CHEF: A Framework for Accelerator Optics and Simulation

Jean-Francois Ostiguy*
Leo Michelotti
Fermilab

Historical Background

- Around 1990, L. Michelotti initiated development of a suite of libraries dedicated to accelerator simulation, with a focus on nonlinear dynamics. The libraries would take advantage of Automatic Differentiation, a then emerging technique (AD was pioneered by M. Berz, especially in the context of Accelerators)

(a) C++ was chosen as implementation language because of its comprehensive support for operator overloading.

- **Vision: a framework allowing one to construct applications treating "scalar" and high order computations on the same footing.**

- **In simple terms: a formal substitution of the type `JetParticle` for the type `Particle` results in high order derivatives computed automatically during tracking. First order tracking = traditional linear optics.At least that is the propaganda … ;-)**

Library Hierarchy

components — High level Components
- Site viewer
- Beamline browser
- Phase space display etc …

physics_toolkit — Tools for calculating lattice functions, moment propagation, normal form analysis etc …

bmlfactory — LEX-YACC based parsing. Supports MAD8 format.

Beamline — Beamline creation, manipulation and edition. Tracking.

Mxyzptlk — Automatic differentiation/differential algebra engine for perturbation theory and tracking.

basic_toolkit — Utility functions, Linear Algebra, Memory management, Physical Constants

A new project: CHEF

Started in mid-2003, with the following goals:

- Convenient, intuitive, interactive.
  General purpose **optics** computations - high priority
- General framework applicable to problems relevant to future machines.
- preserve original vision of a common infrastructure for both linear and nonlinear analysis
- reuse existing code base as much as possible
- runs on std platforms (win32,linux)

Reality Check

The original design of the DA engine had flaws that became to painful to ignore:

- 1st order (optics) computations slower than with a conventional, matrix-based code

- Generally, (mxyzptlk) efficiency not competitive with that of other DA engines (e.g. COSY)

Issues

- DA objects (`Jet`) have a large memory footprint,even when sparsity is taken advantage of.
- Copying such objects is **expensive** !
- Overloaded operations generate temporary copies, some necessary, many **not**
- *Strategy: make the computational cost of multiple copies trivial;eliminate unnecessary copies and optimize the cost of necessary allocations.*

- searching (finding a specific monomial within a sparse set) is expensive !

Strategy: Use algorithms that no not require search !

Overhaul of the DA Engine (mxyzptlk)

•dlist representation previously used for polynomials eliminated
•sparse representation preserved
•new algorithms; basic ops (add, mult) now using based on "register" or "scratchpad" (no search !)
•more compact,efficient monomial indexing scheme
•extensive use of templates, smart ptrs and specialized memory management techniques to address the temporaries issue.
•transparent and further optimized 1st order mode

Current Status:

•first order computations with DA engine now compares favorably with conventional matrix-based codes
•higher order computations efficiency competitive with other DA engines

Some Mxyzptlk (DA Engine) features

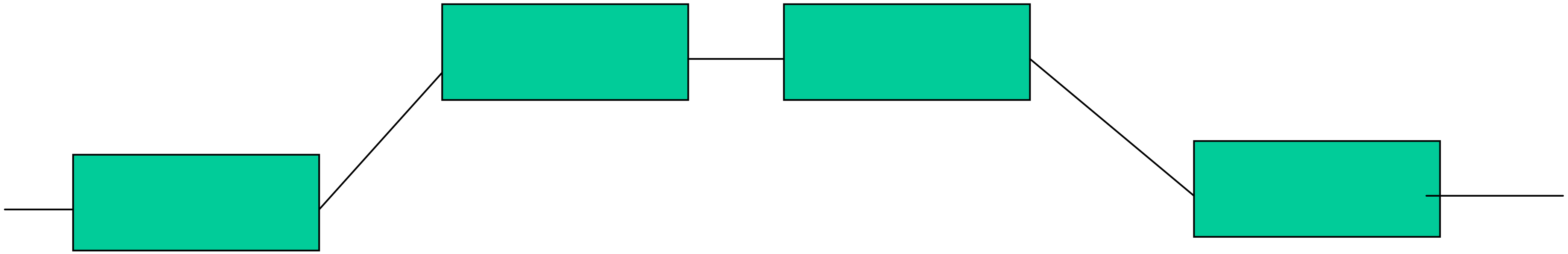
- Order, no of variables, and ref point can be changed dynamically
- Mixed complex and double (Jet) arithmetic (automatic implicit conversion)
- Decoupled interface and implementation
- templated code
- Minimal, fully dynamic memory footprint

CHEF – Some Current Capabilities

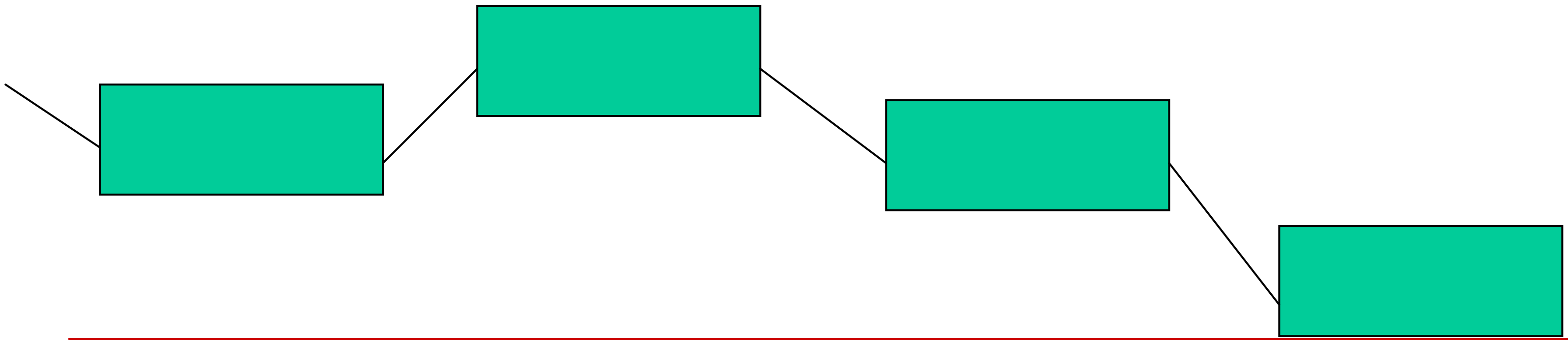
- Coupled/Uncoupled optics. **No** implicit assumption about reference orbit, not even when positioning them.
- Misalignments can be arbitrary in 3D space; no implicit assumption about magnitude.
- tracking with physics fully decoupled from geometry and user overridable (n=0 conventional tracking, n=1, linear optics). Single particle or bunches (distributions)
- Nonlinear maps , normal form analysis
- fully python scriptable (most lib public interface exposed)
- MAD8 lattice format
- database connectivity (survey data, multipoles)
- Runs **natively** on Windows, Linux, Solaris and (soon on Mac/OS-X) Interface based on Qt; **no** platform specific code in the underlying libraries.

Element Positioning: 4-magnet chicane

What we want -



What we get (e.g. MAD) ???

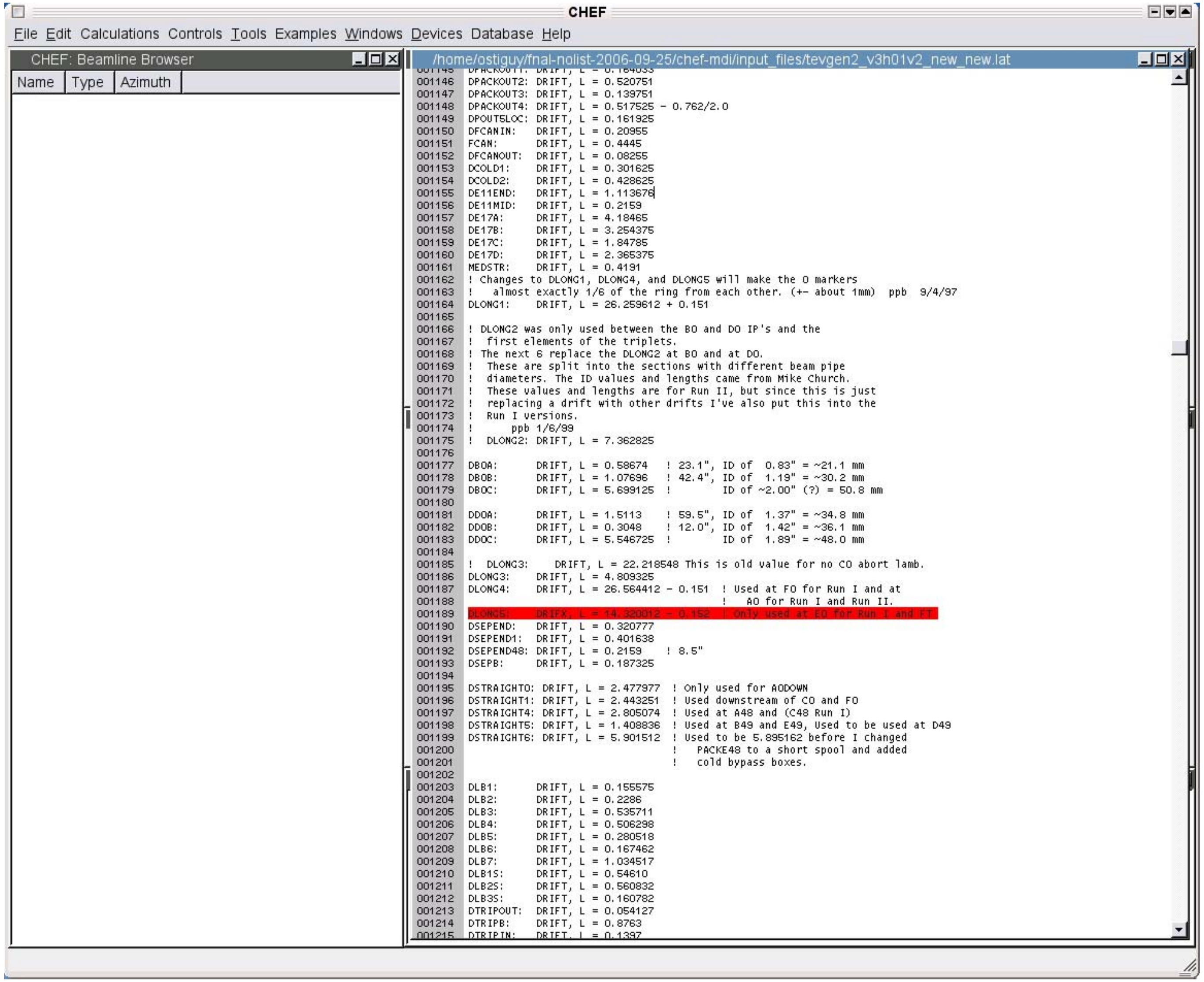


Python Bindings

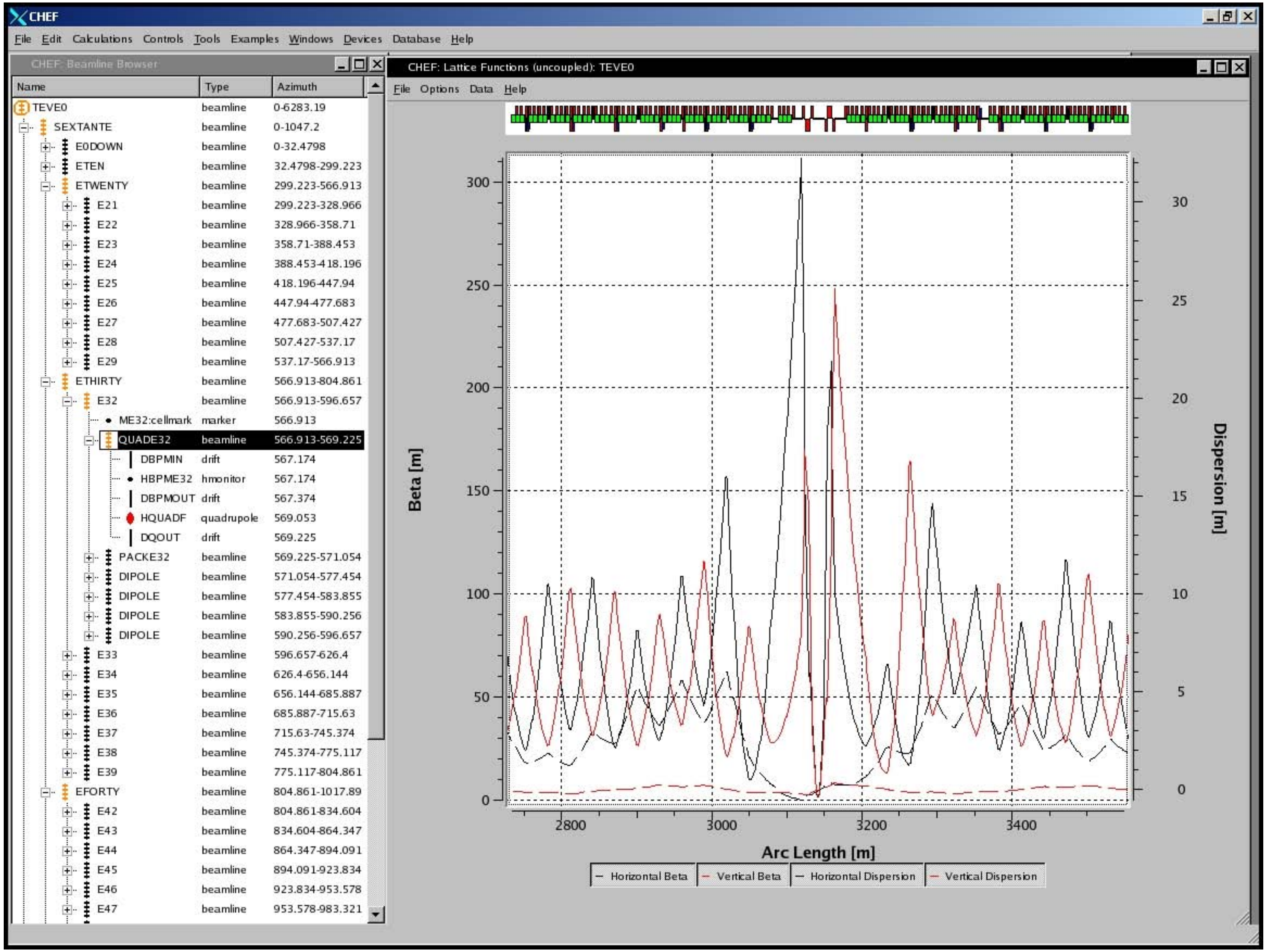
- Python is an excellent “impedance match” for c++. Operator overloading,inheritance relations have direct equivalent and can be preserved
- Bindings based on boost.python. No special binding generator is used; just a c++ compiler + boost.python header files. Exported interface specified in c++, in a declarative style reminiscent of IDL.
- A “mapped” Python object can override an exposed c++ method
- C++ objects can invoke a python method

Screenshots

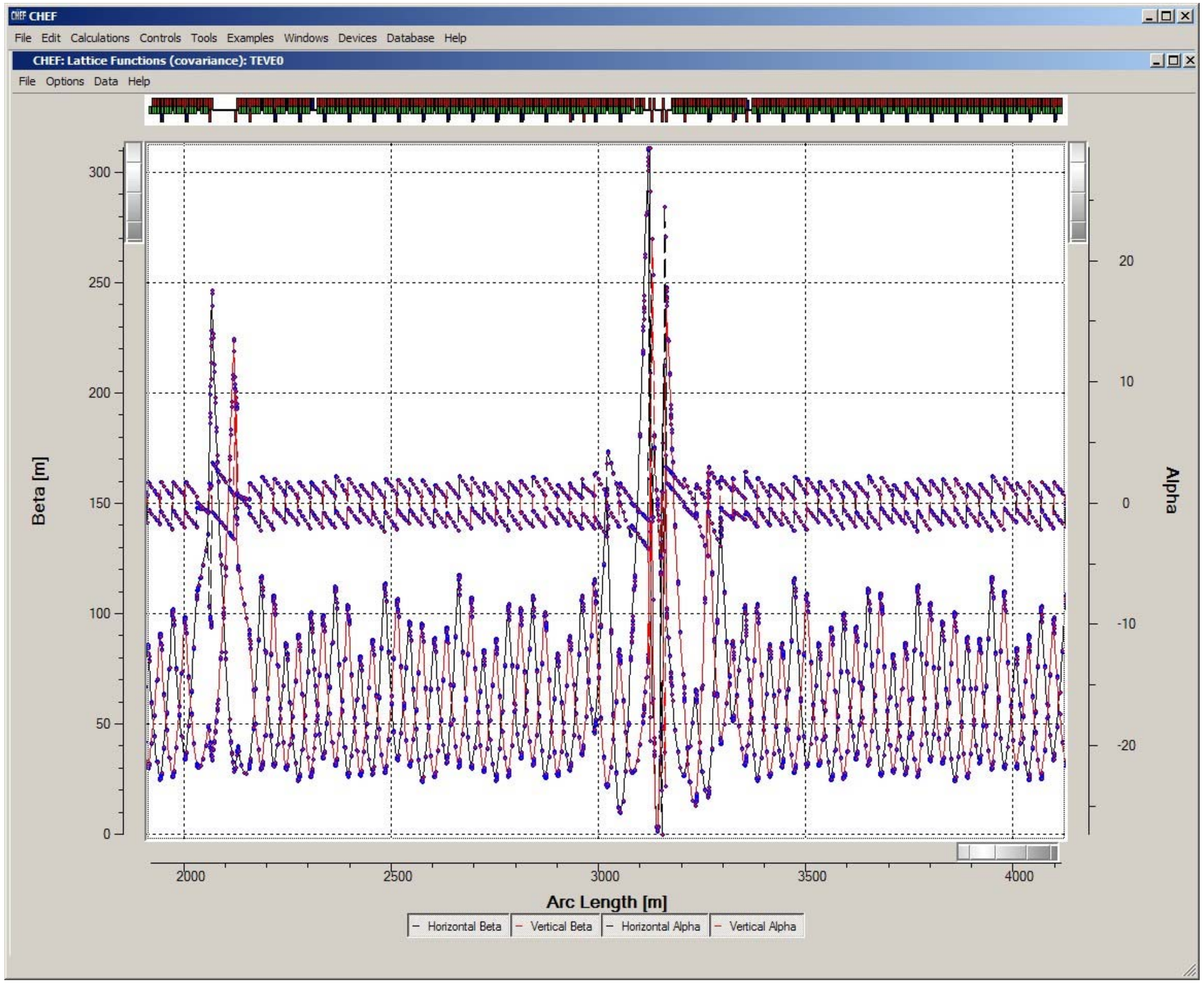
Interactive Parser



Beamline Browser



Optical Functions

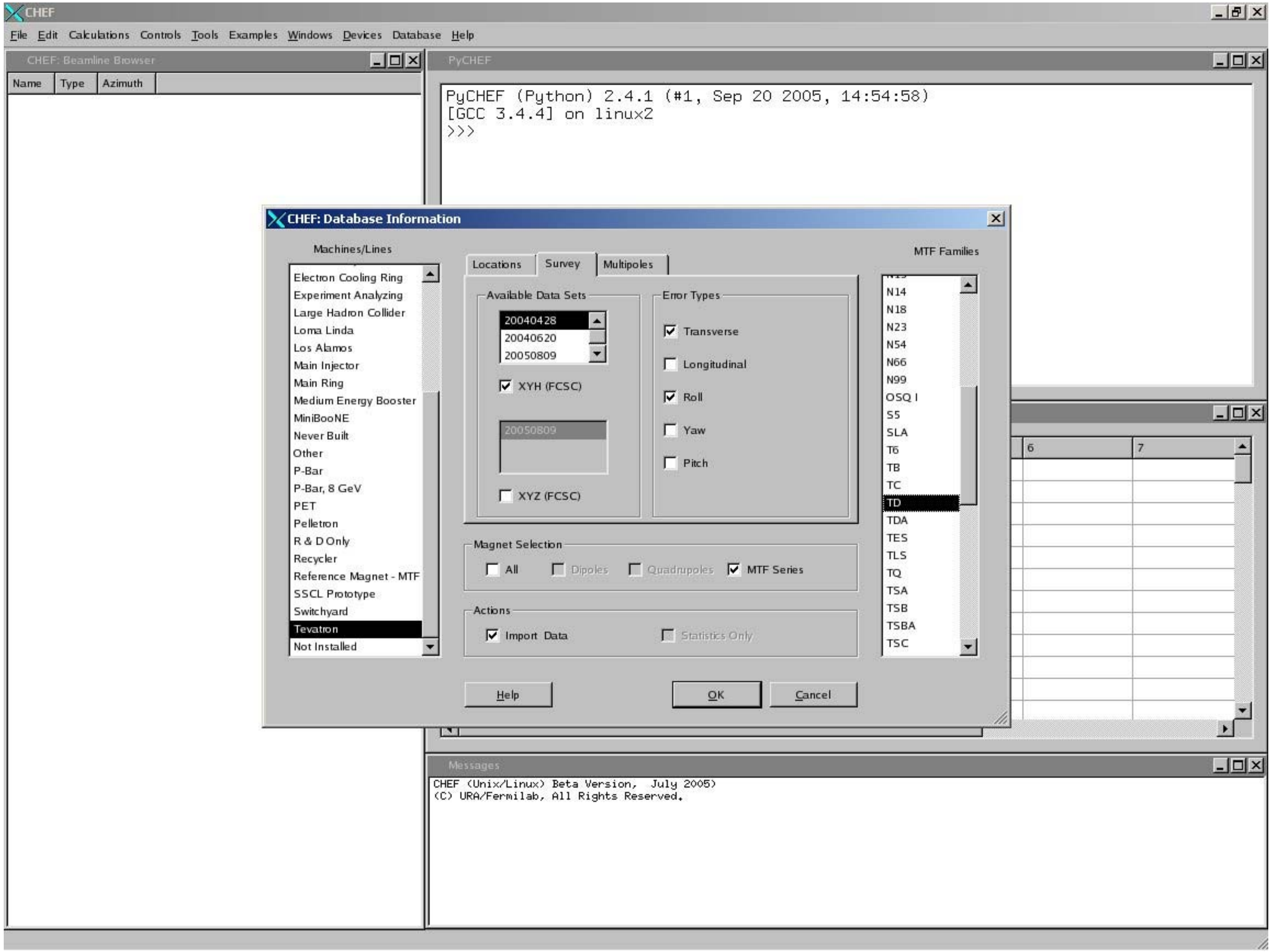


- Traditional (Twiss)
- Covariance
- Edwards-Teng
- Eigenfunctions

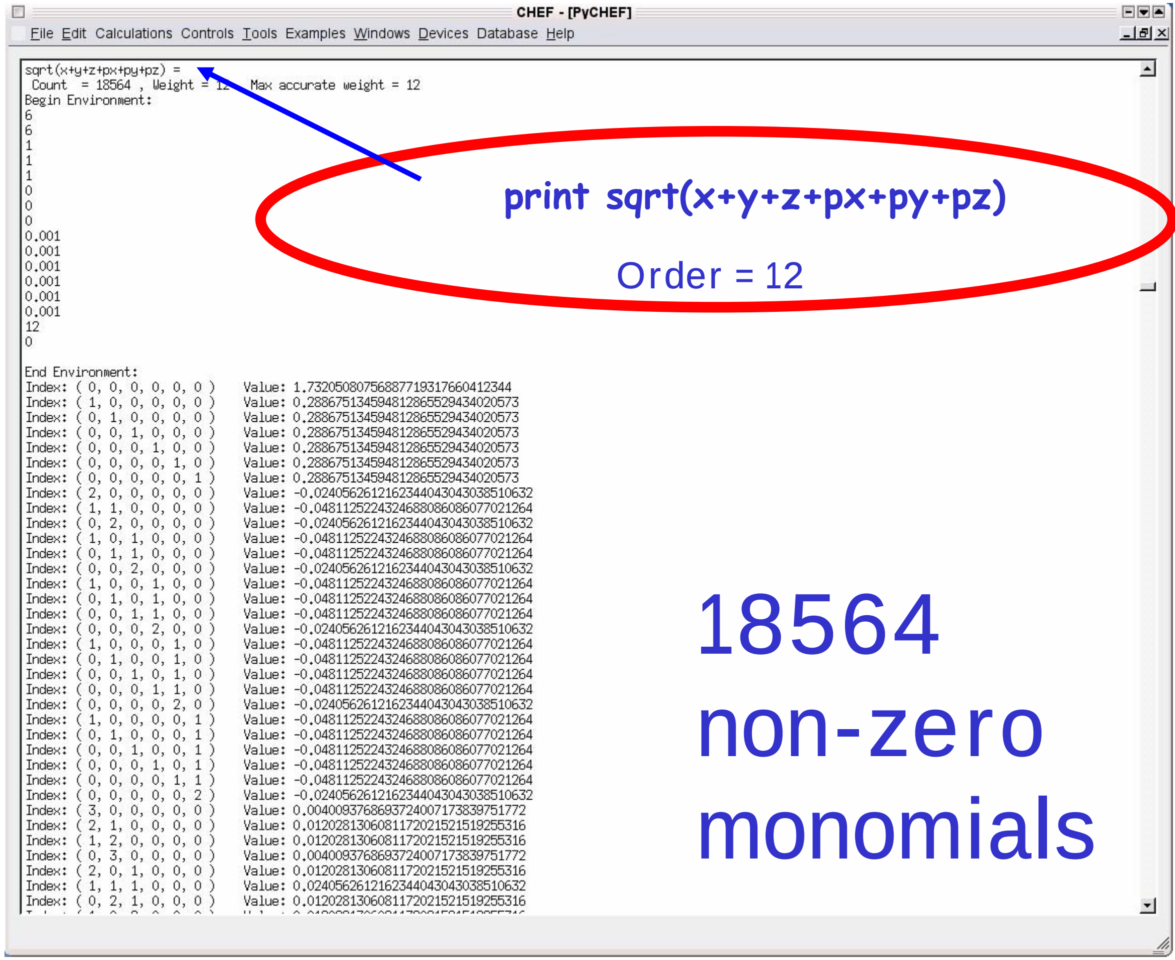
- Rings & Lines

- Closed orbit
- dispersion

Database Connectivity



Scripting: Automatic Differentiation



Declare:

X = coord()

y = coord()

....

x,y bona fide python objects.

Overloaded python operators and basic math functions (e.g. sin(),exp() ..)

Python Scripting – Tracking Demo

CHEF - [/home/ostiguy/fnal-nov-29/chef-mdi/input_files/tracking.py]

File Edit Calculations Controls Tools Examples Windows Devices Database Help

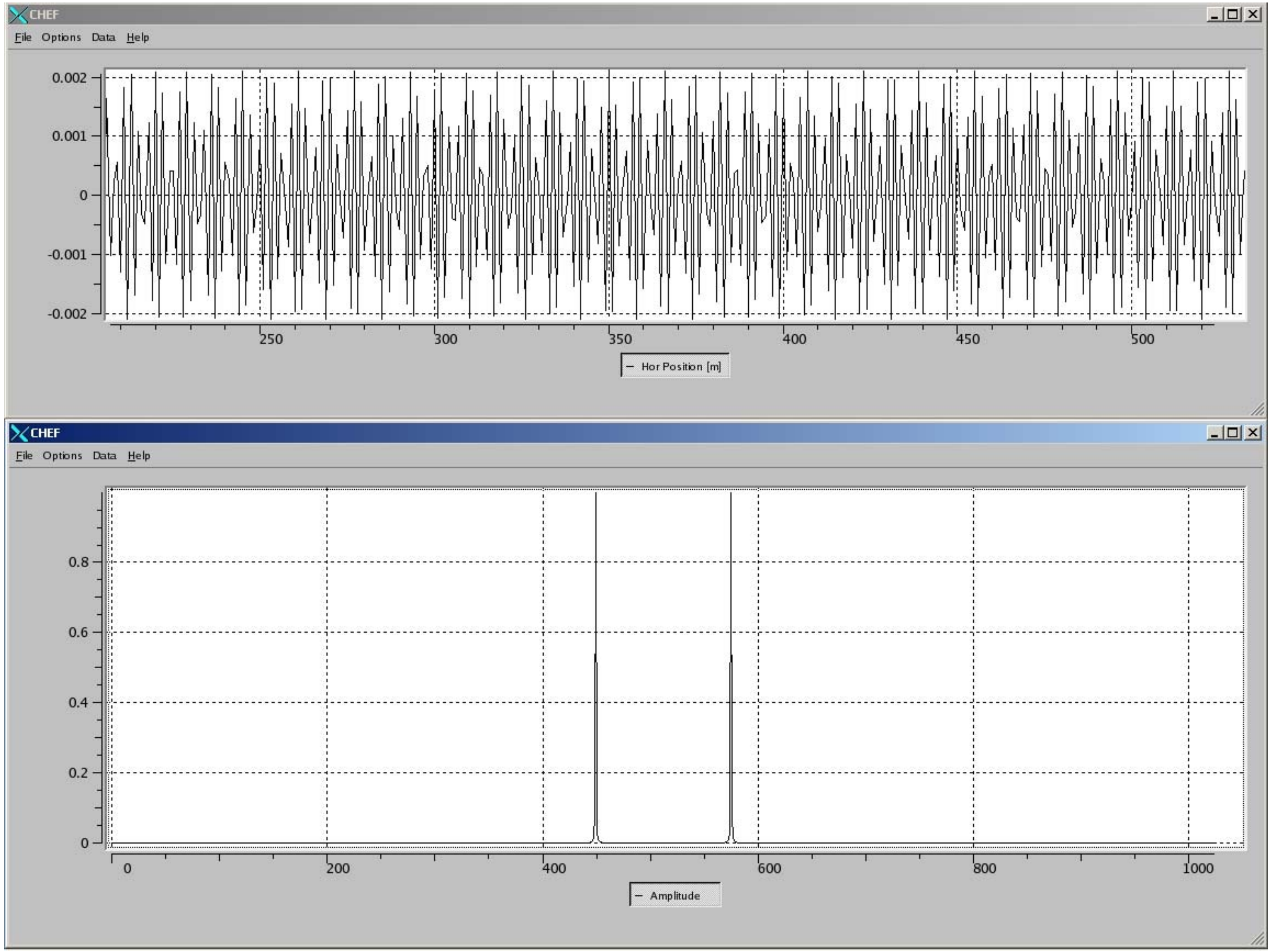
```
bfact    = bmlfactory.bmlfactory("Tevatron-E.lat")

#
# instantiate a beamline
#

pr = beamline.Proton(energy)

print "Reference BRho = ", pr.ReferenceBRho()

tevatron = bfact.create_beamline("normal_tevatron", pr.ReferenceBRho() )

#set initial conditions
pr.set_x(0.001)
pr.set_y (0.0)
pr.set_cdt(0.0)

#setup an array with space for nsamples turns

nsamples = 1024

x_position = numarray.zeros(nsamples, 'Float64')
turnno     = numarray.zeros(nsamples, 'Float64')

# iterate over nsamples turns

# propagate the Proton ( i.e. compute a one-turn map)

for i in xrange(nsamples):
    tevatron.propagateParticle(pr)
    x_position[i] =  pr.get_x()
    turnno[i] = i

#----------------------------------------
# plot and display the position
#----------------------------------------

# setup curves ...

x_curve  = chefplot.CHEFCurve(turnno, x_position,"Hor Position [m]")

# package data into a CHEFPlotData object

data = chefplot.CHEFPlotData()
data.addCurve(x_curve)

# pass the data to a plot widget and display the widget !

plot = chefplot.CHEFPlotMain()
plot.addData(data)
plot.resize(1000,500)
plot.show()
```

Read lattice file and instantiate beamline

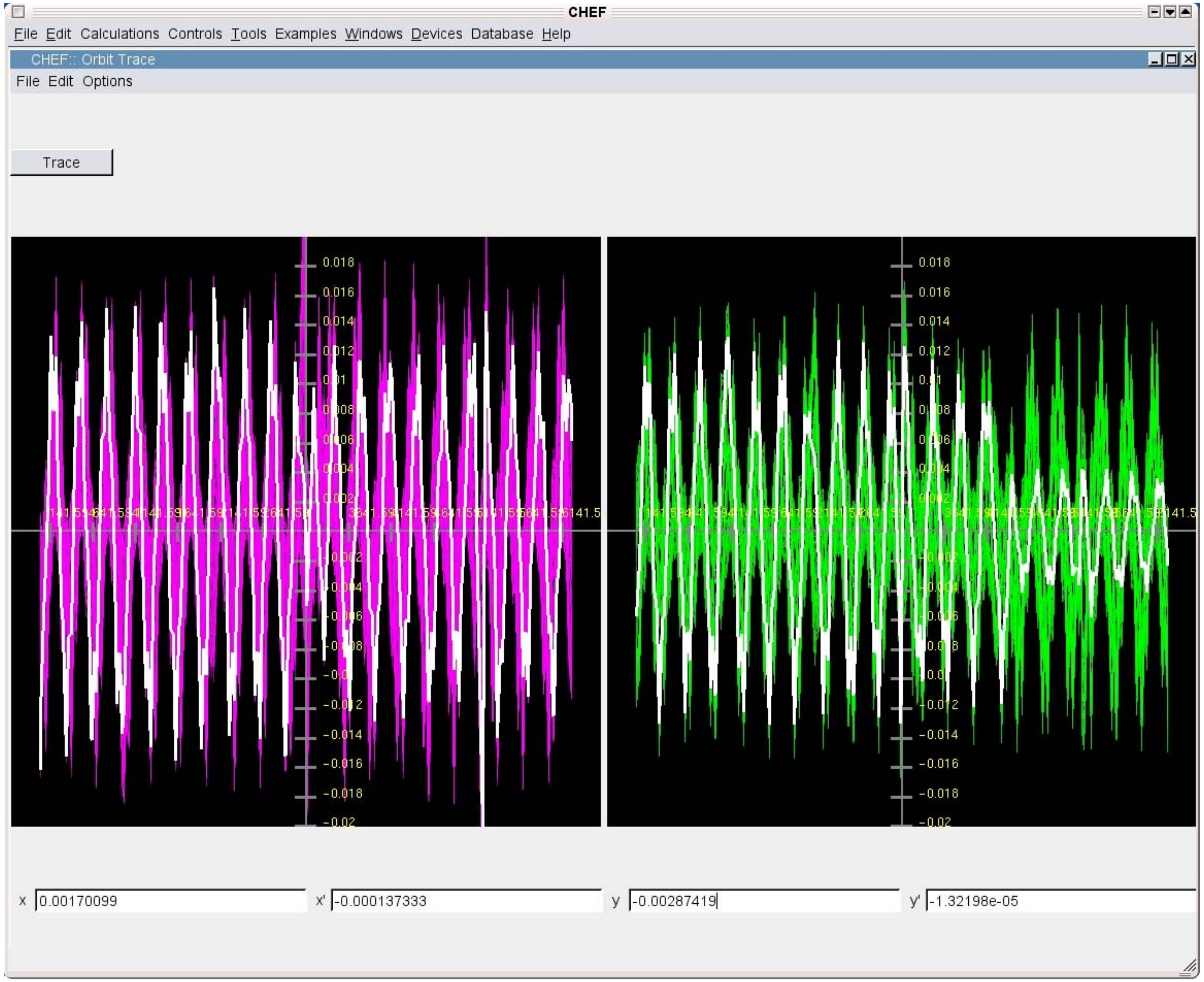
Use numarray extension (linear algebra, signal processing)

Propagate and get and save position

Tracking Demo - Results

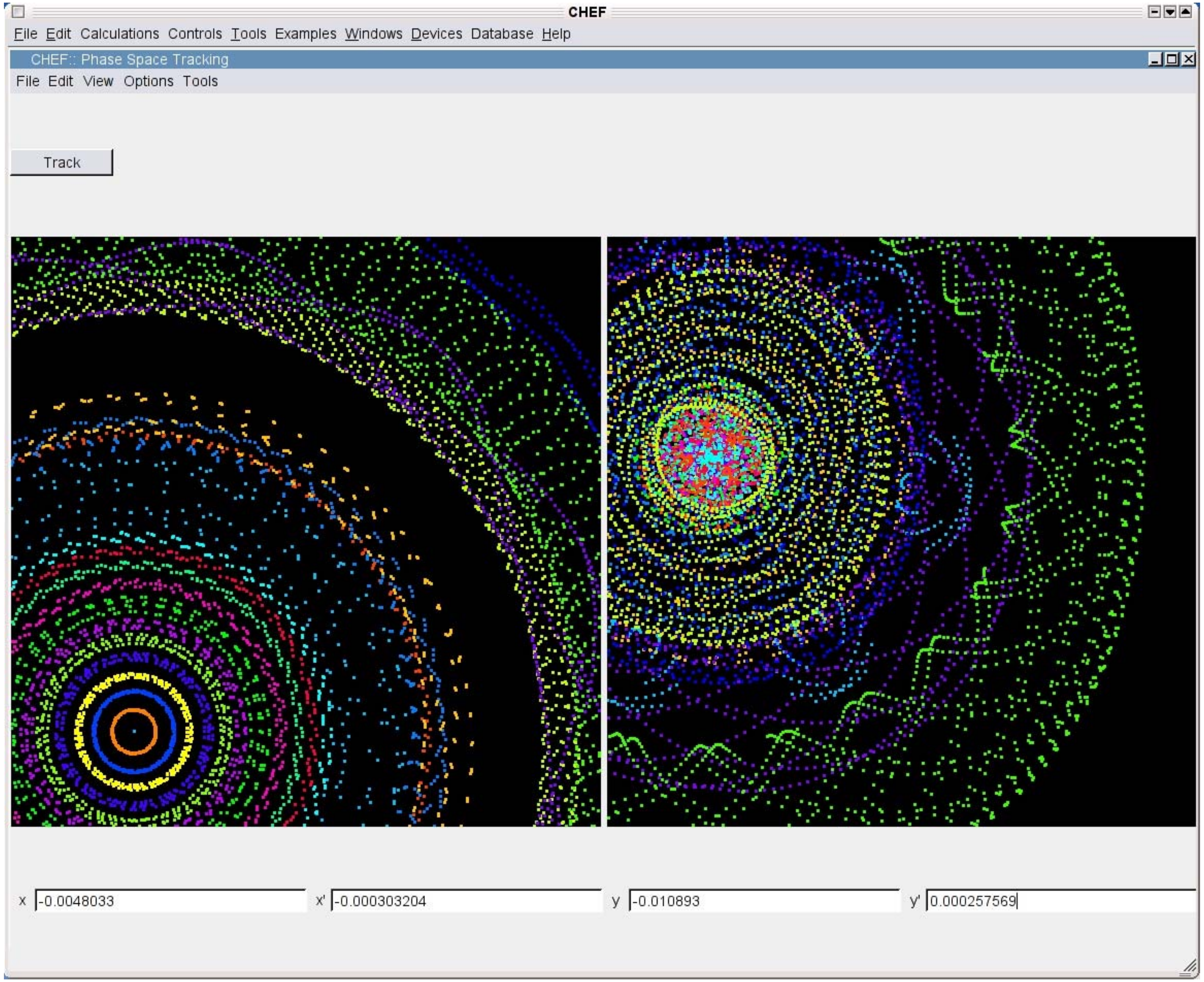


Trajectory Tracer



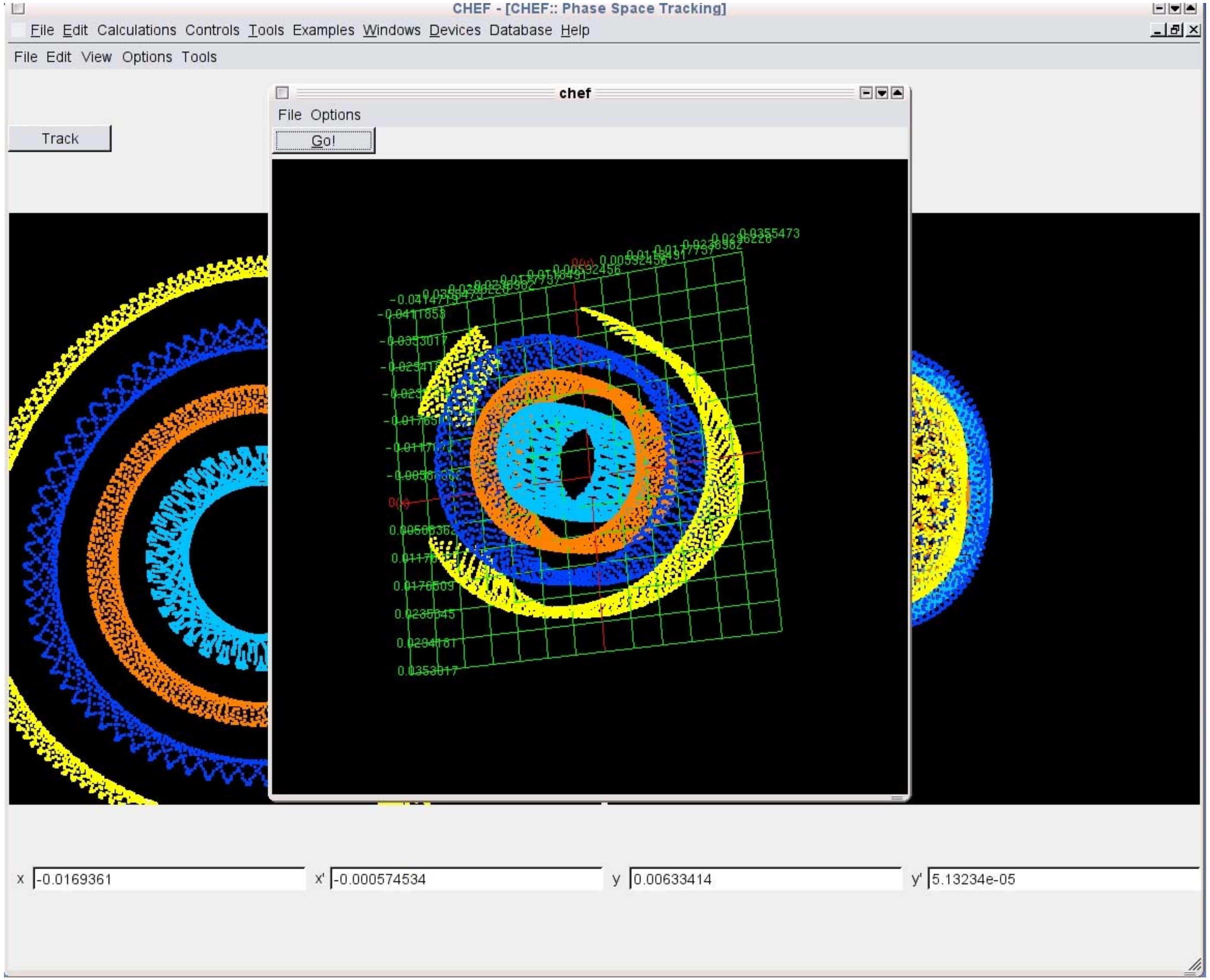
- Persistent display
- OpenGL rendering

Phase Space Tracker

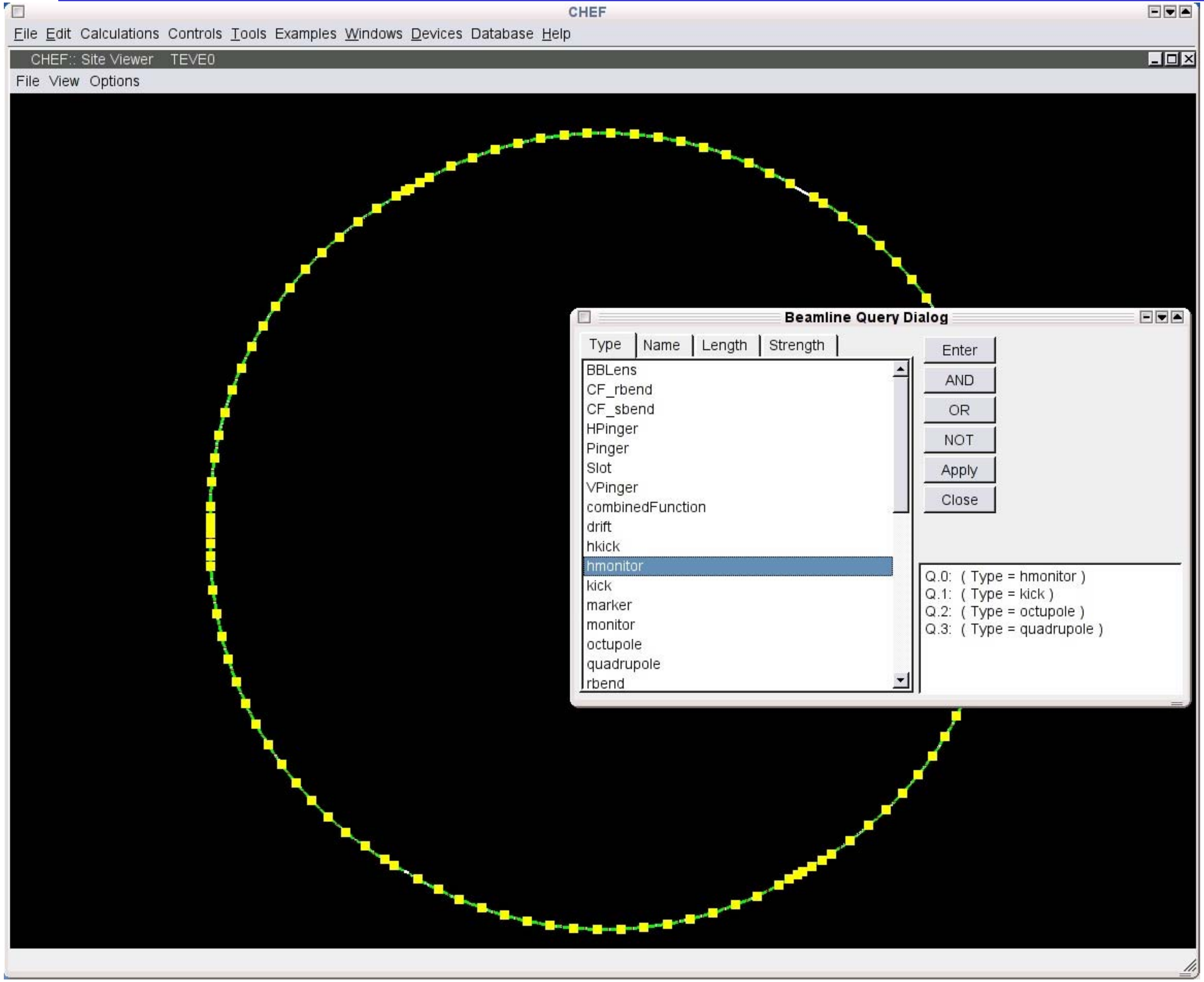


- Persistent display
- OpenGL rendering

Phase SpaceTracker - 3D Phase Space Projection



Site Viewer



- OpenGL rendering
- Search capability
- floor coordinates generation

Status and Plans

- After much effort, CHEF and its libraries are beginning to be applied to real problems

- Code base currently used at FNAL to
  1. study emittance preservation in the ILC (P. Lebrun). Newly developed linac-specific functionality (e.g. wakefields, RF structures) will be integrated. Good agreement with codes like LIAR and Merlin
  2. used by Synergia (3D space charge tracking by P.Spentzouris & J. Amunson)

- Near term (on going work):
  - full xsif parsing;
  - apertures and loss patterns,
  - matching.
  - library capabilities available through the user interface