LBL-25961

Vertically Integrated Simulation Tools for Self-Consistent Tracking and Analysis*

E. Forest and H. Nishimura
Accelerator & Fusion Research Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720

Abstract

A modeling, simulation and analysis code complex, the **Gemini Package**, was developed for the study of single-particle dynamics in the Advanced Light Source (ALS), a 1-2 GeV synchrotron radiation source now being built at Lawrence Berkeley Laboratory. The purpose of this paper is to describe the philosophy behind the package, with special emphasis on our vertical approach.

Introduction

A low emittance ring dedicated either as a damping ring or as a third-generation synchrotron radiation source optimized for insertion devices becomes very sensitive to imperfections. For example, the dynamic aperture will be reduced considerably by errors in the alignment and strength of the magnets. Therefore, the role of a simulation study just after the phase of conceptual design work is to provide a performance evaluation of the machine under a realistic set of errors. There should be tools to carry out necessary studies effectively. In this paper we will describe our approach, with special emphasis on the vertical integration. This has resulted in the creation of the **Gemini Package**.

Requirement and Solution

For a large project, it is proper to put some effort into a dedicated simulation code. Our solution was to create a vertically integrated package, the **Gemini Package**. The phrase "vertical integration" (borrowed from macro-economics) is used here to mean an approach based on a well-defined model of the storage ring that is preserved at all of the hierarchical and sequential stages of manipulations aimed at various goals. Thus, all the codes inside the package are consistent with this model. Here "model" means the Hamiltonian and its integrator.

As shown in Fig. 1, the **Gemini Package** consists of three units: the Pascal program **Gemini**, the Fortran program **Futago**, and a collection of map-analysis programs also written in Fortran. In fact, **Gemini** and **Futago** are used for different purposes, but produce the identical result because of their use of the same model.

To meet our goal, the **Gemini Package** had to obey the following criteria:

* This work is supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Materials Sciences Division, Department of Energy under Contract No. DE-AC03-76SF00098.

1) *Physical Requirement*

Use qualitatively correct Hamiltonian and integrator.

As we have mentioned, the choice of model ultimately determines the validity of any simulation. For example, the complete Hamiltonian for a *combined-function sector bend* (i.e., a group of elements including a drift space, a bending, a quadrupole, a sextupole, and higher multipole magnets), excluding wigglers/undulators and RF acceleration cavities (which are treated separately), is given by:

$$H = -(1+\frac{x}{\rho})((1+\delta)^2 - p_x^2 - p_y^2)^{\frac{1}{2}} + \frac{x^2}{2\rho^2} + \frac{x}{\rho} - \frac{p_\tau}{\beta} - \frac{q}{p_0}A_z$$

$$1+\delta = (1 - 2\frac{p_\tau}{\beta} + p_\tau^2)^{\frac{1}{2}}$$

Here A_z is the vector potential in cylindrical coordinates, (x,p_x,y,p_y) are the regular transverse variables and (τ,p_τ) are the differential time of flight and energy.

Now, the question is whether we should stick to this complete Hamiltonian or introduce some approximation based on physical systems. We have chosen to expand the square-root, leading to the approximate Hamiltonian

$$K = -\frac{x}{\rho}\delta + \frac{x^2}{2\rho^2} + \frac{p_x^2+p_y^2}{2(1+\delta)} - \Delta\delta - \frac{q}{p_0}A_z$$

It is possible to check the validity of this approximation by making use of the code TEAPOT,[1] which calculates the square-root in its model. These tests have confirmed that this approximate Hamiltonian, K, is sufficient for our simulation study on ALS. We have also neglected *realistic fringe fields* because of the size of the ring. It is worth noting that neither of these approximations is valid for a small ring that has bending magnets with a large bending angle. For its realm of validity, the **Gemini Package** has been made generic, but these approximations have to be removed for a small ring. Our group is currently beginning work on a GURU code that will accept any 6-d Hamiltonian as its input.

The advantage of the Hamiltonian K resides in its simplicity and the possibility of implementing the 4th-order explicit symplectic integrator[2] of the Hamiltonian to get a solution. This means that our code is a kick code, and allows the path difference of the off-energy particle to be calculated exactly, permitting accurate tracking with synchrotron oscillations.

CH2669-0/89/0000-1304$01.00©1989 IEEE

2) *Computer Requirement*
User Symbiosis.

A simulation code must be flexible. Since it cannot contain in advance all the logic that may be required by users, it should accept logic defined by users as its input. This is one of the most important and difficult parts of a simulation code. The way in which this is taken care of in **Gemini** is described below.

As depicted in Fig. 2, **Gemini** accepts two kinds of input files: the lattice file and the command file. The lattice file contains essentially the definition of the ideal lattice of the ring. For example, a quadrupole will be specified by its gradient and its length. Since we use a symplectic integrator, the number of steps will also be specified. This is similar to the Standard Input File Format (as used in MAD[3]) for example), but is simplified.

The unique aspect of **Gemini** is that it has a dedicated Pascal compiler[4]) in it to process the command file that contains the user-defined logic, as shown in the appendix. Therefore, a user has absolute freedom and flexibility because he can use **Gemini** as a computer language for accelerator simulation. Most of the fundamental and frequently-used procedures have been implemented, but a user still can *program* his own procedure by making use of the predefined ones as building blocks. Here, the point is that the program **Gemini** itself is also written in Pascal, which means that the meta-language is same as the language a user can use. Therefore, **Gemini** is self-expandable. This method has been originally applied to the simulation code **Tracy**[5]) by one of the authors and proved to work effectively. Any routine used as the input for **Gemini** can also be put inside of the code for general use. This is what we mean by **symbiosis** between the command file and the program **Gemini.**

The program **Gemini** can do tracking and some linear analysis, hence it can be viewed as a regular tracking code. However its Pascal nature has two drawbacks: fast tracking on CRAY machines can only be done in Fortran and, more importantly, the map-production and -analysis codes based on the **Differential Algebra Package**[6]) are also written in Fortran.

How do we resolve this problem? To see how this is solved, one can think of **Gemini** as a realistic ring production facility, i.e., the purpose of the code is to simulate the "commissioning" of the realistic machine. Indeed at any stage in the execution of **Gemini**, the database is modified by the instructions in the command file. This database contains all the ideal lattice information (from the lattice file) and the current modifications leading to the realistic lattice, including errors of the various lattice elements. This is achieved by including operations of a Galilean group (systematic and random translations and rotations to simulate misalignments) in the code. At any stage, a procedure of **Gemini** can generate the full Hamiltonian of the realistic ring and print it on an external file. This is when the Fortran code **Futago** enters into play. **Futago** reads in the Hamiltonian and produces maps (tracking results are zeroth order maps). Since **Futago**'s Hamiltonian (integrator included) is identical to **Gemini**'s, the results are, to machine precision, the same (our machine file bears extreme resemblance to the TEAPOT machine file and the flat file used at SSC).

3) *Mathematical Requirement*
Complete mathematical self-consistency; completely self-consistent perturbation theory for all situations generated by the program.

The fact that the **Futago** model Hamiltonian is identical to that of **Gemini** implies that this requirement is automatically satisfied as far as map generation is concerned. There remains the problem of perturbation theory. Following the theoretical ideas recently developed,[7]) one can reformulate *all* of single-particle dynamics using a *Hamiltonian-free description* of the theory. This reformulated theory provides an extremely powerful conceptual and computational tool. In this formulation, calculations of tunes, distortion functions, resonances etc., are done on the one-turn map. Also, the concepts of phase advances and Floquet variables are totally described in terms of maps between two observation points. To prove the power of this view, using the **Differential Algebra** tools, we applied it to standard perturbation theory (**DALIE** package of Fig. 1) and were capable of producing very high order results (15^{th} order in 4-d, for example) on a realistic lattice of ALS and also of SSC.

The same Hamiltonian-free theory can also be applied to nonperturbative approaches.[8]) This latter approach recently used maps to produce extremely powerful results on invariant tori. We plan to integrate our tools with this nonperturbative approach in the future.

Self-Consistency and a Realistic Simulation

As we have mentioned, one can take **Gemini** as the code to simulate the *commissioning* of a realistic machine. It is interesting that the *commissioning* permitted by the grammatical rigor of **Gemini** can be truly simulated. Indeed, the code can contain procedures which really mimic the actual measurements necessary in the commissioning and operation of the ALS. Originally this is what motivated the development of **Tracy**, a high-level programming code for simulation of machine operation. By integrating this code into a vertical package, we could simulate and analyze not only the performance but also the operation of the ALS in a completely self-consistent manner.

Conclusion

We have emphasized the concept of vertical integration and the **Gemini Package** of simulation codes. Our approach is complementary to a horizontally integrated code, such as MAD. We think it is valuable to create a dedicated vertical package for a fixed ring to carry out a series of heavy simulation studies in a consistent manner. A horizontal approach works effectively at the early stage of the design study of a ring, but at the final stage a vertical method becomes important, especially in conjunction with the commissioning of the machine and its control system.

Acknowledgments

We express our thanks to M. Berz for providing us a powerful DA package and to S. Chattopadhyay for his valuable comments and encouragement. And special thanks to the patient users, M. Zisman, A. Jackson and C. Kim, for much advice during the *commissioning phase* of this package.

References:

1) L. Schachinger and R. Talman, Particle Accelerators, **22**, 35 (1987).
2) R. D. Ruth, private communication.
3) F. C. Iselin and J. Niederer, CERN/LEP-TH/88-38.
4) Based on the Pascal-S compiler by N. Wirth, which is described in the following book: R. E. Berry, Programming Language Translation, 1981, Ellis Horwood Limited, England.
5) M. Berz, SSC-152, 1988. E. Forest, M. Berz and J. Irwin, SSC-166, 1988.
6) H. Nishimura, LBL-25236, 1988.
7) E. Forest, SSC-111, 1987; A. Bazzani, P. Mazzanti, G. Servizi and G. Turchetti, CERN SPS/88-2(AMS), LHC Note 66, 1988.
8) R. L. Warnock, R. D. Ruth, W. Gabella and K. Ecklund, SLAC-PUB-4846, 1989.

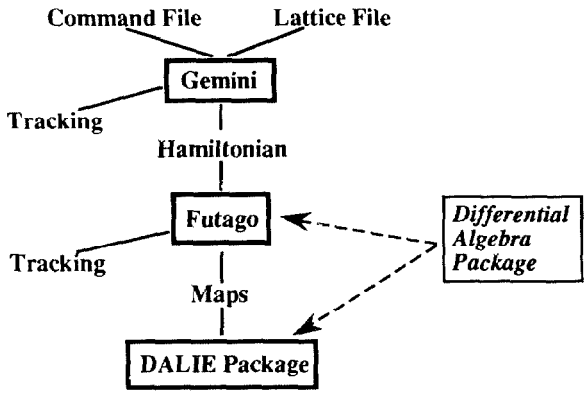

Fig.1. Elements of Gemini Package

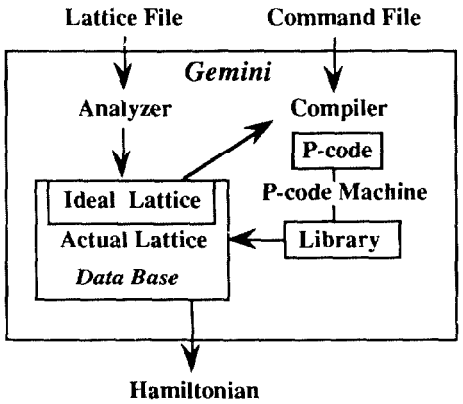

Fig.2. Inside of Gemini

Appendix: Example of a Command File.

```
program EXAM(ALSM1.LAT);{ ALSM1.LAT:= Lattice File }
*  Very simple example of a Gemini command file.
*  Simulate imperfection, apply a COD correction and
*  ship machine files to Futago.
 var i,j:integer;
    Nux,Nuy, x,y,t,k, Xm,Xa,Xr,Zm,Za,Zr:real;
begin
*** Set up the ideal lattice ***
    GetCellFnc(1);{ Calculate tune, dispersion and chromaticity }
    FitDisp(0.0);{ Fit dispersion @ Symmetry point }
    FitTune(14.27696,8.179010);
    FitChrom(0.0,0.0);{ Chromaticity fitting }
*   Get the sextupole strength, then turn them off
    GetKvalue(SF,KSF); GetKvalue(SD,KSD);
    SetKvalue(SF,0.0);   SetKvalue(SD,0.0);
    StoreCell;{ Save the ideal lattice}
    GenMfile('ALSMideal.dat'); { Generate a machine file}
*** Set up COD Correction System ***
    InitBUMP(0.2,0.1);{ Initialize Local Bump }
*** Set up Imperfections
    SetRanCut(2.0);   {Truncate the normal distribution at 2 sigma }
    InitRand(12345);  {Initialize the random seed }
*   Assign Random Errors to Elements
    x:=0.15/1000;      { horizontal misalignment = 0.15 mm  }
    y:=0.15/1000;      { vertical  misalignment  = 0.15 mm  }
    t:=RtoD(0.5E-3);   { roll angle error        = 0.5 mrad }
    k:=0.1/100;        { field strength error    = 0.1  %   }
*   Assign errors to elements
    RndError( BEND , x, y, t, k);
    RndError( Q1   , x, y, t, k);
    RndError( Q2   , x, y, t, k);
    RndError( QF1  , x, y, t, k);


*** Get statistics for 20 machines ***
 for i:=1 to 20 do
  begin
    SetUpError;{ distribute errors }
    ClearCOH;{ clear horizontal correctors }
    ClearCOV;{ clear vertical correctors }
    GetCOD(0.0);{Get COD for dp/p=0}
    { Get  COD statistics before correction }
    GetCODstat(Xm,Xa,Xr, Zm,Za,Zr);
    GetTune(nux,nuy); { get betatron tunes before correction }
    write(i:3,Xr:9:5,Zr:9:5,Nux:9:5,Nuy:9:5);
    for j:=1 to 20 do ExecBUMP(2.0,1);{ COD correction}
    GetCellFnc(0);
    FitTune(14.27696,8.179010); { fit tunes }
    for j:=1 to 10 do ExecBUMP(2.0,1);
*   Turn on sextupoles
    SetKvalue(SF,KSF);  SetKvalue(SD,KSD);
    GetCellFnc(0);{Get twiss function}
    FitTune(14.27696,8.179010); { fit tunes }
    { Get COD statistics after correction }
    GetCODstat(Xm,Xa,Xr, Zm,Za,Zr);
    GetTune(nux,nuy);{ Get betatron tunes after COD correction }
    writeln(Xr:9:5,Zr:9:5,Nux:9:5,Nuy:9:5);
    GenMfiles('ALSM',i);{Generate a Machine File  --> Futago}
    NewSeed;{Get a new random statistics}
    RecallCell;{Load the ideal lattice}
  end;
 end.
```