ORBIT – A RING INJECTION CODE WITH SPACE CHARGE*

J. Galambos, S. Danilov, D. Jeon, J. Holmes, D. Olsen, ORNL, Oak Ridge, TN
J. Beebe-Wang, A. Luccio, BNL, Upton, NY

Abstract

ORBIT (Objective Ring Beam Injection and Tracking) is a new particle tracking code for rings. Modelling capabilities include H^- foil injection mechanisms, longitudinal and transverse space charge effects, and second order matrix transport. Additional code features include a programmable interactive driver shell, and interactive plotting.

1 INTRODUCTION

ORBIT, Objective Ring Beam Injection and Tracking, is a new C++ code developed for the Spallation Neutron Source (SNS) project to build the most intense pulsed neutron source. The code includes H^- foil injection modelling features needed to simulate realistic injection scenarios. Because the SNS project will produce intense pulses, 1-2x10^{14}, of low energy protons, 1 GeV, and is concerned with keeping uncontrolled beam losses to less than 1 part in 10^4, space charge models are provided to calculate beam halo, allowing minimisation of losses. The general code features are described, and some example calculations are shown. A more detailed description of using the code is given in a ORBIT User Manual [1].

2 FEATURES

2.1 Driver Shell

ORBIT is written in C++, and uses an interactive programmable driver shell [2], which can access variables and routines in the compiled accelerator modules. Additionally the driver shell allows the introduction of interpreted code for specific applications. Input files are scripts (in C++ syntax) which set-up the problem by appropriate variable initialisation, and routine calls and contain a combination of interpreted code and calls to compiled code. Because the driver shell is programmable, the tools provided in the compiled accelerator modules can be used together in a generalised fashion, and there are few predetermined execution flow paths. Input files are actually small programs. A portion of a typical input script is show in Table 1, illustrating some driver features such as mixing interpreted and compiled code.

2.2 Accelerator Classes

The accelerator modelling is structured in an object-oriented manner, based on two fundamental classes. First there is a MacroPart class which contains information about a "herd" of macro-particles being tracked. Also there is a Node class, which includes information about different actions that macro-particles may experience as they traverse about the ring. Examples of Node sub-classes are shown in Table 2. The Node class objects are related to the MacroPart classes in that they accept a MacroPart reference as an argument. This arrangement facilitates easy implementation of tracking multiple herds simultaneously. This is useful, for example, in some of the diagnostic procedures which are provided. The class structure is described in more detail in the ORBIT User Manual [1]. In setting up a ring for a particular run, the nodes may be put together in a general fashion.

Table 1: A portion of an input script for an ORBIT run.

```
////////////////////////////////////
// Add a non-accelerating  RF Cavity //
////////////////////////////////////
  Real tFactor;
  Integer nRFHarms = 1;
  RealVector volts(nRFHarms),
    harmNum(nRFHarms), RFPhase(nRFHarms);
  harmNum(1) = 1; RFPhase(1) = 0.;

// make a new interpreted routine:
  Void PSRVolts()
  {
   tFactor = time/0.825;
   if(tFactor > 1.) tFactor = 1.;
   volts(1) = 8. + 9. * tFactor;
  }
//   call a compiled routine:
  addRampedRFCavity("RF 1", 75, nRFHarms,
      volts, harmNum, RFPhase, PSRVolts);

// change a "compiled" variable:
     nLongBins = 64;
```

2.3 Modules and Other Features

Each of the accelerator classes in Table 2 represents an abstraction of an accelerator simulation action and is included in a separate module. Each module has an interface and an implementation. The interface describes how the module appears to a user, i.e. which variables and routines can be manipulated from the shell. The implementation contains the actual algorithms. The ability for the driver shell to communicate with specific module variables and routines is achieved by pre-processing the interface during the code build.

The code also includes features for flexible output and interactive plotting, either X-window or postscript. No proprietary software is used, but several freely available

* SNS research is sponsored by the Division of Materials Science, U.S. DOE, under contract number DE-AC05-96OR22464 with LMER Corp. for ORNL.

packages are incorporated. ORBIT has run on LINUX (Pentium PCs and DEC alpha's), Digital UNIX, IBM and SUN workstations. Typical run times for full space charge treatments are about 16 hours to simulate 1200 turn injection of the SNS accumulator ring using 100,000 macro-particles and 480 azimuthal steps per turn.

Table 2: Node classes in the ORBIT code. Indented entries represent sub-classes.

Node Class	**Description**
Transfer Matrix	Fundamental transport mechanism, using externally generated matrices from MAD.
1st order	Used for linear tracking.
2nd order*	Used for tracking at 2nd order.
Acceleration	Provides energy kicks due to an RF cavity.
RFCavity	Used for non-accelerating RF structures.
	Used for accelerating RF structures.
RampedBAccel*	For synchronous particle acceleration, with a prescribed B and Voltage ramp.
Longitudinal Space Charge	Gives a longitudinal space charge kick.
FFTLSpaceCharge	Use an FFT method, and optionally includes wall impedance effects.
Transverse Space Charge	Gives a transverse momentum kick from space charge.
PWSum	Calculate the space charge force using pairwise sum over macro-particles.
BrueForcePIC	Calculate the space charge force using a brute-force PIC.
FFT-PIC	Calculate the space charge force using an FFT PIC implementation.
Bump	Moves the closed orbit up (or down) as a function of time.
Foil	H^- source, and scattering element. Also calculates foil traversals.
Aperture	Either counts hits at a prescribed position, or removes macro-particles.
RectAperture	For a rectangular shaped aperture.
Thin Lens	Provides macro-particles with a thin lens kick.
ThinMPole*	Generalised multipole kick..
Diagnostic	Calculates diagnostic properties for a macro-particle herd.
MomentNode	Calculate moments of a herd.
StatLatNode	Calculate the statistical lattice parameters.
PMTNode	Poincare-Moment-Tracking – dumps the particle coordinates of a test herd, every "core oscillation" of the main herd.
LongMountainNode	Dumps the longitudinal histogram for a herd.

* These capabilities have been installed and are undergoing testing.

3 EXAMPLE CALCULATIONS

This section illustrates some examples of calculations that can be performed with ORBIT. By no means is it an exhaustive set of examples.

3.1 Injection Painting

Figure 1 shows an example of transverse and longitudinal phase space distributions resulting at the end of a programmed closed orbit bump painting, with no transverse space charge. These plots illustrate the built-in plotting capabilities. The hollow horizontal phase space profile was produced by a programmed closed orbit bump that spent more time injecting at the outer regions of x than for the inner regions of x.

3.2 Space Charge

Figure. 2 shows an example of a tune spread "necktie" diagram calculated by ORBIT with space charge at the end of injection. The tunes were calculated by the tune diagnostic and subsequently dumped to a file. Then this tune plot was created by an auxiliary plotting application, gnuplot. There are many built-in methods for dumping information to files, and it is also possible to perform customised data dumps from input script files by creating new output streams on the fly. Another example of a space charge diagnostic is a beam quadrupole moment shown in Fig. 3. This diagnostic shows the beam quadrupole moment for two different beam intensities during three turns at the end of injection. This is an example of a "high frequency" diagnostic. These moment diagnostics, as well as other diagnostics can also provide information at lower frequency, such as, once a turn. As indicated in Table 2, there are many diagnostic tools provided to understand the behaviour of the transverse space charge effects.

Finally in Fig. 4 we show an example of a benchmark comparison of simulated and experimentally observed beam profiles. This is a simulation of a Proton Storage Ring case, and is described in detail in Ref. [3]. Figure 4 shows the experimentally observed vertical beam profile, the simulated beam profile, and the simulated profile neglecting space charge effects. The simulated profile

agrees fairly well with the experimentally observed profile, but the agreement is not good if space charge effects are neglected. The ORBIT code is available at http://www.ornl.gov/~jdg/APGroup/Codes/Codes.html .

4 REFERENCES

1. J. Galambos, J. Holmes, D. Olsen, "ORBIT User Manual, V.1.0", SNS-ORNL-AP Tech. Note 11, March 1999.
2. S. W. Haney, "Using and Programming the SuperCode", UCRL-ID-118982, Oct. 1994.
3. J. Galambos, S. Danilov, D. Jeon. J. Holmes, D. Olsen, F. Neri, M. Plum, SNS/ORNL/AP Tech. Note 009.

Figure 1. Example ORBIT plots for transverse, upper, and longitudinal, lower, phase space distributions, calculated at the end of a programmed injection bump.

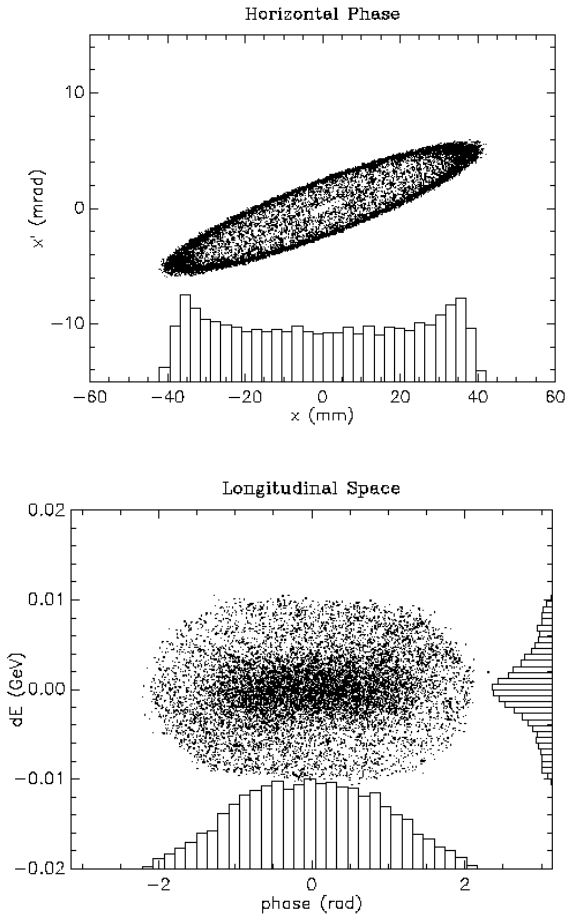

Figure 2. Transverse tune spread "necktie" information calculated by ORBIT at the end of injection.

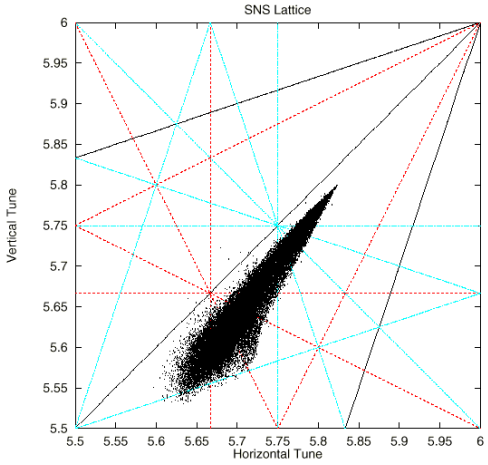

Figure 3. Example of the beam quadrupole moment verses position for three turns around the PSR ring near the end of injection, for two different intensities.

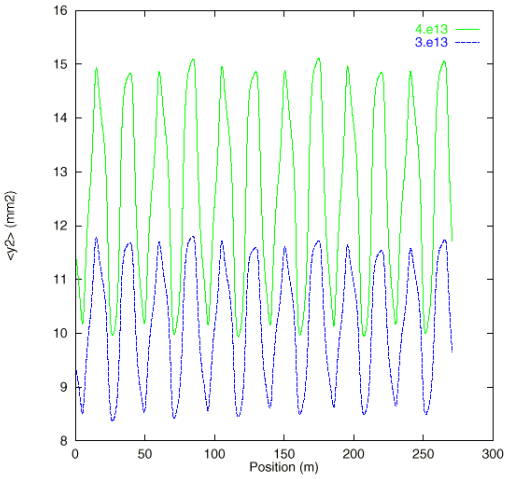

Figure 4. Comparison of calculated and measured PSR beam profiles. Inclusion of space charge is crucial to matching the observed profiles