Lucretia: A MATLAB-BASED TOOLBOX FOR THE MODELLING AND SIMULATION OF SINGLE-PASS ELECTRON BEAM TRANSPORT SYSTEMS *

P. Tenenbaum †, SLAC, Stanford, CA, USA

Abstract

We report on Lucretia, a new simulation tool for the study of single-pass electron beam transport systems. Lucretia supports a combination of analytic and tracking techniques to model the tuning and operation of bunch compressors, linear accelerators, and beam delivery systems of linear colliders and linac-driven Free Electron Laser (FEL) facilities. Extensive use of Matlab scripting, graphics, and numerical capabilities maximize the flexibility of the system, and emphasis has been placed on representing and preserving the fixed relationships between elements (common girders, power supplies, etc.) which must be respected in the design of tuning algorithms. An overview of the code organization, some simple examples, and plans for future development are discussed.

INTRODUCTION

In 2002, the Linear Accelerator Research program (LIAR) [1] was modified to extend its capabilities to the full Low Emittance Transport (LET) of a linear collider (including the bunch compressor, linac, and beam delivery system) by adding a ray-tracing tracking engine which coexisted with the native macroparticle tracker of LIAR; at the same time, the source code was reorganized to create a subroutine, callable from Matlab, which could combine the beam dynamics simulation capability of LIAR with the power and flexibility of the Matlab environment. The final product of these modifications is described in [2].

The mat-LIAR package was used successfully for a number of Start-to-End (S2E) simulations, running from each damping ring to the IP, during the preparation of the International Linear Collider Technical Review Committee's 2003 report [3], but was subject to a number of limitations:

- The use of two independent tracking engines (the LIAR engine for linacs, the ray-tracer for bunch compressors and final focus areas), as well as the interfacing back and forth between them, was complicated and error-prone, in particular resulting in loss of information about the longitudinal charge distribution produced by a bunch compressor
- Accessing and changing the lattice and beam definition used by mat-LIAR was cumbersome, often relying on passing text strings to the command parser of the original LIAR program
- The mat-LIAR package inherited a number of limitations from the original LIAR program; most notably, tracking a beam through a subsection of the accelerator was not possible
- Since LIAR was originally intended to study only linacs, there was no native provision for studying two beamlines which intersect at the collision point; studies of colliding beams were performed using two mat-LIAR images, each of which had one of the beamlines loaded into its data structures.

We have developed a new simulation package for high performance single pass electron transport lines which addresses the limitations above and extends LIAR's capabilities for realistic simulations of accelerator operations, tuning, and stabilization, and also makes more complete use of the Matlab environment. The package, Lucretia, is suitable for simulations of linear collider low emittance transport regions (bunch compressor, linac, and beam delivery system), as well as linac-driven free electron lasers.

USER INTERFACE

Lucretia follows the conceptual design of the Accelerator Toolbox (AT) [4]: a collection of Matlab scripts and functions, almost all written in the Matlab programming language, rather than a fully self-contained program with a command-line interpreter. This organization allows the user to seamlessly integrate Matlab's graphics, mathematics, and data-management tools with Lucretia's tools for accelerator simulation.

Accelerator Representation

The accelerator is represented by a set of five Matlab data structures. Cell Array BEAMLINE contains the beamline elements, fully-instantiated, in beamline order; arrays GIRDER, KLYSTRON, and PS contain the properties of mechanical supports, RF power sources, and DC power sources, respectively; array WF contains data related to single- and multi-bunch wakefields. The data structures are cross-referenced to one another (for example, BEAMLINE{}.Klystron shows which klystron powers a given beamline RF device, while KLYSTRON().Element shows which elements are powered by a given klystron). Since all of the data is resident in Matlab, it is possible for the user

* Work supported by the United States Department of Energy, Contract DE-AC02-76SF00515.

† quarkpt@slac.stanford.edu

to examine any of the data at any time and ensure that parameters are properly set, etc. Any switches which control tracking behavior (wakefields enabled or disabled, etc.) are set element-by-element and stored for each element in the BEAMLINE array.

Each element in the beamline can have mechanical offsets and excitation errors; beam position monitors can have, in addition, electrical offsets and resolution limits. Furthermore, girders can have mechanical offsets and klystrons and power supplies can have excitation errors which are applied to all elements they support. This simplifies the bookkeeping associated with a complicated tree of misalignments and errors, and forces tuning algorithms to respect realistic limitations such as elements which are powered in series.

Because it is a common practice to represent a single physical element with several instances in a lattice (in the vernacular, to have "slices" of a magnet), Lucretia allows the user to indicate "slicing," which will then be respected by error-setting and strength-tuning functions. Similarly, a cluster of devices which are tightly coupled to one another (a "block") can be established, and blocks are respected by misalignment functions. These features allow the user to define an intermediate level of structure to the lattice between individual instances in the BEAMLINE array on the one hand and girders, klystrons, and power supplies on the other.

All Lucretia functions which operate on the accelerator data, including the Twiss propagation and tracking functions, take as arguments the range of elements, klystrons, etc., which are to be operated on by the function. This permits the user to store multiple disjoint beamlines in a single set of arrays, which in turn permits direct simulation of effects such as colliding beams which have passed through two separate Low Emittance Transports.

Beam Representation

Like the accelerator, the beam representation is resident in Matlab; any number of beams can be defined. The beam data structure is in turn composed of bunch data structures, and each bunch contains one or more dimensionless rays. Each ray has 6 coordinates (transverse positions and fractional momenta, arrival time relative to the reference particle times the speed of light, total momentum) and a charge. Each beam also includes an inter-bunch arrival time interval, and each bunch has an array to indicate whether each particle was stopped during tracking, and if so at what point in the beamline.

Lucretia only tracks dimensionless rays, since these are the more appropriate beam representation for systems which introduce high-order correlation into both the transverse and longitudinal degrees of freedom. Within this limitation, however, the user is free to define any combination of rays which suit their current purposes. For example, it is possible to emulate the LIAR beam, which is composed of macroparticles, by generating 9 rays with identical charges, momenta and arrival times but different transverse positions (to be precise, one ray at the centroid and the others at $\pm 2.12\ \sigma$ in each transverse degree of freedom).

ORGANIZATION OF THE SOURCE CODE

The current Lucretia distribution contains 71 functions, almost all of which are written in the Matlab scripting language. These functions are used to generate and configure the lattice, add misalignments and errors, perform tuning and analysis, and generate beam data structures. Although the Matlab interpreter is not fast compared to execution of compiled code, execution speed of these functions is acceptably high.

In tests it was found that execution of tracking was too slow when implemented as Matlab functions executed by the interpreter. For this reason, the tracking system was written in C and compiled into a dynamically linked function (known as a "mexfile"). The resulting function is executed from the Matlab command line in a manner and using a syntax identical to standard Matlab interpreted functions, but with execution speeds hundreds of times faster. Since calculation of R-matrices and Twiss function propagation share much of the tracking code, these functions were also written in C and compiled; for similar reasons, the lattice verification function (which ensures that the lattice is constructed without errors and in a self-consistent manner) is also a compiled, dynamically-linked function.

SUPPORTED ELEMENTS AND PHYSICS

The supported element classes in Lucretia include: drifts; dipole steering magnets; sector bends (with or without focusing gradient); quadrupoles, sextupoles, and thin-lens multipoles; linear acceleration RF structures; beam position monitors; markers; several classes of beamline instrument (generic instruments, profile monitors, wire scanners, toroids, current monitors, synchrotron light monitors, bunch length monitors); collimators; and marker points. These elements support the standard suite of beam dynamics effects (steering, bending, linear and second-order dispersion, linear and second-order longitudinal motion, chromatic focusing, sinusoidal linear acceleration, nonlinear focusing in multipoles, collimation by perfectly opaque objects). RF structures also support single bunch longitudinal and transverse wakefields, as well as multi-bunch transverse wakefields.

TRACKING

Lucretia's tracking function takes as arguments the starting and ending elements for tracking, the beam which is to be tracked, and the index numbers of the first and last bunches which are to be tracked. The user can track a single element, an entire beamline, or any subsection of the beamline. As described above, this flexibility facilitates simulation of colliding-beam facilities while allowing all

beamlines to be stored in a single set of Matlab data structures.

In addition, the user can specify whether multi-bunch tracking is performed "element-wise" (in which all bunches are tracked through an element before any bunches are tracked through the next element) or "bunch-wise" (in which each bunch is tracked through all the elements before the next bunch is tracked through any elements). The former mode is the default, since it is faster and more memory-efficient. The latter mode permits simulations in which feedbacks measure and correct beam properties within a single bunch train: the first bunch is tracked through the accelerator, and the resulting instrument data is collected; changes are made to accelerator settings; the second bunch is then tracked through the modified accelerator; etc.

BEAM INSTRUMENTATION

During tracking, the user can select any subset of the beam position monitors or other instruments for which beam data will be returned. The data is returned in a Matlab structure array which is a return argument to the tracking function. Selection or elimination of instrument data is done via tracking flags embedded in each element's entry in the BEAMLINE array.

The most common beam instrumentation is beam position monitors (BPMs). Each BPM can be set to return no information, to return physically realistic information only (position readings with resolution limits, position offsets, and electrical offsets taken into account), or to return both realistic and unrealistic information (unrealistic information includes the centroid momentum, the matrix of beam second moments, and the angle of the beam trajectory with respect to the BPM). The BPM can be selected to return information for each tracked bunch individually, or else for all of the bunches in a train in aggregate.

Other beam instruments such as profile monitors can be configured to return no information or else to return a standard set of values including the beam's transverse and longitudinal position, bunch length, three transverse second moments ($< xx >$, $< xy >$, and $< yy >$), and charge. As with the BPMs, this information can be returned bunch-by-bunch or in the aggregate in the case of multibunch tracking.

Finally, the RF structures can be configured to return beam position signals based on reconstructing the power and phase of excited dipole modes. An RF structure can have any number of HOM-BPMs, uniformly spaced longitudinally in the structure. As with conventional BPMs, the HOM-BPMs return data with resolution and static offsets taken into account. HOM-BPMs always return aggregated data for a bunch train, not data for individual bunches.

PLATFORMS AND SUPPORT

Lucretia is available for Windows XP, Solaris, and Linux computers, and runs under Matlab 7. Source code, executables, examples, and documentation can all be found at the Lucretia website,

`http://www.slac.stanford.edu/accel/ilc/codes/Lucretia/`

In principle, Lucretia can be ported to any platform for which Matlab 7 is available.

FUTURE DEVELOPMENTS

In future releases of Lucretia, we anticipate adding the following features:

- A combined function solenoid-dipole-quadrupole (SDQ) element, which permits simulation of complicated IR geometries including detector solenoids
- Incoherent synchrotron radiation
- Arbitrary changes in the nominal survey line of the accelerator
- An interface to the beam-beam collision simulation code GUINEA-PIG [5]
- An implementation of Seryi's ground motion model for accelerators [6]
- Improved simulation of particles which are not fully relativistic, taking into account effects at the order of $m_e c^2/p^2$.

In addition, the Lucretia C functions have been structured in such a way as to facilitate adaptation into an operating environment other than Matlab. One likely candidate is the open-source mathematics package Octave, which also supports dynamically-loaded compiled functions.

ACKNOWLEDGEMENTS

The work described in this paper could not have been accomplished without the assistance, advice, and support of T. Raubenheimer, A. Terebilo, G. White, A. Wolski, and M. Woodley.

REFERENCES

[1] R. Assmann *et al*, "LIAR: A Computer Program for the Modeling and Simulation of High Performance Linacs" (1997).

[2] P. Tenenbaum *et al*, "Recent Developments in the LIAR Simulation Code," (2002).

[3] *International Linear Collider Technical Review Committee Second Report* (2003).

[4] A. Terebilo, "Accelerator Modeling with Matlab Accelerator Toolbox," (2001).

[5] D. Schulte, "Beam-beam Simulations with GUINEA-PIG" (1998).

[6] A. Seryi *et al*, "Effects of Dynamic Misalignments and Feedback Performance on Luminosity Stability in Linear Colliders" (2003).

