3D objects were not included in the initial translation of the program, they were already foreseen and offset measurements were identified as being the distance to a line or a plane. The observation equations however remained mostly unchanged.

The program itself went through two or three major revisions with large parts of the code being completely restructured as our knowledge of object-oriented programming improved and as we identified the weak points in our new code. As the completion of the re-write approached, some more minor changes were made which had a direct impact on improving the software performance. (This shows the advantage of having computer programming students to work on the software, despite the disadvantage of having to teach them the fundamentals of surveying.)

Hash tables have been included throughout the application, dramatically increasing the execution speed by means of more rapid searches through the data sets. Sparse matrices have also been included for the least squares matrices.

Initial tests were made using existing sparse matrix libraries (such as TAUCS), but as we also looked towards including quadruple precision (see below) it was decided to write our own implementation. The sparse matrix is based around a compressed column storage algorithm which reduces the memory storage requirements and provides faster operations; see an example in Fig. 2. Matrix multiplication has been implemented, with special cases where diagonal matrices are involved or only the lower triangular part of the result matrix is required, e.g. for symmetric matrices. A special algorithm has also been implemented for the simultaneous multiplication of three matrices. For the least squares computations Cholesky and LDLT decompositions and solving methods are available.

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}$$

Value	10	3	3	9	7	8	4	8	8...9	2	3	13	-1
Row Index	1	2	4	2	3	5	6	3	4...5	6	2	5	6

Column Pointer	1	4	8	10	13	17	20

Figure 2: Example of Compressed Column Storage

At this stage quite a significant number of changes had been made to the program. Versions of the C++ application had been produced along the way as we moved towards implementing the full functionality, and more, of the FORTRAN program. Testing had been carried out at each stage principally by running a number of test files through previous versions of the software and the newly baptised version. This initially posed a problem since the transformation algorithm, a fundamental part of the data processing, had been changed between the FORTRAN version and the C++ version [5]. Small discrepancies existed between the output results, and considerable time was spent analysing the differences case by case to show that it was only due to this change in methodology and not due to any errors in the new algorithms themselves.

With a new representation for the least squares matrices, and all the other changes, small discrepancies between the intermediate results from least squares matrix manipulations between the new and the previous versions were once again evident, even though the differences in the results from the two computations were negligible. Whilst we could not rule out an error in the new code, passing through the code and testing did not reveal anything. We began to wonder of there wasn't another possibility. With the extended output precision of the software, the coordinates of points across the CERN site now ranged up to 10 km with a precision of 0.1 micrometres or 12 significant figures. This could start to approach the limits of the precision maintainable with double precision real numbers in calculation intensive software, which least squares algorithms processing of large data sets certainly are! It was possible that the change of matrix representation and associated algorithms led us to a different solution due to a different set of rounding errors.

Quadruple precision was the obvious answer, but a search of the internet revealed that this was only really a

possibility using software emulation. Ironically quad precision was already a reality in some FORTRAN compilers. One of the best options was an unpublished implementation in the Intel compiler. Fortunately CERN had access to this compiler, so the modifications were made to the code to enable us to switch between double precision and quadruple precision at compile time. This change probably slows the processing time by a factor of 2-4 times but this was outweighed by the gains provided by the other changes described above. In the end this did not provide any significant improvement in the discrepancies between the results, but as already mentioned the differences between the final results from the computations were negligible. In any case we shouldn't need to worry about calculation precision for some time!

Application Output Data Formats

A number of changes have also been made to the format of the data output from LGC. The first change was implemented to improve the flexibility in the control of the precision of the output results data.

A number of keywords existed in the FORTRAN version to create output for specific applications such as the early CLIC studies. However these did not always produce a consistent level of precision throughout the entire output data file. A new keyword has now been implemented to allow the results to be output with a precision which can be set to lie anywhere between metres to tenths of a micrometre. This is sufficient for current projects and installations, but can be easily extended in the future if required.

Typically the output files are presented as text files with the data in a tabulated format. This can now also be changed to give comma separated variable (CSV) format, for easy import into Excel for example.

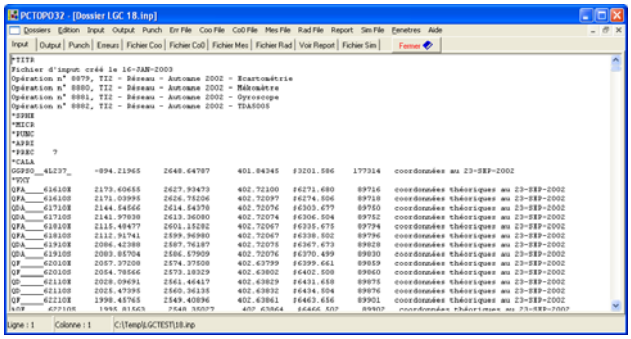

Figure 3: PCTOPO interface

The current user interface for LGC is provided either by Geode, or a utility providing access to a number of our applications, including LGC, called PCTOPO, see Fig. 3. At this time a student project was also launched to create a GUI front end for LGC with an emphasis on the graphical presentation of the network of measurements for a given field operation, see Fig. 4. The work was never finalised, but has provided a basis for future work in this direction, and has shown the areas where more work is still required, such as in the presentation of the long narrow measurement networks found in accelerator tunnels.

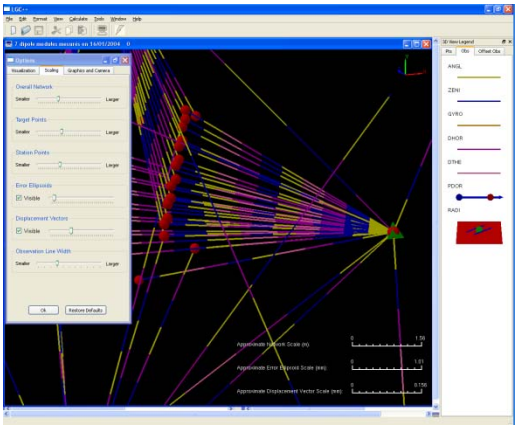

Figure 4: LGC GUI 3D Network View

NEW FILE FORMAT

With a new C++ version of LGC now available with an implementation of the full functionality available from the original FORTRAN code, plus some new functionality and the bugs in the original code fixed, it was time to look towards extending the program further.

A list of areas where LGC could be extended had existed from the time when the project to re-write it first began. A review of this list identified the introduction of LTD measurements (or any equivalent instrument such as a non-levelled theodolite) as the first priority. Work had already begun on the insertion of LTD measurements into the SURVEY database, and the obvious next step was to be able to export them to an LGC input file where they could be processed simultaneously with any other measurements.

The input of the LTD data was obviously going to require a modification in the input data format for those measurements. Keeping each different type of measurement separate, e.g. horizontal angle, zenith distance, and spatial distance, required a strong link to be created between the corresponding measurements. Rather than taking the current input format, and forcing a solution for the LTD measurements to conform to it as best as possible, it was decided to rework the whole input file format. In reality this meant the measurement data input format, in the new input file format the calculation options and the point data input remain largely unchanged.

New sections have now been added to declare instruments and targets, and these sections allow for a much more detailed stochastic model such as those found in other applications. Standard errors for instrument, or target, height or centring can now be introduced, and also for other values specific to a given measurement. As before the defaults values are applied throughout the file if no specific values are specified for a given round of measurements or individual measurement.

Measurements made from a given instrument station are now grouped together. For a total station, theodolite,

or the laser tracker the measurements are then grouped by a given horizontal orientation, or V0, and then by round of measurements. This allows for a much more direct control of links between the measurements that were made at the same time. It is expected that this will lead to a better appreciation of the correlations between related measurements during the analysis of the results.

REVISED OBSERVATION EQUATIONS

The changes to the input format of the LGC data file also allowed for more far reaching changes to a number of measurements, principally the offset measurements, and the introduction of 3D objects, other than spatial points, into the list of unknowns to be determined.

If we consider the example of the horizontal offsets, that are used extensively in most accelerator planimetric survey and alignment operations at CERN. As the data is presented, these measurements have always been characterised by two spatial points (called anchor points), a measured point, and the measured offset distance (the horizontal distance between the measured point and the straight line joining the two anchor points), see Fig. 5. All three of these points must be included in the calculation.

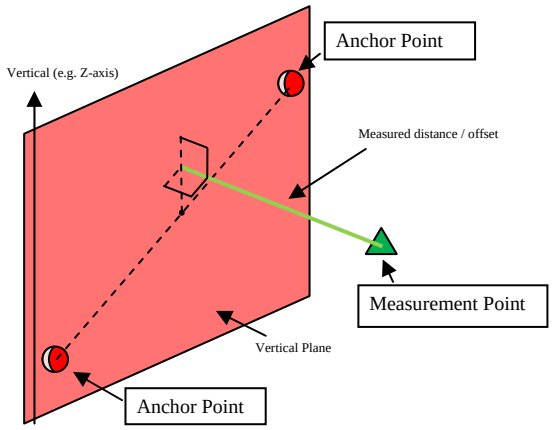

Figure 5: Horizontal Offset Measurement Defined using Points

In reality, for horizontal offsets, the instrument is actually placed on a point and the distance to a vertical plane, characterised by a stretched wire, is measured. It is also becoming more frequent for the wire to be stretched between to two tripods whose position is not important. In this case the offset from the two anchor points is also measured. Prior to the inclusion of these measurements in an LGC data file, each measurement linked to a given stretched wire installation are reduced to the straight line passing through the two anchor points.

The problem with this pre-processing is that if there is an error in the measurement to one of the anchor points, all the reduced measurements relying on that anchor point measurement will be wrong. It is then necessary to return to the original measurements and identify a different anchor point.

A way to avoid the problem is to introduce a vertical plane as an unknown object, see the simplified Figure 6. The measurements to this plane from the points are all processed in the least squares adjustment, and the parameters of the plane are included in the list of unknown parameters. There will obviously be a change in the observation equations depending on which representation is chosen.

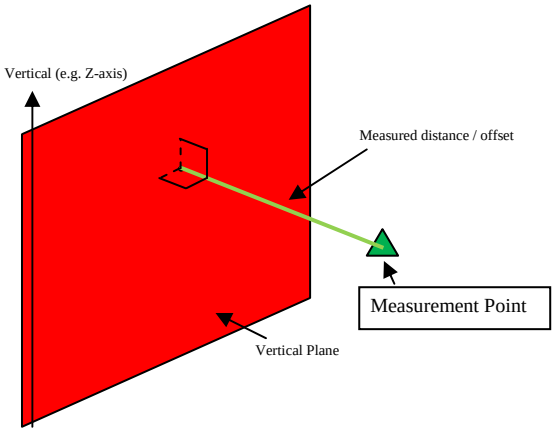

Figure 6: Horizontal Offset to a Vertical Plane

In vector form the observation equation for the offset measurement defined in terms of three points is given in Eq. 1.

$$S^2 = (X_M - X_{A1}) \cdot (X_M - X_{A1}) - \left(\frac{(X_M - X_{A1}) \cdot (X_{A2} - X_{A1})}{|X_{A2} - X_{A1}|} \right)^2 \quad (1)$$

where,

S	= offset distance
X_M	= 2D coordinates of the measurement point
X_{A1}, X_{A2}	= 2D coordinates of the anchor points

The observation equation for the offset to a vertical plane (see Eq. 2) is much simpler, has the advantage of using the actual field measurements, and does not require and any specific pre-processing of the measurements. This means that measurements can be removed from the calculation without any problem. The disadvantage is that it does increase the number of unknowns in the adjustment. With the computing power available from desktop computers today, the increase in the number of unknowns should not be a major drawback.

$$S = X_M \cdot \hat{n} + d \quad (2)$$

where,

S	= offset distance
X_M	= coordinates of the measurement point
$\hat{n}$	= horizontal unit vector normal to the plane
d	= the plane distance parameter

Since the plane's distance parameter in Equation 2 is the distance to the origin, it is also better to introduce a fixed point much closer to the area being measured (see

Eq. 3). This means that a small rotation of the plane should not induce significant changes in this parameter and will help the convergence of the least squares process.

$$S = (X_M - X_0) \cdot \hat{n} + d_0 \quad (3)$$

where in addition,

X_0 = coordinates of an arbitrary fixed point
d_0 = the plane distance parameter, from X_0

Another less obvious consequence is the requirement to add constraints to the least squares adjustment, such as a constraint to ensure that the plane vector remains a unit vector. Again, this is not a major problem from a computational power point of view, but does complicate the least squares formulation.

This same philosophy has been applied to the offset to a spatial line and the offset to a vertical line measurements. Intuitively in each case a line in 3D space has been introduced as an unknown.

Further work is still needed to refine the processing of these new observation equations. The solutions are currently less stable than would have been expected, and require further investigation to understand these instabilities and why they were less apparent with the original observation equations. Further work is also need to refine the decomposition and least squares solution methods for these more complicated formulations.

CONCLUSIONS

Following many years of intermittent development a new version of LGC written in C++ has now be produced which includes all of the functionality included in the original F77. This new version also includes several improvements such as: a revised algorithm for the processing the observations which is base around a more rigorous process for transforming the point coordinates between the necessary reference frames; and a sparse matrix model exploited in the least squares solution.

This re-writing of LGC has also resulted in the creation of a software library which we have called SurveyLib, and which is also now the basis for a couple of other new in-house applications. We are also hopeful that the structure of this new application will enable us to introduce new concepts and developments into the code much more easily than in the F77 program.

Further developments have now been rapidly applied to the new application in order to permit the integration of LTD measurements into an adjustment. This has pushed us towards a change in the format of the input file, and encouraged us to simultaneously make a change in a number of observation equations. This in turn led to the introduction of a couple of simple 3D objects whose parameters are included in the list of unknown parameters to be determined. These changes have required us to evolve the algorithm of the least squares solution. When time and resources are available we hope to conclude on the best way to manage these new concepts, and finalise this latest version too.

When we began this project the choice of programming language was one of the first questions that were addressed. The choice of C++ was made primarily taking into account: the current and future support available from the CERN IT Department; and the ability of the finished application to process large data sets and the corresponding least squares matrices as fast as possible.

Ten years ago it was difficult to find surveyors who had experience in writing C++ (or even other object-oriented languages such as Java). For those with an interest in programming, and a mathematical background, training courses were periodically available at CERN, but not always when they were needed. This then meant taking time to teach them the language within the survey group. In either case the surveyor would not be productive for maybe six months, and this was only worthwhile if they would be at CERN for more than a year. More recently we have been working with computer programming students with a strong mathematical background, and found teaching them the fundamentals of surveying or least squares a more productive process. Finally we are now seeing some surveying students with a good knowledge of C++.

There have been times when the difficulties of finding people to work on the project, for the reasons outlined above, have put in doubt the initial choice of C++ as the programming language. Visual Basic seems to be the language that most surveyors know, but it would be difficult to imagine such a processing intensive application written in this language. A similar problem exists with Java, which may have fewer hidden dangers than C++, but is still perhaps 2 – 10 times slower. C has less functionality, and FORTRAN now has very little support at CERN. Taking all these factors into account the author still believes that C++ was the correct choice. The appearance of surveyors with programming skills in this language would appear to vindicate that.

This development has only been possible with the contribution of numerous people (students, fellows and staff) and the author would like to thank them all for their help in reaching the final goal. It was not always easy and we all had new things to learn along the way.

REFERENCES

[1] L. Brunel, “SIMULGEO: Simulation and reconstruction software for optogeometrical systems”, CERN-CMS-NOTE-1998-079, CERN

[2] C. Amelung, “ARAMyS Manual” Atlas Muon Note, Draft, 2006; http://amelung.web.cern.ch/amelung/aramys-manual.pdf

[3] U. Breymann, “Designing Components with the C++ STL”, Harlow: Addison-Wesley, 1998.

[4] E. Gamma et al., “Design Patterns: Elements of Reusable Object-Oriented Software”, Boston, MA: Addison-Wesley, 1995.

[5] E. Claret, “Ecarts de Coordonnées entre LGC et LGC++ pour les Calculs Géodésiques”, Activity Report, EST-SU/ACG, CERN, 2003 : EDMS Doc No. 427072.