DESIGN AND APPLICATIONS OF THE BMAD LIBRARY FOR THE SIMULATION OF PARTICLE BEAMS AND X-RAYS[*]

D. Sagan, Wilson Laboratory, Cornell University, Ithaca, NY, 14853, USA

Abstract

The open-source Bmad software library has been developed for simulating both charged particle beams and X-rays. Owing to its modular, object-oriented design, Bmad has proved to be versatile, and is currently used in a number of programs at Cornell's Laboratory for Elementary-Particle Physics. This paper will discuss the design of the Bmad library and how features such as the ability to simulate overlapping elements, the ability to define the action of control-room "knobs," and the ability to select among different tracking algorithms, have all contributed to a flexible simulation environment that eases the task of both programmers and users alike. Also discussed are the uses that Bmad has been put to, including machine control and the integration of particle beam and X-ray simulations.

INTRODUCTION

Bmad, an open-source software library for simulating both charged particle beams and X-rays, has been in development at Cornell beginning in the 1990s[1]. Originally, the syntax for the lattice files for Bmad was patterned after the syntax of the MAD program[2]. Since, at that time, only a subset of the MAD language was used, the name "Baby MAD" or "Bmad" for short was chosen.

The initial purpose for developing Bmad was modest: Simply to be able to compute Twiss parameters and the closed orbit, and to provide a standard lattice description format. As Bmad evolved, the scope of Bmad expanded so that currently Bmad can simulate such things as spin, X-ray photons, coherent synchrotron radiation, intra-beam scattering, Touschek effect, etc., etc.

Bmad has proved to be versatile and, as a result, is now used in a number of programs at Cornell. Experience with Bmad has shown that there are a number of design features that have made Bmad especially useful. The purpose of this paper is to discuss this and to discuss some of the applications that Bmad has been used for. Finally, plans for the future of Bmad will be presented.

DESIGN PHILOSOPHY

The aim of the Bmad project is to:

- Cut down on the time needed to develop programs.
- Minimize computation times.
- Cut down on programming errors.
- Provide a simple mechanism for lattice function calculations from within programs.
- Provide a flexible and powerful lattice input format.
- Standardize sharing of lattices between programs.

To maximize code reuse, Bmad, written in Fortran, is designed to be object oriented from the ground up. For example, it takes only one line of executable code to parse a lattice file:

```
type (lat_struct) lat
call bmad_parser('lat.bmad', lat)
```

The call to `bmad_parser` in this example causes the file named "lat.bmad" to be parsed and the information to be stored in a variable named `lat` of type `lat_struct` (equivalent to a C++ class). For communication with C++ code, Bmad defines a set of C++ classes and there are interface routines to convey information between between the Fortran structures and the C++ classes.

From the beginning, Bmad development has been driven by the need to solve the practical problems arising from the requirements of machine simulation, lattice design and machine control. As a result, one emphasis of Bmad development has been on minimizing the bookkeeping tasks of both programmer and user. There are a number of features that have proved to be especially useful in this regard, and some are discussed below: The ability to superimpose elements on top of other elements, the ability to slice elements into pieces, the ability to define controller elements that control the attributes of other elements, and the ability to choose the algorithm used for particle tracking.

Superposition of Elements

"Superposition" is the ability to overlap lattice elements spatially. Figure 1 shows an example which is a greatly simplified version of the IR region of Cornell's CESR storage ring when CESR was an e+/e– collider. As shown in Fig. 1A, two quadrupoles named `q1w` and `q1e` are partially inside and partially outside the interaction region solenoid named `cleo`. In the lattice file, the IR region layout is defined to be

```
cesr: line = (... q1e, dft1, ip, dft1, q1w ...)
cleo: solenoid, l = 3.51, superimpose, ref = ip
```

The line named `cesr` ignores the solenoid and just contains the interaction point marker element named `ip` which is surrounded by two drifts named `dft1` which are, in turn, surrounded by the `q1w` and `q1e` quadrupoles. The solenoid is added to the layout on the second line by using superposition. The "ref = ip" indicates that the solenoid is placed relative to `ip`. The default, which is used here, is to place the center of the superimposed `cleo` element at the center of the `ip` reference element. Within a program, the representation of the lattice in the `lat_struct` structure that Bmad creates will contain two lists: One list,

[*] Work supported by the National Science Foundation under contract PHY-1002467.

ISBN 978-3-95450-116-8

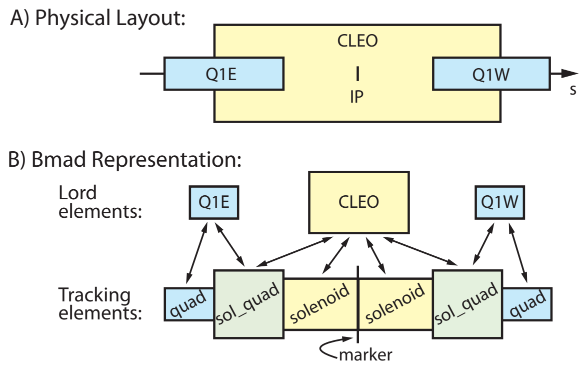

Figure 1: Superposition Example. A) Interaction region layout with quadrupoles overlapping a solenoid. B) The Bmad lattice representation has a list of split elements to track through and the undivided "lord" elements. Pointers (double headed arrows), keep track of the correspondence between the lords and their "slaves."

called the "tracking section," contains the elements that are needed for tracking particles. In the current example, as shown in Fig. 1B, the first IR element in the tracking section is a quadrupole that represents the part of `q1e` outside of the solenoid. The next element is a combination solenoid/quadrupole, called a `sol_quad`, that represents the part of `q1e` inside `cleo`, etc. This `sol_quad` element illustrates a restriction on superpositions that the combination of elements must be able to be represented by a valid Bmad element type. Bmad does have a combination solenoid/quadrupole type so superimposing solenoids with quadrupoles works.

The other list in the `lat_struct` that Bmad creates is called the "lord section." This list will contain the undivided elements which, in this case are `q1e`, `q1w`, and `cleo`. Pointers are created between the lords and their "slaves" in the tracking section so that changes in parameters of the lord elements can be transferred to their corresponding slaves by calling the appropriate Bmad routines.

Superposition has proven to be very useful. For one, since Bmad does the bookkeeping of splitting up elements and keeping the slave element attributes up-to-date, there is less chance of an error occurring. Another advantage is that a program itself does not have to "know" about how things are split since Bmad will take care of the details. Thus, for example, varying the field strength of the `cleo` solenoid in a program is trivial.

Element Slicing

Element "slicing" is similar to superposition discussed in the previous subsection in that slicing involves dividing elements longitudinally. The difference is that slicing involves crating a temporary structure within a program that represents a given section of the divided element. Slicing is hidden from the program's user.

Slicing relieves the programmer of a number of burdensome bookkeeping details when doing calculations that involve points internal to the elements. For simulations of such things like intra-beam scattering, Touschek effect, and coherent synchrotron radiation, the ability to step through an element slice-by-slice is essential.

Slicing also alleviates the need to, for example, split quadrupoles in two in the lattice file since calculating, say, the Twiss parameters at the center of any element is easily done on-the-fly using slicing.

Controllers

The ability to define controller elements that control the attributes of other elements has proved to be a very useful feature. Controllers can be used to simulate such things as the effect of control room "knobs" or simulate power supplies that power multiple magnets. A specialized `girder` controller can be used to simulate that effect of a support structure that supports a set of lattice elements. The following shows an example from a standard CESR lattice file:

```
h20w: overlay = {b20w[hkick] :0.5, &
                 b21w[hkick] :0.5}, hkick
h20w[hkick] = 0.01
```

For historical reasons, a controller is called an `overlay`. Here a controller named `h20w` controls the horizontal kick (`hkick`) attribute of two magnets named `b20w` and `b21w`. Setting the `hkick` attribute of the controller to 0.01 in the second line sets the hkick attribute of b20w and b21w to 0.005 each due to the 0.5 coefficients set in the first line.

Being able to define controllers as "just another element" simplifies program development and there is less of a maintenance burden having to maintain separate files specifying the lattice and who controls what.

Tracking Methods

Tracking and transfer map calculations (henceforth "tracking" will mean either one) are at the heart of many simulations, and different problems will have different requirements in terms of accuracy, speed, etc. To provide flexibility, Bmad implements a number of different tracking methods and what method is used can be selected on an element-by-element basis.

The commonly used "bmad_standard" method uses thick element formulas to quickly perform tracking. While not symplectic, this tracking method is useful for such tasks as computing Twiss parameters and finding the closed orbit. As an example, the program `CesrV`, which is used for data taking, machine corrections, and machine analysis at CESR, is able to calculate Twiss and coupling corrections in a few seconds using a single 2.67 GHz Xeon based computer. In this case, there are around 200 variables and 300 data values in a lattice with about 2000 elements.

For symplectic tracking, the "symp_lie_ptc" method, which uses Étienne Forest's PTC package[3], is available. Alternatively, the "taylor" method uses PTC to construct a Taylor map for tracking. Taylor tacking is faster than

ISBN 978-3-95450-116-8

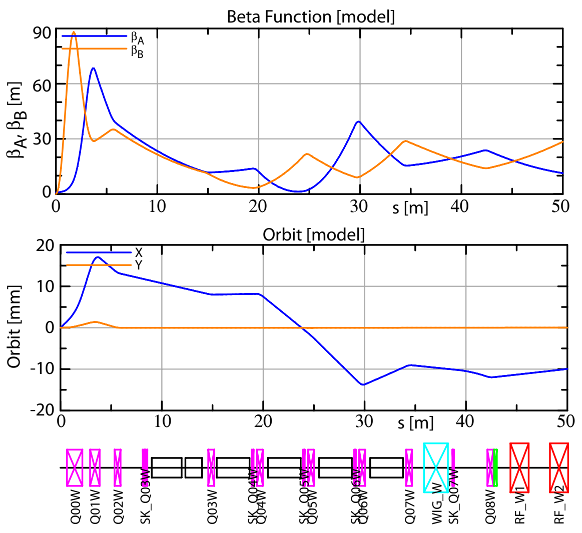

Figure 2: Example of a Tao plotting window.

symp_lie_ptc but will have a limited domain in phase space where the tracking is accurate.

There are a number of other tracking methods such as “runge_kutta” to track through a field profile, “linear” which just uses the linear part of the transfer map, and “custom” which allows for tracking with custom code.

BMAD ECOSYSTEM

The versatility of Bmad has resulted in Bmad being used as the computational engine powering a number of programs. It is thus possible to talk about the Bmad “ecosystem” Examples include

bbu_program A program to simulate the beam-beam breakup instability in Energy Recovery Linacs[4].

cesrv Data taking (orbits, phase/coupling, etc), simulation, and correction program for Cornell’s CESR ring[5].

dark_current_tracker Program for simulating dark current electrons generated in a machine[6].

freq_map Frequency map program.

ibs_sim Analytic intra-beam scattering (IBS) calculation.

synrad3d Tracking of synchrotron radiation photons in the beam chamber including reflections[7].

tao A general purpose design and simulation program[8].

touschek_track Tracking of Touschek particles to determine where they are lost[9].

Three examples are illustrated below.

Tao: General Purpose Simulation Program

The disadvantage of Bmad, common to all software libraries, is that by itself it is not a program. That is, it cannot be run straight out-of-the-box. Over time, it became apparent that there was a great need for a general purpose program that could be used without any programming effort. As a result, a program called Tao[8] (Tool for Accelerator Optics) has been developed that uses Bmad as its calculational engine.

Tao has a general nonlinear optimization algorithm that can do such things as lattice design and orbit response matrix analysis. It has flexible plotting capabilities as displayed in Figs. 2 and 4. In many ways, Tao is similar to programs like MAD. A significant difference is that Tao is built from the ground up to be modular and extensible and so it is relatively easily, to add custom commands to Tao. For example by writing custom commands to read data and set magnet strengths, Tao can be become an on-line control system program to do such things as orbit flattening, etc.

The combination of Bmad and Tao thus gives the best of all possible worlds: The flexibility of a toolkit coupled with the ease of use of a program.

Cornell ERL DOOCS Interface

A control program based upon DOOCS (Distributed Object Oriented Control System), written in C++ and Java, has been developed for the Cornell Energy Recovery Linac[10] (ERL) test machine. Bmad was interfaced to this program for orbit and Twiss parameter calculations. A custom interface layer was developed to be able to translate between the C++ structures on the DOOCS side and the appropriate Bmad structures on the Fortran side. To simplify code development, use was made of the Bmad interface between Bmad Fortran structures and equivalent Bmad C++ structures. For lattice elements, the Bmad structure on the Fortran side is an `ele_struct`. On the C++ side, the equivalent Bmad class is called `C_ele`. To transfer element information from DOOCS to Bmad, for example, a `C_ele` instance is created on the C++ side of the interface and the appropriate information is transferred to it. Simplified, the code looks like:

```
C_ele quad;
quad.name = "q1";
quad.key  = QUADRUPOLE;
quad.value(B1_GRADIENT) = q_grad;
```

The `quad` instance is then passed to the Fortran side of the interface. On the Fortan side the code looks like:

```
subroutine ele_convert (c_ele)
  type (c_ele_struct) c_ele
  type (ele_struct) ele
  call ele_to_f (c_ele, ele)
```

The call to `ele_to_f` effects the transfer of information to the `ele_struct` and the element data can now be used by Bmad proper.

While not as clean as interfacing between two C++ code bases – the interface has two layers instead of one – the structure/class interface provided by Bmad greatly simplifies the integration of Bmad with C++ based code.

Dark Current Simulation

An example of the utility of Bmad in developing new simulation code is the `dark_current_tracker`

ISBN 978-3-95450-116-8

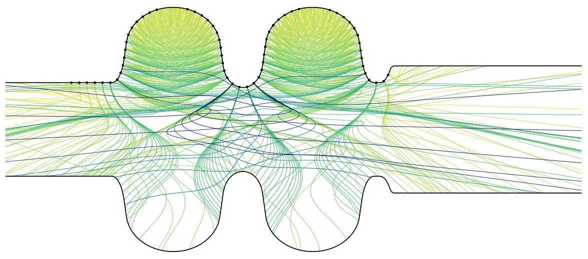

Figure 3: Dark current tracking in an RF accelerating cavity. Only part of the cavity is shown. The black circles are points of high field where particles are generated.

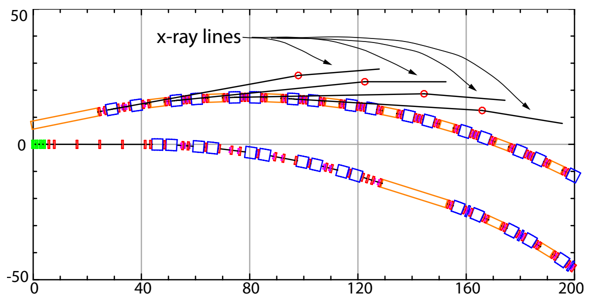

Figure 4: Simple example of the Cornell ERL with four x-ray lines.

program[6]. The problem was to simulate the trajectories of field emitted electrons from the walls of RF cavities. One of the first tasks of the program was to pull in information on the lattice including the beam chamber walls. At the time, Bmad had a model for specifying the wall for X-ray focusing capillaries, and it was decided that this model could be used for the beam chamber walls. This saved an appreciable amount programming time and effort. It also meant that the chamber wall could be specified in the lattice file which simplified the user interface.

Bmad provided many helper routines for the dark current simulation for doing such things as determining whether a particle had hit the wall, etc. Missing was suitable tracking code since the particles could reverse directions and the standard Bmad tracking methods are s-based and so assumed that particles always traveled in one direction. To overcome this, a time-based tracking routine was developed and merged into the Bmad library. Figure 3 shows a simulation from the program. Here the trajectories of a number of particles generated at high field regions in an RF cavity are shown.

With Bmad, the amount of work needed to create the simulation program was greatly reduced and, in turn, the capability of Bmad was increased by inclusion of the new tracking method which could be invoked in any program simply by setting an element's tracking method to use it. Additionally, the added ability to be able to specify a beam chamber wall will help with future simulations.

X-RAY SIMULATIONS

With the advent of Cornell's ERL project, one evolutionary track that Bmad has been following is the ability to simulate X-rays along with particle beams to provide an integrated simulation environment. One issue is the ability to define the entire machine in single lattice which contains both accelerator and X-ray components. Being able to treat the ERL as an integrated whole simplifies a number of simulations. For example, it simplifies the simultaneous simulation of the effect of electron beam movement on all the X-ray lines.

To define a line branching from a machine, Bmad uses `branch` and `photon_branch` elements. A `branch` or `photon_branch` element has zero length and marks the beginning of the branch line. `branch` elements are for describing such things as beam dumps where the charged particle being tracked can branch to. For X-ray lines, the `photon_branch` element is used. For example the following in a lattice file creates a branch:

```
  br1: photon_branch, superimpose, &
                   ref = sa01, to = tg_mono
  tg_mono: line = (...)
```

Here `br1` is a `photon_branch` which is placed at the center of an element named `sa01`. The associated X-ray line (not shown) is called `tg_mono`. There is no limit as to how many branches a lattice can have and branch lines can themselves contain branches ad infinitum. Figure 4 illustrates this showing four X-ray lines attached to an ERL.

For X-ray lines, besides the standard elements like `drift` and `marker`, There are a number of specialized elements that Bmad defines:

```
  capillary,          mirror,
  crystal,            multilayer_mirror
```

The `crystal` element supports both Bragg and Laue diffraction. The `capillary` element simulates X-ray focusing within a fine capillary[11]. The `mirror` element is a bendable mirror, and the `multilayer_mirror` is a mirror that depends upon multiple thin layers of dielectric material for reflection.

PRESENT WORK AND FUTURE PLANS

Bmad has had a steady evolution due to a steadily increasing demand for varying types of simulations. Currently, there are a number of areas of active development.

One thread is driven by the desire to be able to simulate an ERL from generation of the electrons at the gun cathode, through X-ray generation in wigglers and undulators, to propagating the X-rays to the experimental end stations. This work has two parts: Developing low energy simulation capabilities and developing X-ray simulations.

The work on low energy simulations is reaching maturity. Recent work, for example, has resulted in the creation of a `e_gun` element for simulating the cathode region of an electron gun. The major outstanding issue is

ISBN 978-3-95450-116-8

space charge. Since simulating space charge is complicated, rather then reinvent the wheel, the present goal is to integrate Bmad with existing space charge codes Impact-T[12] and OPAL[13].

X-ray simulation development is still in it's infancy. Photon generation code exists for bends and wigglers but not for undulators. The first real-world simulations are currently a work in progress. At this point in time, as mentioned in a previous section, Bmad has some locally developed code for photon tracking through crystals and a few other elements. Having locally developed code provides great flexibility for being able to simulate new effects. However, X-ray simulation can be quite complicated and it would be advantageous to be able to leverage the capabilities of existing codes. To this end, a collaboration is underway to interface Bmad with the Shadow[14] photon simulation code. In the long term, simulation of partially coherent X-rays would be useful. One possibility that is being explored is integration with the code SRW[15] developed by Oleg Chubar.

Up to now, the use of Étienne Forest's PTC code has been restricted to tracking through elements one element at a time. While this provides great flexibility in terms of being able to choose what type of tracking is used for a given element, it does not provide a way of using PTC's powerful analysis tools for calculating such things as beam emittances, resonance driving terms, etc. To remedy this, the appropriate interface code has, of late, been developed so that Bmad can now create a PTC `layout` for analysis. This makes possible the option of using PTC directly with Bmad simply providing the lattice parsing tools. It just takes two executable lines of code to construct a PTC layout from a Bmad lattice file:

```
type (lat_struct) lat
call bmad_parser (file_name, lat)
call lat_to_ptc_layout (lat)
```

In addition to the above, there are a number of minor projects on the drawing board, including development of nonlinear controllers, improved thread safety in multi-threaded environments, improved regression testing, and the interfacing to H5hut[16] – a high-performance library for storing particle position data.

ACKNOWLEDGMENTS

Many people have helped in the development of Bmad. In particular, thanks must go to David Rubin for his continued support and to Étienne Forest whose PTC package brings symplecticity to Bmad. Additionally, thanks must go to Chris Mayes for his work and for the use of Fig. 3.

REFERENCES

[1] D. Sagan, "Bmad: A relativistic charged particle simulation library," *Nuc. Instrum. and Methods in Phys Research A* **558**, 356–359 (2006). `http://www.lepp.cornell.edu/~dcs/bmad`.

[2] See: `http://mad.web.cern.ch/mad/`

[3] E. Forest, and F. Schmidt, "The FPP and PTC Libraries", Proceedings of ICAP 2006, Chamonix, France (2006)

[4] J. A. Crittenden, G. H. Hoffstaetter, M. Liepe *et al.*, "Recent progress on beam-breakup calculations for the Cornell x-ray ERL," *Proceedings of the 2009 Particle Accelerator Conference* (2009).

[5] D. Sagan, R. Meller, R. Littauer and D. Rubin, "Betatron phase and coupling measurements at the Cornell electron/positron storage ring," *Phys. Rev. ST Accel. Beams* **3**, 092801 (2000). Note: While the paper does not mention it, CesrV is the progame being used here for the analysis.

[6] C.E. Mayes, C. Chiu, G.H. Hoffstaetter, V.O. Kostroun, L.M. Nash, D.C. Sagan, "Dark Current Simulations for the Cornell ERL", Proceedings of IPAC2011, San SebastiÌĄn, Spain. 2214 (2011).

[7] L. Boon J. Crittenden, T. Ishibashi, K. Harkay, "Application of the SYNRAD3D Photon-Tracking Model to Shielded Pickup Measurements of Electron Cloud Buildup at CesrTA", *Proceedings of the 2011 International Particle Accelerator Conference* (2011).

[8] D. Sagan, and J. C. Smith, "The Tao accelerator simulation program," *Proc. 2005 Part. Accel. Conf.*, 4159–61 (2005).

[9] M. P. Ehrlichman, and G. H. Hoffstaetter, "Collimating Touschek particles in an energy recovery linear accelerator," *Proceedings of ERL09* (2009).

[10] G. H. Hoffstaetter, B. Barstow, I. Bazarov *et al.*, "The Cornell ERL prototype project," *Proceedings of the 2003 Particle Accelerator Conference*, 192–194 (2003).

[11] S.A. Hoffian, D.J. Thiel, D.H.Bilderback, "Developments in Tapered Monocapillary and Polycapillary Glass Concentrators", Nucl. Instr. & methods in Phys. Res. A347, 384, (1994).

[12] J. Qiang, R. D. Ryne, S. Habib, V. and Decyk, "An object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators," *J. Comp. Phys.* **163**, 434 (2000).

[13] A. Adelmann, Ch. Kraus, Y. Ineichen *et al.*, "The Object Oriented Parallel Accelerator Library (OPAL), Design, Implementation and Application," *Proceedings of the 2009 Particle Accelerator Conference* (2009). `http://amas.web.psi.ch/docs/`.

[14] C. Welnak, G. J. Chena, and F. Cerrina, "SHADOW: A synchrotron radiation and x-ray optics simulation tool," *Nuc. Instrum. and Methods in Phys Research A*, 344–347 (1994).

[15] O. Chubar and P. Elleaume, "Accurate and efficient computation of synchrotron radiation in the near field region," Proceedings of the 6th European Particle Accelerator Conference p. 1177 (1998).

[16] M. Howison, A. Adelmann, E. Wes Bethel, Achim Gsell, Benedikt Oswald, Prabhat, "H5hut: A High-Performance I/O Library for Particle-based Simulations". IASDS 2010 Workshop on Interfaces and Abstractions for Scientific Data Storage, Heraklion, Crete, Greece, September 20-24 (2010).