RECENT IMPROVEMENTS TO CHEF, A FRAMEWORK FOR ACCELERATOR COMPUTATIONS *

J. -F. Ostiguy and L. P. Michelotti
Fermi National Laboratory, Batavia, IL 60510, USA

Abstract

CHEF is body of software dedicated to accelerator related computations. It consists of a hierarchical set of libraries and a stand-alone application based on the latter. The implementation language is C++; the code makes extensive use of templates and modern idioms such as iterators, smart pointers and generalized function objects. CHEF has been described in a few contributions at previous conferences. In this paper, we provide an overview and discuss recent improvements.

INTRODUCTION

Formally, CHEF refers to two distinct but related things: (1) a set of class libraries (2) a stand-alone application based on these libraries. The application makes use of and exposes a subset of the capabilities provided by the libraries.

CHEF has its ancestry in efforts started in the early nineties. At that time, A. Dragt, E. Forest [2] and others showed that ring dynamics can be formulated in a way that puts maps rather than Hamiltonians, into a central role. Automatic differentiation (AD) techniques, which were just coming of age, were a natural fit in a context where maps are represented by their Taylor approximations.

The initial vision, which CHEF carried over, was to develop a code that (1) concurrently supports conventional tracking, linear and non-linear map-based techniques (2) avoids "hardwired" approximations that are not under user control (3) provides building blocks for applications.

C++ was adopted as the implementation language because of its comprehensive support for operator overloading and the equal status it confers to built-in and user-defined data types.

It should be mentioned that acceptance of AD techniques in accelerator science owes much to the pioneering work of Berz [1] who implemented –in fortran – the first production quality AD engine (the foundation for the code COSY). Nowadays other engines are available, but few are native C++ implementations. Although AD engines and map based techniques are making their way into more traditional codes e.g. [5], it is also probably fair to say that map-oriented codes are still perceived as a specialized niche.

* Work performed under Fermi Research Alliance Contract DE-AC02-07CH11359 with the United States Department of Energy.

AUTOMATIC DIFFERENTIATION

Analytic nonlinear maps can be approximated locally by Taylor series expansions in phase space coordinates. The transfer map through an element or sequence of elements can be represented, to some arbitrary order k, by a finite set of numbers: the numerical values of the map derivatives. AD techniques allow convenient and efficient computation and manipulation of such representations. For example, consider the numerical values of a function f and its derivatives up to order k at some at reference point $\mathbf{x_0}$. If a composition $h = g \circ f$ is formed, then all those quantities can be calculated exactly for h. The rules for carrying this out are the same for functions of either real or complex variables.

To implement this in an object-oriented language, a new data type is created by aggregating the values of a function and its derivatives at $\mathbf{x}_0$, up to some order. Basic operations on the new type are then defined: e.g. the rules for multiplication are a generalization of the familiar Liebnitz's rule. Note that the algebra of numerical derivatives is the same as that of polynomials (truncated to the same order).

In the context of CHEF, functionality related to automatic differentiation is contained in the mxyzptlk library. The library makes no reference to accelerator concepts and can be used independently. Suggestive of a closely related mathematical concept, the primary data type is called Jet. Through judicious use of templates and implicit conversions, both real and complex Jets are supported and can be mixed. The Jet class and its associated functionality are used to construct other classes corresponding to Hamiltonian perturbation theory concepts i.e. Mapping, LieOperator etc.

A Taylor polynomial in m variables of order k has $\frac{(m+k)!}{m!k!}$ terms. Even at modest order, this is not a small number. Attention to efficiency and memory management is key to a successful implementation. Accordingly, the internals of mxyzptlk have been carefully optimized. Extensive use of templates and modern C++ idioms result in compact code not obscured by the mechanics of memory management.

A Jet is implemented as a contiguous sequence of "terms" each comprised of a coefficient and a pointer into an ordered table whose entries are monomial exponent tuples. The representation is *sparse*, that is, null terms are not stored. Jet multiplication, which typically dominates cpu usage, is implemented by accumulation of monomial products in a reserved area containing entries for all possible monomials. Table lookup is used to determine monomial

product indices. Specialized memory allocation and recycling strategies are employed for terms.

By design, the number of variables and the reference point are not hardwired, but rather encapsulated in a `JetEnvironment` object specified at runtime. Multiple environments can exist within the same program.

As a rough indication of performance, a 6th order map for a ring comprising a few hundred elements can be computed in a few minutes on a typical desktop machine. At first order, where a `Jet` carries the same information as traditional transfer matrices, the performance penalty compared with the latter is minimal.

BEAMLINES

Beamlines are naturally described hierarchically. Internally, `CHEF` models a beamline as a container of smart pointers to element instances. A `beamline` inherits from the base element type, allowing beamlines to contain other beamlines, recursively, to arbitrary depth. From a computer science viewpoint, a beamline is a single-rooted tree structure. In order to take advantage of a well-documented and predictable interface, beamline elements are accessed using a collection of iterators modeled after the STL iterators. Single level forward and reverse iteration as well as depth-first and breath-first hierarchical traversal are supported. Facilities are provided to select elements, edit or transform beamlines in a variety of ways. Of interest is that selected elements may be subdivided longitudinally at user-specified intervals. One application for this is the introduction of localized space charge kicks. `CHEF` allows individual elements to be displaced and rotated. One distinctive feature is that functionality is provided to handle either small perturbations or general geometry changes. In both cases, the code determines appropriate local coordinate transformations. While in the former case, the transformation depends only on the targeted element. in the latter, it also depends on its nearest neighbors.

A capability to determine particle loss caused by finite size apertures has recently been implemented. Two types of apertures are supported (elliptical and rectangular). To minimize overhead when concrete apertures can be ignored, apertures are implemented as a decorator for `Propagator` objects which are described below. `CHEF` currently provides stable and fairly complete support for the standard MAD8 and XSIF formats, including symbolic expressions, macros, and arbitrary level file inclusion. A parser for the MADX format is under development and should be available soon. The parsers are implemented as fully reentrant GLR parsers using `bison` and `flex`.

PARTICLES AND BUNCHES

`CHEF` was initially focused on proton dynamics. Later, the code base started being used to model electron linacs and this motivated a more consistent and general internal treatment of particles and bunches in order to include wakefield effects. Specifically, `CHEF` provides two abstract base types `Particle` and `JetParticle` from which all concrete particle types such as `Proton` or `JetElectron` are derived. Each particle type has a phase space state attribute; the basic difference between a Particle and a JetParticle is that the state of the latter is an array of `Jets` rather than scalars. All propagation algorithms are consistently templated on a parameter which can be `Particle` or `JetParticle`. In this manner, the exact same same code is used both for particle tracking and map generation without runtime penalty. One may also propagate `ParticleBunch` and `JetParticleBunch` which are, as their name suggest, containers for particles. Both types of containers are endowed with STL-style iterators. Additional functionality is provided to populate bunches according to common statistical distributions, to compute projections, moments, and so forth.

PROPAGATION PHYSICS

`CHEF` strives to separate the details of propagation physics from beamline element description that is, to make propagation physics a runtime choice. For each element, propagation physics is handled, in local coordinates, by a procedure encapsulated into a `Propagator` object. A default set of `Propagators` is provided: exact for dipoles and drifts; based on sequences of thin kicks for others. In some instances, it can be advantageous to treat specific elements differently by assigning custom propagators to them (e.g. a pre-computed map). Alternatively, a different set of default propagators may be used in order to introduce domain specific approximations.

ANALYSIS

As expected, `CHEF` provides facilities to compute optical lattice functions. Both standard (uncoupled) and generalized (coupled) lattice functions may be computed for either rings or beamlines. Although the map based techniques do not offer special advantage over traditional matrix codes for basic linear optics, the fact that high order maps are available and can be operated on makes it possible to generalize computations of quantities such as dispersion, chromaticities or momentum compaction beyond first order. `LieOperator` objects (provided by `mxyzptlk`) are used to implement the normal form procedures developed by Forest and Irwin [2]. Fig. 1 shows one example: orbits in longitudinal phase space are first displayed in standard (ϕ, E) coordinates and subsequently in normal form coordinates, transformed to three different orders. As the order increases, the range over which orbits look circular becomes larger until it effectively fills the central stable region. Outside the separatrix, orbits become distorted but the transformations are not intended to be applied in this region.

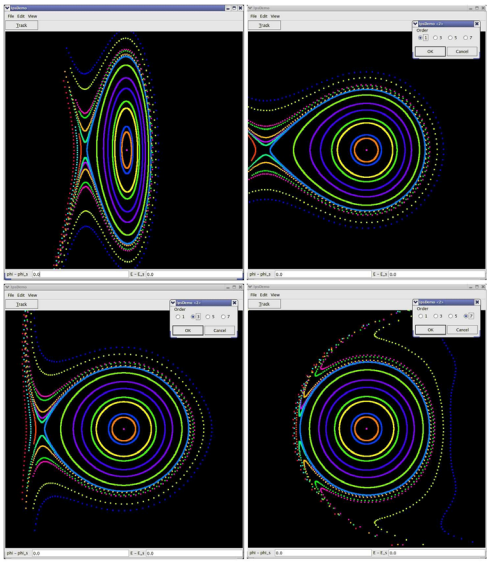

Figure 1: Longitudinal phase space orbits displayed in standard and normal form coordinates at orders 1,3 and 7.

Embedded Database

One recurring practical issue with accelerator codes is the bookkeeping involved in storing and keeping track of computed quantities for future reference, comparison purposes or as inputs to other computations. For that purpose, we have found it advantageous to take advantage of `SQLite` [3], a lightweight and fast embedded SQL database engine. To the extent that the database metatada describes organization and data types, the data sets are automatically self-describing. Databases can be saved in a well documented binary format. A collection of readily available tools can be used to query, sort, browse and database contents.

`PYTHON` BINDINGS

Many scientific applications are now structured as a high level "steering" or "control" module, written in an interpreted language with computationally intensive functionality implemented in a traditional compiled language and made available through a dedicated binding layer. The main advantage of this approach is that most applications involve a substantial amount of logic and bookkeeping that can be implemented, modified and debugged more quickly in an interpreted language. With that in mind, a fairly comprehensive set of python bindings has been developed, to make the `CHEF` libraries functionality accessible from `python` scripts. As a scripting language, `Python` represents a good match for `C++`: it has a compatible notion of class and it also provides support for operator overloading and the concept of iterator. `Python` is well-supported within the scientific community. `Synergia` [6], a parallel code used to model beam dynamics in presence of 3D space-charge fields, imports functionality from `CHEF` through its `python` interface.

APPLICATION TO LINACS

`CHEF` has been adapted for high energy linacs, and in particular, has been used to study emittance preservation issues. While the map machinery is a less compelling feature when dealing with single pass machines, `CHEF` provides needed generic infrastructure and is quite capable as a conventional tracking tool. To handle linac simulations, an accelerating structure element, a capability to handle wakefields and a facility to establish reference trajectories in the presence of acceleration have been added. Detailed comparisons with linac specific codes were performed. Small, but nevertheless significant discrepancies were observed and mostly traced to inconsistent conventions and definitions. The issues were progressively understood, and excellent agreement was confirmed. As an example, Fig. 2 shows a comparison, for a version of the International Linear Collider lattice, of the vertical emittance evolution predicted by `CHEF` to that of predicted by `Lucretia` [4], a linac code developed at SLAC. In this comparison, all quadrupoles and accelerating structures have been subjected to identical sets of random misalignments. Transverse and longitudinal wakefields are included. Vertical corrector excitations are set to values determined by an emittance growth minimization algorithm.

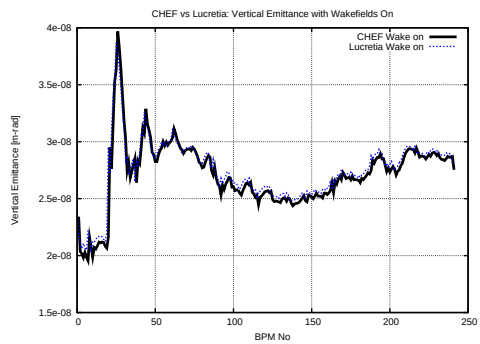

Figure 2: Comparison of predicted vertical emittance evolution between `CHEF` and Lucretia.

REFERENCES

[1] M. Berz, Particle Accelerators 24 (1989) 109-124

[2] E. Forest, "Beam Dynamics, A new Attitude and Framework", Harwood Academic Publishers, 1998 and the references therein.

[3] `http://www.sqlite.org`.

[4] P. Tenenbaum, *Lucretia: A Matlab-Based Toolbox for the Modeling and Simulation of Single-Pass Electron Beam Transport Systems*, PAC 2005, Knoxville, TN

[5] P. K. Skowronski, F. Schmidt, E. Forest, *Advances in MAD-X Using PTC*, PAC 2007, Albuquerque, NM

[6] J. Amundson et al., *Synergia: A 3D Accelerator Modelling Tool with 3D Space Charge*, J. Comp. Phys., 211, 11 (2006), pp 229-248.