Merlin++

A flexible and feature-rich library for accelerator simulations

Roger Barlow
Sam Tygier and Scott Rowan

The University of Huddersfield

13th November 2020

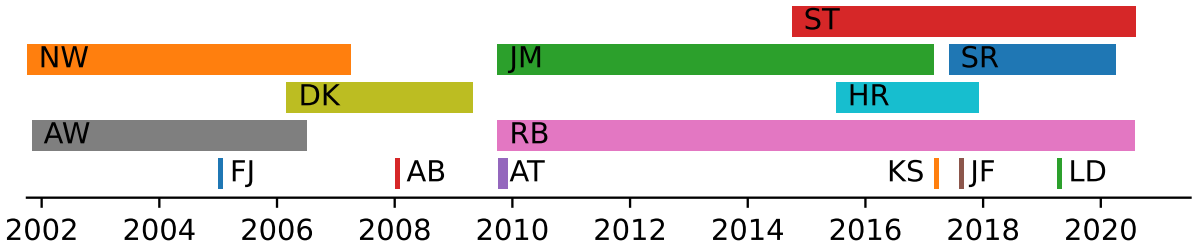
What it's all about

- What is Merlin++?
  - A short history
  - Some examples
  - Where Merlin belongs
- The big picture
  - A library not a program
  - A truly Object Oriented program
  - Using the C++ compiler
  - Writing solid software
- Performance and Features
- Getting started

What it's all about

A short history

History of Merlin++, formerly Merlin,



First developed at DESY, circa 2000, by Nick Walker for ILC studies

Extended by Andy Wolski to include linac and damping rings

Added Twiss parameter calculations and symplectic integrators

More features including wakefields, collimation and synchrotron radiation

Handed on to Manchester/Huddersfield in 2009

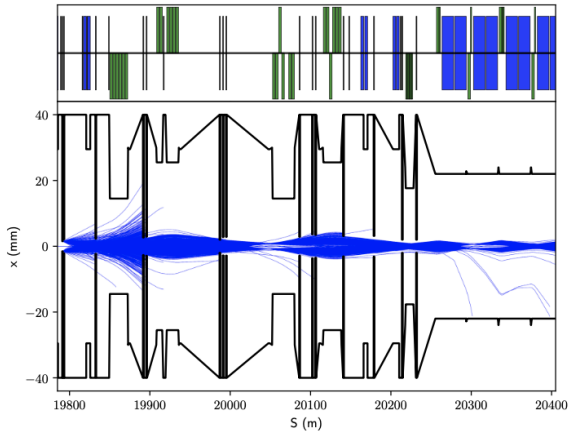
Developed including advanced scattering models and Hollow Electron Lens for LHC and HL-LHC collimation studies

Tidied up and renamed Merlin++

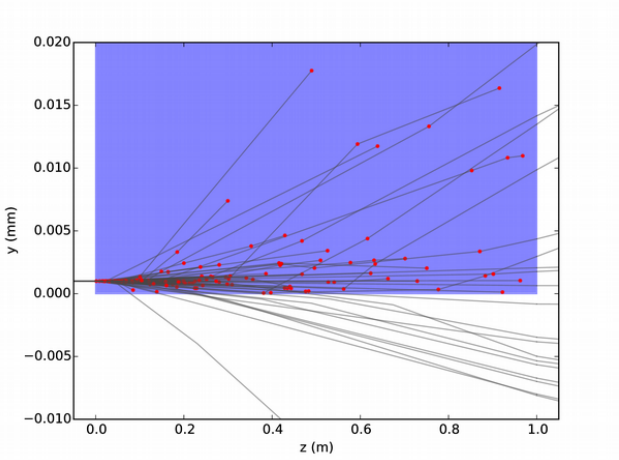
What it's all about

Some examples

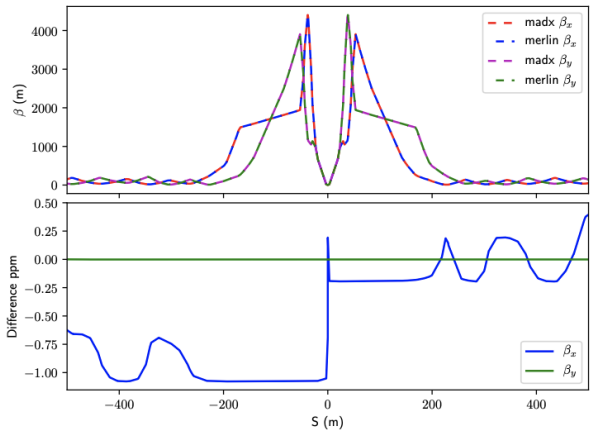
Particles in the LHC collimators



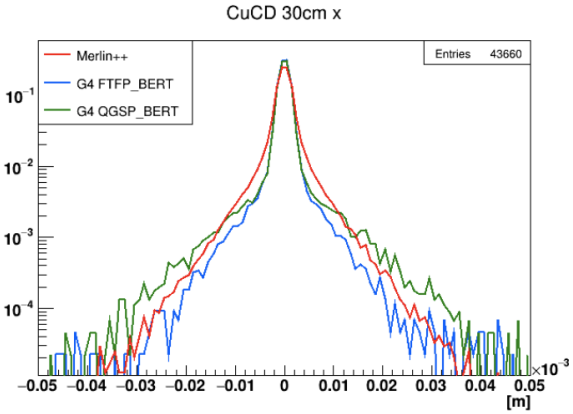
Beam hitting the edge of a collimator



β functions around ATLAS from MAD and Merlin



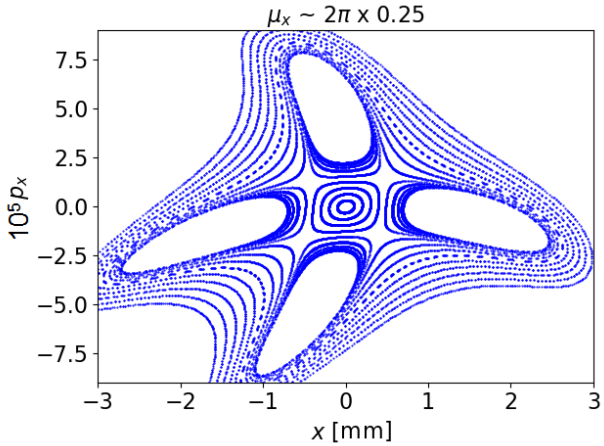
Beam through a copper slab: Comparison with 2xGeant4

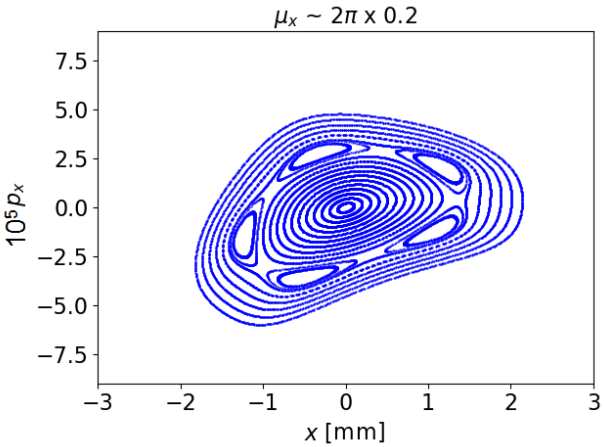


What it's all about

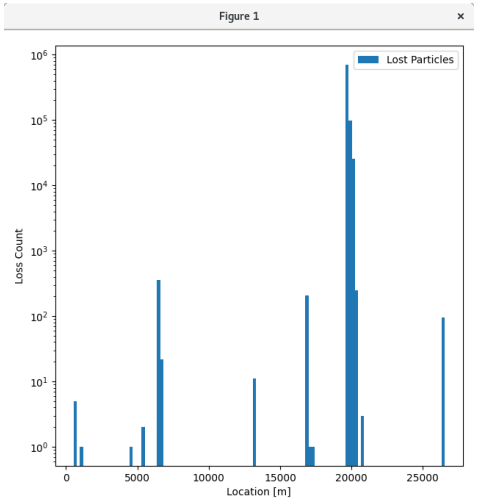
Some more examples

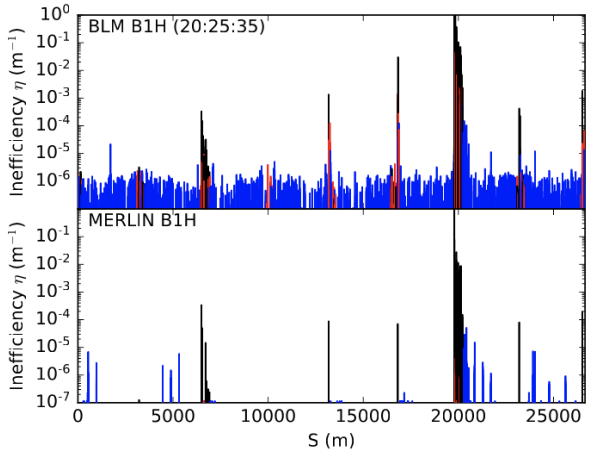
Sextupole at the end of a FODO lattice





Losses in the LHC





What it's all about

Merlin, MAD and others

MAD

Many more features

Geant4

Does not do detailed collisions and cascades

Does do beam optics and particle bunches – and much faster

FLUKA

Same as Geant4

SixTrack

Similar purpose – but cleaner

All the rest

Merlin++ is general purpose, specific aspects can be added

The big picture

Why Merlin++ is a library, not a stand-alone not a program

For	Against
No need to write parser	Have to compile program
Full flexibility of C++ language	User can do stupid things
User can do what they want with results. Developer does not have to anticipate	
User can easily add their own classes	

For the user

a library rather than a stand-alone program brings power and responsibility

A lot more power

and only a little more responsibility

The big picture

A full Object Oriented design

Accelerator (or beamline) comprises many components

Magnets (dipoles, quads, sextupoles), drifts, collimators...

Also applies to: particle distributions, trackers, scattering models ...

C-style solution

enumeration and switch

C++ solution

Inheritance: a quad is a magnet which is a component, and a particle is transported through it by its own member function

Extending Functionality

Makes it easy for the user to include a new process
Add child class with new feature - no need to change the core of Merlin++
If useful, can add to the library for other users

The big picture

Using new C++ features

You can write FORTRAN programs in any language

Writing C++ code involves a continual battle not to write in C
Design philosophy from the start was to use not just inheritance but all features of C++, e.g. Templates
Continue this philosophy as C++ develops (C++11, 14, 17...)
Some features of Merlin got included in later C++ versions. Discard and adapt new ones

Random Numbers

Nick wrote Merlin with its own random number generator, as the standard that came with C++ at the time wasn't good enough for long-cycle simulations.
C++11 included a proper Mersenne Twister random number generator
So we use it, and drop the old one

For Merlin++, backwards compatibility is not an argument

The big picture

Writing high qualitycode

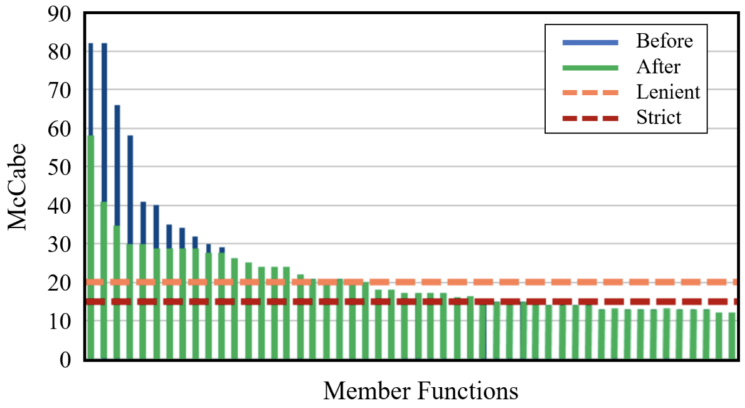
Good code is fast, usable and sustainable – code quality can be measured!
With help of CS colleagues (Colin Venters' group) , analysed Merlin
Scott Rowan et al, *Sustainability of the Merlin++ particle tracking code*, CHEP2018 https://doi.org/10.1051/epjconf/201921405028

Criteria e.g. from UK Software Sustainability Institute
Some are just tickboxes: licensing
Some use tools: github, uncrustify, doxygen, cmake tests
Some are providing material: website, tutorials, documentation
Some are more profound
e.g. meaningful names (`PointInside()` became `CheckWithinApertureBoundaries()`)

Sustainability Metric	Met Criteria	Initial Evaluation	Met Criteria	New Evaluation
Understandability	3/7	Unsatisfactory	6/7	Excellent
Documentation	3/19	Poor	14/19	Satisfactory
Buildability	3/9	Unsatisfactory	8/9	Excellent
Installability	7/14	Unsatisfactory	9/14	Satisfactory
Learnability	0/5	Poor	3/5	Satisfactory
Identity	3/7	Unsatisfactory	5/7	Satisfactory
Copyright	1/5	Poor	5/5	Excellent
Licencing	3/4	Satisfactory	4/4	Excellent
Governance	1/2	Unsatisfactory	2/2	Excellent
Community	1/11	Poor	6/11	Satisfactory
Accessibility	6/11	Satisfactory	8/11	Satisfactory
Testability	1/17	Poor	11/17	Satisfactory
Portability	10/16	Satisfactory	10/16	Satisfactory
Supportability	4/19	Poor	10/19	Satisfactory
Analysability	6/16	Unsatisfactory	13/16	Excellent
Changeability	3/10	Unsatisfactory	9/10	Excellent
Evolvability	0/3	Poor	2/3	Satisfactory
Interoperability	2/3	Satisfactory	3/3	Excellent

Cleaning up the code

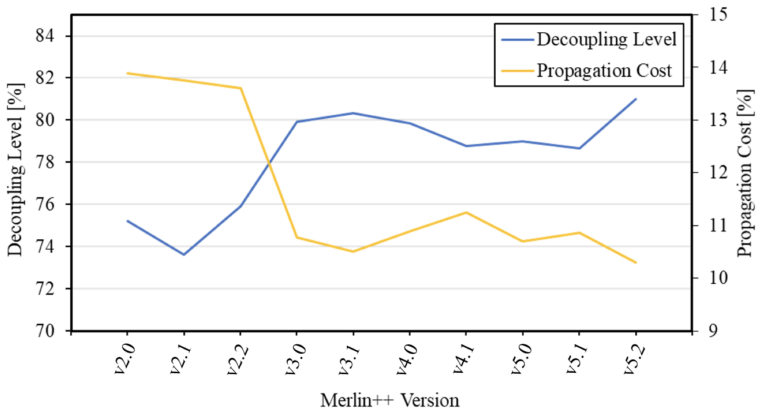
McCabe value: measures complexity (e.g. lots of `if` and `switch` statements means complex (=hard to read) code). Measured by `Metriculator` package
Cleaning up code also improves speed, through improving look-ahead (measured by valgrind)



The bad guys:

Long Methods, Large Classes, Long Parameter Lists, Switch Statements, Alternative Classes With Different Interfaces, Parallel Inheritance Hierarchies, Duplicate Code, Dead code and Middle Man classes

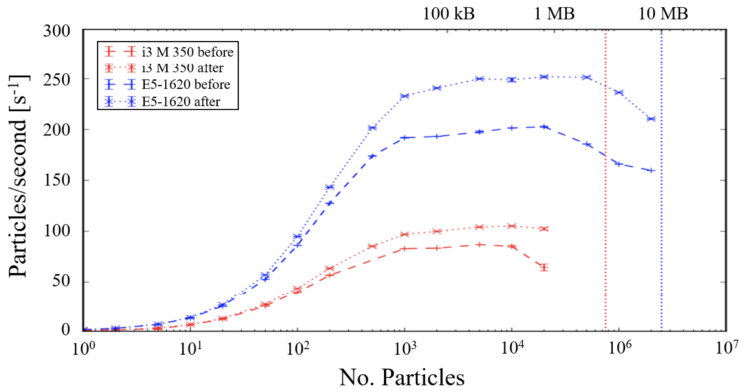
Cleaning up the code - decoupling



Use `ArchDiaDV8` tool to measure Propagation Cost (if you change one thing, how much elzse needs changing?) and Decoupling Level (are code modules independent?)

Overall figures generally good. Looking at history: got a lot better, then gradually worse, then better again thanks to Scott's clean-up efforts.

Not just cosmetic: changes also increase speed



Implemented on

- linux (obviously). Ubuntu and Centos
- Windows because some people do use it
- MacOS because laptops can be useful as well as beautiful
- Raspberry Pi just because we can
- HTC condor for high-volume work. Random number seeding needed

Testing it on different architectures and operating systems has brought odd issues to light, and forced us to conform to standards

Runs standalone or from Eclipse

Physics Features

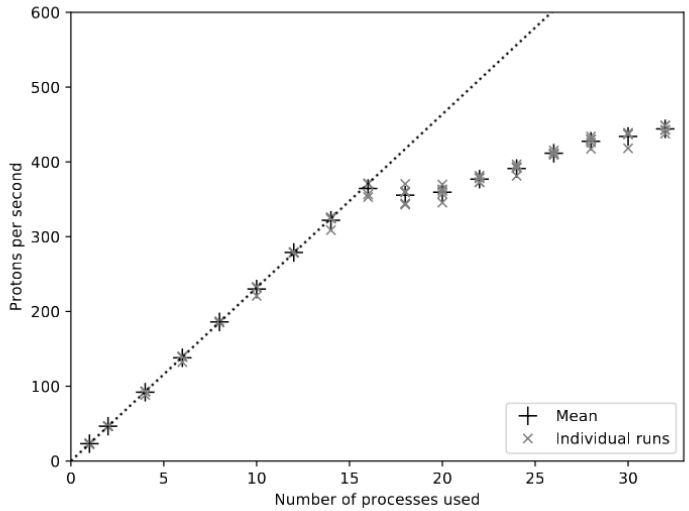
- Scattering: various models including the 'Practical Pomeron' implementation for elastic and diffractive scattering
- Synchrotron radiation - from a cooling point of view. Doesn't track the SR photons, though it could
- Spin tracking - havn't tested this but it's there
- High-order wakefields - geometric and resistive, for circular beampipes
- Hollow Electron Lens - Haroon Rafique's thesis
- Heavy Ions - Sam Tygier working on this for RHIC

Performance

For a collimation study (horizontal halo) with scattering

- Tracking 10,000 particles for 10 LHC turns takes 110 secs on a desktop
- Tracking 1,000,000 particles for 100 LHC turns takes 13782 seconds ($\sim$ 4 hours)

Can use multiple cores with openmp (results shown for 16-core Xeon)



Getting started and keeping going

(1) Read the paper, arXiv 2011.04345. Tells you enough about Merlin++ to decide whether it's going to be useful for you

(2) Go to the website. http://merlinpp.org

(3) Click on' Quickstart guide' and 'Installation' and follow instructions

(4) Try the tutorials, then start writing your own code - either from scratch or by adapting one of the examples.

arXiv.org > physics > arXiv:2011.04335

Physics > Accelerator Physics

[Submitted on 9 Nov 2020]

Merlin++, a flexible and feature-rich accelerator physics and particle tracking library

Robert Appleby, Roger Barlow, Dirk Kruecker, James Molson, Haroon Rafique, Scott Rowan, Sam Tygier, Nicholas Walker, Andrzej Wolski

Merlin++ is a C++ charged-particle tracking library developed for the simulation and analysis of complex beam dynamics within high energy particle accelerators. Accurate simulation and analysis of particle dynamics is an essential part of the design of new particle accelerators, and for the optimization of existing ones. Merlin++ is a feature-full library with focus on long-term tracking studies. A user may simulate distributions of protons or electrons in either single particle or sliced macro-particle bunches. The tracking code includes both straight and curvilinear coordinate systems allowing for the simulation of either linear or circular accelerator lattice designs, and uses a fast and accurate explicit symplectic integrator. Physics processes for common design studies have been implemented, including RF cavity acceleration, synchrotron radiation damping, on-line physical aperture checks and collimation, proton scattering, wakefield simulation, and spin-tracking. Merlin++ was written using C++ object orientated design practices and has been optimized for speed using multicore processors. This article presents an account of the program, including its functionality and guidance for use.

Subjects: **Accelerator Physics (physics.acc-ph)**
Cite as: arXiv:2011.04335 **[physics.acc-ph]**
(or arXiv:2011.04335v1 **[physics.acc-ph]** for this version)

The Merlin++ accelerator simulation program library

Merlin++ Home
Quickstart Guide
Developers' Guide
Debugging Tips
Profiling the code
Class definitions from doxygen
Historical
Merlin++ Publications
Merlin++ Authors
Contact the project team

The Merlin++ project

Merlin++ is an accelerator simulation program that tracks beams through components of a ring or beamline. Its functionality is similar to MAD but it has many extra features.

Merlin++ is written in C++, and unlike MAD and most other packages, it is actually a library of C++ routines. The user writes their own program and compiles it against the MERLIN class definitions and function libraries.

For convenience of potential users Merlin++ has been released in the Eclipse IDE, as this is in widespread and common use, however considerable care has been taken to ensure that neither Eclipse nor any other system is essential to the use of Merlin++. Similarly ROOT can be used to present results, but other methods are also available.

The release includes several example projects and test jobs, but we hope that users will develop their own projects from scratch and not just adapt those provided.

News

2020-03-06: Merlin++ 5.03 released
2019-03-19: Merlin++ 5.02 released

Merlin++ Particle Tracker

```
Test project /home/scott/git/merlin-cmake/build
    Start  1: have_python
1/16 Test  #1: have_python ..........................   Passed    0.69 sec
    Start  2: bunch_test
2/16 Test  #2: bunch_test ...........................   Passed    0.01 sec
    Start  3: bunch_io_test
3/16 Test  #3: bunch_io_test ........................   Passed    0.00 sec
    Start  4: landau_test.py
4/16 Test  #4: landau_test.py .......................   Passed    0.61 sec
    Start  5: aperture_test
5/16 Test  #5: aperture_test ........................   Passed    0.00 sec
    Start  6: collimate_particle_process_test
6/16 Test  #6: collimate_particle_process_test ......   Passed    0.00 sec
    Start  7: particle_bunch_constructor_test
7/16 Test  #7: particle_bunch_constructor_test ......   Passed   24.45 sec
    Start  8: lhc_optics_test
8/16 Test  #8: lhc_optics_test ......................   Passed    0.66 sec
    Start  9: lhc_fft_tune_test
9/16 Test  #9: lhc_fft_tune_test ....................   Passed    3.52 sec
    Start 10: cu50_test.py_1e7
10/16 Test #10: cu50_test.py_1e7 ....................   Passed   41.54 sec
    Start 11: cu50_test.py_1e7_sixtrack
11/16 Test #11: cu50_test.py_1e7_sixtrack ...........   Passed    9.96 sec
    Start 12: lhc_collimation_test.py_1e4
12/16 Test #12: lhc_collimation_test.py_1e4 .........   Passed  105.42 sec
    Start 13: basic_hollow_electron_lens_test.py
13/16 Test #13: basic_hollow_electron_lens_test.py ....   Passed    0.39 sec
    Start 14: diffusive_hollow_electron_lens_test
14/16 Test #14: diffusive_hollow_electron_lens_test ...   Passed    0.05 sec
    Start 15: datatable_test
15/16 Test #15: datatable_test ......................   Passed    0.01 sec
    Start 16: datatable_tfs_test
16/16 Test #16: datatable_tfs_test ..................   Passed    0.34 sec
```

Tutorial 1 - LatticeConfig

The following snippet shows how to manually define a simple FODO lattice storage ring, made solely of quadrupole and dipole,, within Merlin++.

First, an instance of the AcceleratorModelConstructor class is opened, and a AcceleratorModelConstructor::NewModel() is defined. Subsequently, a loop appends individu loop to complete the circle. Finally, an instance of AcceleratorModel is defined by the member function AcceleratorModelConstructor::GetModel().

```
AcceleratorModelConstructor latticeConstructor;
latticeConstructor.NewModel();
double rigidity = beamenergy/eV/SpeedOfLight;
double curv = (2*pi/(4*ncell));
//FODO lattice periodic
for (int n=1;n<(ncell+1);++n) {
  latticeConstructor.AppendComponent(new Quadrupole("QF",lquad,0.0098*rigidity), n==1 ? 0 : 0.15*lcell-ldipole);
  latticeConstructor.AppendComponent(new SectorBend("MB",ldipole,curv,rigidity*curv), 0.15*lcell-lquad);
  latticeConstructor.AppendComponent(new SectorBend("MB",ldipole,curv,rigidity*curv), 0.2*lcell-ldipole);
  latticeConstructor.AppendComponent(new Quadrupole("QD",lquad,-0.0098*rigidity), 0.15*lcell-ldipole);
  latticeConstructor.AppendComponent(new SectorBend("MB",ldipole,curv,rigidity*curv), 0.15*lcell-lquad);
  latticeConstructor.AppendComponent(new SectorBend("MB",ldipole,curv,rigidity*curv), 0.2*lcell-ldipole);
}
latticeConstructor.AppendDrift(0.15*lcell-ldipole);
AcceleratorModel* lattice = latticeConstructor.GetModel();
```

To confirm the lattice is viable for tracking it we calculate the closed orbit of a single reference particle following:

```
ClosedOrbit theClosedOrbit(lattice,beamenergy);
Particle particle(0);
theClosedOrbit.FindClosedOrbit(particle);
```

Finally

We have come through a fairly major re-write and clean-up of Merlin++, now we want to widen the user base

So do give it a try...

and get in touch