Exploratory Orbit Analysis

LEO MICHELOTTI
Fermilab*, P.O.Box 500, Batavia, IL 60510

I don't think anybody could be better acquainted with the thickness of his own head than myself. ... I do, notwithstanding, feel as if I could express my feelings in a most remarkable manner, if — if — I could only get a start.

— Charles Dickens
Dombey and Son

Unlike the other documents in these proceedings, this paper is neither a scientific nor a technical report. It is, rather, a short personal essay which attempts to describe an Exploratory Orbit Analysis (EOA) environment — a thing which does not exist, which could exist, which probably should exist, but which never will exist without a sustained, coherent effort combining the skills of many people over many years and without the prerequisite understanding that time and money spent on developing tools is invested, not squandered.

Let us begin with the observation that creating specialized computing environments to address specific classes of problems is hardly a new idea: CAD systems are basically nothing more than this; in the medical profession, environments now exist in which doctors can quickly and easily explore data from PET, CAT, or NMR scans in a variety of ways, such as enhancing structures of interest, slicing images along arbitrary planes, or highlighting diseased tissue; and, of course, the graphics-driven, analysis environment attached to the detectors of modern high energy physics experiments is an example familiar to us all. The important point is this: no one expects an engineer to be an expert on how the procedures of his (or her) CAD system actually operate, a doctor to understand Radon transforms, or every high energy physicist who uses CDF to provide his own graphics and statistical packages. We expect these people to be experts in their own specialties, and we expect others to provide the appropriate "environment" in which they can carry out tasks specific to those specialties quickly and efficiently.

In contrast to this, accelerator physicists need to understand Hamilton-Jacobi theory, Lie groups, Fokker-Planck equations, computer science, Lyapunov stability theory, global resonance analysis, and a host of other things in order to proceed, individually and piecemeal, with their calculations. The activity of developing reusable toolkits, by which the expertise of one person can be shared by another, is still not given enough emphasis relative to that of quickly writing non-reusable programs for doing individual calculations of a (perceived) high priority. Tool development is too often considered a secondary activity, to be done on the side while tackling the "real" problems.

Analyzing the behavior of a four or six dimensional nonlinear dynamical system is at least as difficult as analyzing events in high-energy collisions; the consequences of doing it badly, or slowly, would be at least as devastating; and yet the level of effort and expenditure invested in the latter, the very attention paid to it by physicists at large, must be two orders of magnitude greater than that given to the former.

It is difficult to choose the model which best explains the behavior of a physical device if one does not first understand the behavior of the available models. The time is ripe for the development of a functioning EOA environment, which I will unsuccessfully try to describe below, to help us achieve this goal.

Before beginning, let me apologize for not spending enough time learning the accomplishments of others in this area sufficiently to comment on them here. I hope to learn much during the course of this Conference, and a more serious paper is planned for the future. In the meantime let me cite the work of Schachinger, Talman, and their coworkers in building a unified schema for doing simulations, performing operations, and analyzing data on the SSC [18] and, of course, the continuing accomplishments of Keil, Iselin, and company in expanding, upgrading, and maintaining MAD [6,8]. Perhaps what I am trying to do is already contained in the work of these people, or others, but on the chance that there yet remain one or two complementary points to be made, let us continue with the essay.

Exploratory Data Analysis As a starting point for designing an EOA environment we might look at what already has been accomplished in the field of statistics. Exploratory Data Analysis (EDA) is, by now, a well-established field which comprises a collection of tools for manipulating multivariate data with the purpose of revealing their structure. It should be obvious that choosing a good way to organize and display a body of data can be enormously helpful in mining its information content and, conversely, that choosing a bad one can either bury it further or, even worse, mislead one to the wrong conclusions.[1] EDA provides a battery of tools to aid in the pursuit of this goal (that is, mining the information content, and all that). It usually proceeds under the assumption that the analyst knows nothing about the data's underlying "dynamics." Exploration accordingly progresses in a model independent manner by employing clustering, polynomial regression, curve fitting, histogramming, analysis of variance, two-way correlation tests, transformations, and various other general purpose, statistical techniques.[24,4,7] The analyst is not expected to understand how these tools work; they are available for his *rapid and efficient* use. Structural clues revealed by these manipulations can then influence the development of models in the "confirmatory" (as opposed to "exploratory") stage of the analysis.

In a similar way, we need an environment in which accelerator physicists who want to explore the behavior of particle orbits can *quickly and easily* employ a variety of tools from perturbation theory, Lyapunov stability analysis, or whatever without becoming experts on them. An Exploratory Orbit Analysis (EOA) environment can be designed in analogy to EDA environments so that physicists can explore dynamics, form hypotheses, and test them.

Language A programming language is more than the medium for communicating instructions to a machine; it is also a medium in which people formulate solutions to problems. A good language will aid the performance of *both* tasks, not just the first. "Ideally one approaches the task of designing a program in three stages: first gain a clear understanding of the problem, then identify the key concepts involved in a solution, and finally express that solution in a program. However, [in reality] the details of the problem and concepts of the solution often beome clearly understood only through the effort to express them in the program – this is where the choice of programming language matters."[22] Those who design EDA environments have advocated using an interpreted shell language, such as LISP or S, rather than compiled languages, like Pascal and C.[12] Programs in an EDA environment are fluid; people frequently make small changes on a daily basis, as they try various ways of looking at data. The principal advantage of interpreted languages is their ability to respond instantaneously to these changes without leaving the environment — compilation is an obtrusive activity, and it takes time. This argument is compelling, but a successful toolkit must attract its potential community of users, and it seems unlikely that many accelerator physicists will ever program in LISP: LISP's source code is not transparent, and most people not in the artificial intelligence business are not attracted to it. My current preference is to work in C++ .[22,13] I anticipate that opposition to this powerful and easy language may be milder, and anyone who invests the four or five days necessary to learn it will (almost surely) never program in FORTRAN again. C++ was built upon C by adding many of the features which make Ada so useful: "class" structures permit the creation of legitimate new variable types which then act like part of the language itself; classes permit data hiding and inheritance, leading to *easy and natural* implementations of object-oriented programming; dynamic memory allocation is trivial; and C++ links easily with standard libraries, such as IMSL, so no one need fear losing access to useful software.

Object-oriented programming is a new methodology which changes the very approach to programming. Rather than immediately asking, "How do I write the program?" one first steps back and asks, "What objects are most convenient for expressing the problem and obtaining a solution?" It is a difficult concept to describe to someone who has not tried it. As a metaphor, consider sitting behind the wheel of a new car. What you interact with are fairly standard objects: an ignition switch, a steering wheel, an accelerator pedal, and so forth. The implementation of those objects and the hardware to which they are connected may vary between cars, but their *functions* are

*Operated by the Universities Research Association, Inc. under contract with the U.S. Department of Energy.

[1] For those to whom it is not obvious, I recommend reading Tufte's beautiful masterpiece.[23]

familiar and independent of the implementation. You thus know what needs to be done in order to start the car and get it moving; regardless of the make or model, there is no need to refer to complicated manuals to perform these simple tasks.

Object-oriented programming tries to do that with computers as well, the most successful application being the Macintosh interface built upon Xerox PARC's Smalltalk. In our own world, consider, for example, that a `tuneDiagram` may be a useful object to use in a variety of programs. Its principal function would be the selection of a working point by placing a cursor in the appropriate location in the diagram. Having said that much, I have almost completely specified everything that an applications program needs to know about the object; its implementation is then a logically independent task which can be carried out in a number of possibly machine-dependent ways. (The applications program itself stays machine independent; all the machine dependence gets buried in the implementation of objects.)

There is room for debate on the relative merits of interpretive languages, like LISP, and compiled languages, like C++ , but what seems beyond doubt is that FORTRAN, whose principal advantage is its long history, is *not* the language of choice for developing an EOA environment. (This proposition is either completely obvious, or its justification is too lengthy to include here.)

Graphics One of the aphorisms attributed to the late Richard Feynman, but probably of multiple informal authorship, in various forms, observes that, (paraphrased) If you want to understand an electron, it helps to look at it. It also has been noted by many that advances in science have followed the development of new tools: better instrumentation leads to new observations, which in turn lead to new understanding, which provides new questions for further experiments, and so forth. Both of these statements can be expected to hold true for the study of nonlinear dynamical systems as well. The use of computer graphics to discover the phenomenon of "strange attractors" in dissipative systems and to learn the approach to chaos in one degree-of-freedom Hamiltonian systems is well known. We must push this technology to the limit in order to explore the behavior of multi-dimensional Hamiltonian systems.

The purpose of graphics in an EOA environment is *not* to draw pictures; the purpose of graphics is *to act as the interface between a human being and information buried in a computer.* Accordingly, graphics hardware and software are no small considerations. EOA requires creating and naming graphics objects, object instancing, hierarchical ordering, and easy connectivity between graphics objects and peripheral devices. Because of this need to create, edit, and *interact with* objects in the graphics world, EOA graphics software should be based on Phigs-like protocols rather than ACM Core or GKS, despite the fact that Phigs is not yet a standard.

At Fermilab we offload EOA graphics tasks onto an Evans and Sutherland PS390 terminal, connected via Ethernet to a VAX (VMS) cluster; this arrangement will soon be enhanced by establishing communication with a SUN (Unix) workstation. (This is not necessarily the best possible system, but it works tolerably well, and the PS390 interfaced easily with our VAX. Computer hardware is changing so rapidly that devices can become obsolete while purchase requisitions go through the process of being approved. Any list of competitive alternatives would include the fully integrated Silicon Graphics IRIS workstation and the new Ardent super-computer.) Peripheral devices include a data tablet and a dialbox. The natural use of the latter is in performing viewing transformations of displayed data — rotation, translation, scaling, and the like. However, the dials can be connected to any node in the hierarchy that accepts numeric input. They therefore can be used to perform a large variety of other tasks, such as: (a) a dial can act like a "gain" knob on an oscilloscope by preferentially scaling data in one direction without affecting other graphics objects; (b) by connecting a dial to a decision node in the graphics hierarchy, and instancing a set of orbits under this node, it is possible to "flip" through the orbits, enabling a crude form of animation; (c) dial readings can be sent back to the host computer and connected to variables in a program. Most importantly, the dials can be connected and disconnected dynamically so that their function at any given moment depends on what the user wants to do.

AESOP AESOP (Analysis and Exploration of Simulated Orbits in Phasespace) is our first attempt to create a prototype EOA graphics interface at Fermilab. It was written as a shell which a "user" can link to any four-dimensional mapping of his choice. A deliberate effort was made to separate the graphics functions from the physics so that others would find the shell easy to use. It has been moderately successful at this: I have been using it to explore the offset beam-beam interaction while others at Fermilab have linked AESOP to TEAPOT, TEVLAT, or their own mapping routines.

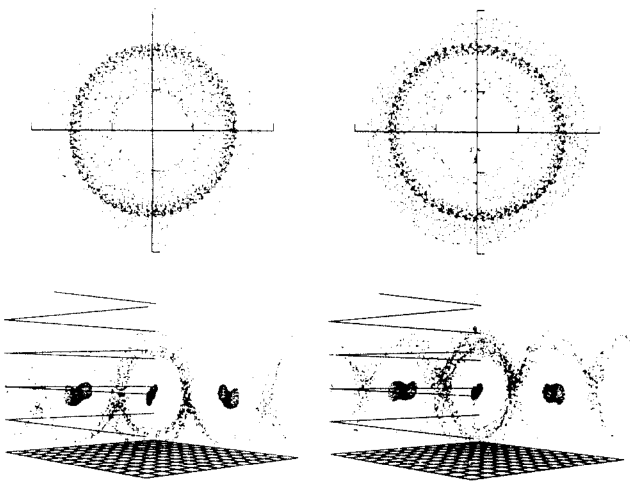
Figure 1: Separatrix of a $2\nu_x - 2\nu_y$ resonance and island orbits.

[17,21,11] A typical AESOP screen, showing the separatrix and two island orbits of the $2\nu_x - 2\nu_y$ resonance generated by the beam-beam interaction, is displayed in Figure 1. The top two viewports contain two-dimensional projections of four-dimensional transverse phase space: the upper left (right) shows the orbit projected along normalized horizontal (vertical) Cartesian coordinates. The three-dimensional projections in the lower viewports are "angle-angle-action" or $\delta\delta I$ plots, as defined in references [14, p.278] and [17]; this type of display has also been advocated by Ruth, *et al.* [29] and has been used by others as well. (As an upgrade, I plan to diagonalize the auto-covariance matrix of the action variables in order to suggest the best linear combinations for display. This should be especially useful near a dominant resonance, where the eigenvalue along the resonance direction will be small.) The interactive part of AESOP allows the user to select "initial" conditions for the orbit by using the data tablet to pick their locations in the top two viewports; an upgraded version will enable the use of the bottom two viewports by using a four-dimensional cursor connected to the dialbox. AESOP allows the user to change the control parameters of his mapping — tune, beambeam tune shift, multipole strengths, or whatever — so that he can quickly explore orbit variations following a change in control parameters. If the analyst finds orbits that are noteworthy, they can be stored in a file, called a "fable."[2] The fables are, if you will, entries in an electronic logbook. They can be edited and suitable comment added so that anyone can read them back and understand their significance.

Slicing and dicing *A capability for doing easy things easily* will be an indispensable attribute of a functioning EOA environment. Accordingly, an interpreter for performing operations similar to database filters will be an essential component. For example, it is frequently useful to choose parts of an orbit where some criterion is satisfied, such as when one wants to correlate some statistic, like eigenvalues of the single-turn Jacobian, with a phase space structure, like a separatrix. AESOP then would have to interpret command sequences like, "`Select all points on` *this* (chosen with a cursor) `orbit with` $\delta p/p > 0.001$. `Display them in a` $\delta\delta I$ `plot in viewport 5. Color red all those with at least one eigenvalue of the single-turn Jacobian off the unit circle.`" One then could *quickly* test hypotheses and guesses, such as whether large eigenvalues correlate with passage through a resonance separatrix which happens to move into the vicinity of an orbit when $\delta p/p$ is large. Slicing and dicing data in this way is the sort of operation that database programs do well, and it will be necessary to include similar kinds of command interpreters in the EOA environment. A capability for offloading such calculations in a multi-tasking environment, would also be useful. The analyst could then perform other operations rather than remain idle while waiting for results to emerge.

Perturbation theory It is said that all reasoning is based on metaphor. Perturbation theory seeks to establish connections between systems which are not yet understood and those that are, in the hope of transferring this understanding from the latter to the former. Although there is no unique,

[2]Get it? AESOP's fables ... get it?? Rather than dwell on the negative connotation of a fable as a story of questionable veracity, I prefer to think of it as one which is designed to teach a lesson.

best way of doing this, most of the promising methods which have been studied in the last few years seek to construct a set of "normal form" variables in which to express the Hamiltonian.[1,25,14,15] One feature which all approaches have in common is the attempt made by their adherents to automate them as much as possible, the most developed of these probably being the MARYLIE code of Dragt and his students [1] and the DA software package of Berz [2]. Like Dragt, I (once) prefer(red) to automate a Lie algebraic series, but many others work with canonical transformations ala generating functions.[3] With full automation, there is no reason why access to any perturbative calculation should be reserved to experts. A functioning EOA environment would contain a battery of these procedures — enabling, for example, the calculation and display of distorted KAM tori, amplitude dependence of tunes, or distorted resonance separatrices for comparison with tracking — with a user-interface so friendly that they become trivial to call up and use. For comparison to non-integrable perturbative models, such as a Hamiltonian with two resonance terms, we would need for a symplectic numerical integrator.[3,9]

Separatrix search We who labor in the vineyard of Hamiltonian dynamics are at a disadvantage compared to our colleagues who work with dissipative systems. The phase space of a Hamiltonian system is a symplectic manifold. The practical consequence of this is almost overwhelming: whereas our dissipative friends can start anywhere in their phase spaces and are almost guaranteed to converge onto interesting structures, such as attractors, we must painstakingly hunt through ours for the important structures, which typically are separatrices or stability boundaries. Finding separatrices is *the* fundamental computational problem in studying the dynamics of periodic Hamiltonian systems. It is relatively easy to solve in two dimensions (one degree of freedom): resonant orbits, stable or unstable, are periodic and generally can be found by employing Newton's method on the appropriate iterate of the period-advance map. The separatrix is then constructed by (a) linearizing the map about an unstable resonant orbit, (b) finding the eigenvectors of the linearized map, to get the directions of the stable and unstable manifolds, and (c) iterating the map, beginning from a small displacement along the unstable manifold. However, higher dimensional systems are more difficult: resonant orbits are generally not periodic, so Newton's method is applicable only in special cases, and, even when integrable, separatrices are themselves complicated, multi-dimensional objects, difficult to visualize and to describe.[16] Having an analytic approximation to their locations helps, and perturbation theoretic tools, such as the techniques based on Hamilton-Jacobi theory [25], may be useful here. In the absence of such an approximation, we must fall back on a graphics-oriented, brute force search. This is not good enough: Think of Newton's method as the replacement of a symplectic map with a dissipative map in such a way that a fixed point of the former is an attractor of the latter. Might there not be an appropriate generalization when we seek not a fixed point but a resonant orbit?

Measuring chaos An EOA environment should contain a set of tools for detecting and quantifying chaos, especially if one wants to explore systematically the characteristics of a large number of orbits (assembly line EOA). One obvious candidate is the computation of Lyapunov exponents.[10,5] Here is another calculation which could be done as a sub-process, as the number of turns required to evaluate these numbers can be large; the analyst may prefer to continue performing other tasks while the progress of this calculation is displayed in a strip chart on his screen.

More detailed "proximity analysis" might examine the turn-by-turn behavior of the eigenvalues of the Jacobian, as is done in reference [5]. Chaos is associated with noisy spectral transforms as well, and these should not be neglected as a possible diagnostics. Minimal spanning tree clustering may be able to estimate geometric characteristics of an orbit, such as its regularity, fairly quickly.

Although rambling, this listing is not exhaustive. There are obvious things which can be done immediately, and if a full EOA environment is successfully to come into being, it will have to prove useful in its early and intermediate stages. Indeed, experience gained by users during the early stages will be invaluable in shaping the direction of future development. However, completing the job, will clearly require a serious, long term commitment in a supportive atmosphere (relatively) free from crises.

[3]On the other hand, unlike Dragt or Berz, my efforts have had all the dramatic impact of a gnat landing on the shoulders of a hairy water buffalo, as was succinctly summarized in reference [19].

References

[1] Alex J. Dragt, *et al.* Lie algebraic treatment of linear and nonlinear beam dynamics. In *Annual Review of Nuclear and Particle Science, Vol. 38*, pages 455–496, Annual Reviews, Inc., 1988.

[2] Martin Berz. Differential algebraic description of beam dynamics to very high orders. *Particle Accelerators*, 24(2), March 1989. to be published.

[3] P. J. Channell and C. Scovel. *Symplectic Integration of Hamiltonian Systems.* Technical Report, Los Alamos National Laboratory, 1988. LA-UR-88-1828.

[4] S. H. C. du Toit, A. G. W. Steyn, and R. H. Stumpf. *Graphical Exploratory Data Analysis.* Springer-Verlag, New York, 1986.

[5] R. Gerig, Y. Chao, and L. Michelotti. Modelling nonlinear behavior in the fermilab main ring. This Conference.

[6] H. Grote, F. C. Iselin, E. Keil, and J. Niederer. Recent developments of mad. This Conference.

[7] David C. Hoaglin, Frederick Mosteller, and John W. Tukey. *Understanding Robust and Exploratory Data Analysis.* John Wiley & Sons, New York, 1983.

[8] F. Christoph Iselin and James Niederer. *The MAD Program. Version 7.2. User's Reference Manual.* Technical Report, European Organization for Nuclear Research, 1988. CERN/LEP-TH/88-38.

[9] Feng Kang. Difference schemes for hamiltonian formalism and symplectic geometry. *Journal of Computational Mathematics*, 4:279–289, 1986.

[10] A. J. Lichtenberg and M. A. Lieberman. *Regular and Stochastic Motion.* Springer-Verlag, New York, 1983.

[11] Sateesh Mane. Private communication.

[12] John A. McDonald and Jan Pedersen. *Computing Environments for Data Analysis: Part 3: Programming Environments.* Technical Report 24, Department of Statistics, Stanford University, May, 1986.

[13] Leo Michelotti. Differential algebras without differentials: an easy c++ implementation. This Conference.

[14] Leo Michelotti. Introduction to the nonlinear dynamics arising from magnetic multipoles. In Melvin Month and Margaret Dienes, editors, *Physics of Particle Accelerators*, American Institute of Physics, 1987. AIP Conference Proceedings No. 153.

[15] Leo Michelotti. Moser-like transformations using the lie transform. *Particle Accelerators*, 16:233, 1985.

[16] Leo Michelotti. Resonance topology. In Eric R. Linstrom and Louise S. Taylor, editors, *Proceedings of the 1987 IEEE Particle Accelerator Conference*, IEEE, 1987.

[17] Leo Michelotti and Selcuk Saritepe. Orbital dynamics in the tevatron double helix. This Conference.

[18] V. Paxson, S. Peggs, C. Saltmarsh, and L. Schachinger. A unified approach to building accelerator simulation software for the ssc. This Conference.

[19] S. G. Peggs and R. M. Talman. Nonlinear problems in accelerator physics. In *Annual Review of Nuclear and Particle Science*, pages 287–325, Annual Reviews, Inc., 1986.

[20] R. D. Ruth, T. Raubenheimer, and R. L. Warnock. Superconvergent tracking and invariant surfaces in phase space. *IEEE Transactions on Nuclear Science*, NS-32(5):2206–2208, 1985.

[21] Steve Stahl. Private communication.

[22] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Massachusetts, 1986.

[23] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.

[24] John W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, Massachusetts, 1977.

[25] R. L. Warnock and R. D. Ruth. Invariant tori through direct solution of the hamilton-jacobi equation. *Physica*, 26D:1–36, 1987.