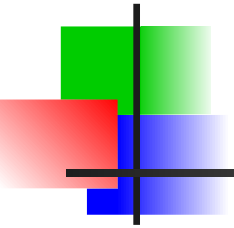
OPAL developments

C. Rogers

Status of OPAL developments

- Developing OPAL for vFFA
  - Drive vFFA design using series of (reasonably extensive) pythong scripts
  - Particularly focused on injection system
    - Pulsed magnets
    - Vertical orbit excursion
    - Charge exchange injection
    - Numerical approach to lattice design (no analytical approach exists)
  - Challenges
    - Dynamic aperture is very important
    - Effect of magnet fringe fields
    - Dipole phasing/beam time structure
- Driver
  - More elegant approach to lattice design (direct interface to python)
  - Improved code flexibility to support “exotic” lattice design



PyOpal

- Fundamental code unit is the (templated) PyOpalObject<C>
  - Has a shared pointer to a class C which is normally a OpalObject
  - Handles mapping of python object member data to Opal Attribute data
  - Handles a few common functions, where requested by dev
    - Execute
    - Register
    - Unwrapping Elements
    - Field lookup
- Code is in:
  - git@gitlab.psi.ch:ext-rogers_c/src.git

Example PyOpalObject

```
#include "PyOpal/ExceptionTranslation.h"
#include "PyOpal/PyOpalObject.h"

#include "Elements/OpalVerticalFFAMagnet.h"

//using namespace boost::python;
namespace PyOpal {
namespace PyVerticalFFAMagnet {

std::string track_run_docstring = std::string();


const char* module_docstring = "build a vertical_ffa_magnet";

template <>
std::vector<PyOpalObjectNS::AttributeDef> PyOpalObjectNS::PyOpalObject<OpalVerticalFFAMagnet>::attributes = {
    {"B0", "b0", "", PyOpalObjectNS::DOUBLE},
    {"FIELD_INDEX", "field_index", "", PyOpalObjectNS::DOUBLE},
    {"MAX_HORIZONTAL_POWER", "max_horizontal_power", "", PyOpalObjectNS::INT},
    {"CENTRE_LENGTH", "centre_length", "", PyOpalObjectNS::DOUBLE},
    {"END_LENGTH", "end_length", "", PyOpalObjectNS::DOUBLE},
    {"END_FIELD_MODEL", "end_field_model", "", PyOpalObjectNS::STRING},
    {"ENGE_PARAMETERS", "enge_parameters", "", PyOpalObjectNS::FLOATLIST},
    {"WIDTH", "width", "", PyOpalObjectNS::DOUBLE},
    {"HEIGHT_NEG_EXTENT", "height_neg_extent", "", PyOpalObjectNS::DOUBLE},
    {"HEIGHT_POS_EXTENT", "height_pos_extent", "", PyOpalObjectNS::DOUBLE},
    {"BB_LENGTH", "bb_length", "", PyOpalObjectNS::DOUBLE},
    {"BB_START_POSITION", "bb_start_position", "", PyOpalObjectNS::FLOATLIST},
    {"BB_START_NORMAL", "bb_start_normal", "", PyOpalObjectNS::FLOATLIST},
    {"BB_END_POSITION", "bb_end_position", "", PyOpalObjectNS::FLOATLIST},
    {"BB_END_NORMAL", "bb_end_normal", "", PyOpalObjectNS::FLOATLIST}
};

template <>
std::string PyOpalObjectNS::PyOpalObject<OpalVerticalFFAMagnet>::classDocstring = "";

BOOST_PYTHON_MODULE(vertical_ffa_magnet) {
    ExceptionTranslation::registerExceptions();
    PyOpalObjectNS::PyOpalObject<OpalVerticalFFAMagnet> element;
    auto elementClass = element.make_class("VerticalFFAMagnet");
    element.addGetOpalElement(elementClass);
    elementClass.def("get_field_value", &PyOpalObjectNS::getFieldValue<OpalVerticalFFAMagnet>);
}

}
}
```

Example PyOpalObject

```
template <>
std::vector<PyOpalObjectNS::AttributeDef> PyOpalObjectNS::PyOpalObject<OpalVertica
    {"B0", "b0", "", PyOpalObjectNS::DOUBLE},
    {"FIELD_INDEX", "field_index", "", PyOpalObjectNS::DOUBLE},
    {"MAX_HORIZONTAL_POWER", "max_horizontal_power", "", PyOpalObjectNS::INT},
    {"CENTRE_LENGTH", "centre_length", "", PyOpalObjectNS::DOUBLE},
    {"END_LENGTH", "end_length", "", PyOpalObjectNS::DOUBLE},
    {"END_FIELD_MODEL", "end_field_model", "", PyOpalObjectNS::STRING},
    {"ENGE_PARAMETERS", "enge_parameters", "", PyOpalObjectNS::FLOATLIST},
    {"WIDTH", "width", "", PyOpalObjectNS::DOUBLE},
    {"HEIGHT_NEG_EXTENT", "height_neg_extent", "", PyOpalObjectNS::DOUBLE},
    {"HEIGHT_POS_EXTENT", "height_pos_extent", "", PyOpalObjectNS::DOUBLE},
    {"BB_LENGTH", "bb_length", "", PyOpalObjectNS::DOUBLE},
    {"BB_START_POSITION", "bb_start_position", "", PyOpalObjectNS::FLOATLIST},
    {"BB_START_NORMAL", "bb_start_normal", "", PyOpalObjectNS::FLOATLIST},
    {"BB_END_POSITION", "bb_end_position", "", PyOpalObjectNS::FLOATLIST},
    {"BB_END_NORMAL", "bb_end_normal", "", PyOpalObjectNS::FLOATLIST}
};
```

Opal Attribute name

Python member name

Expected type

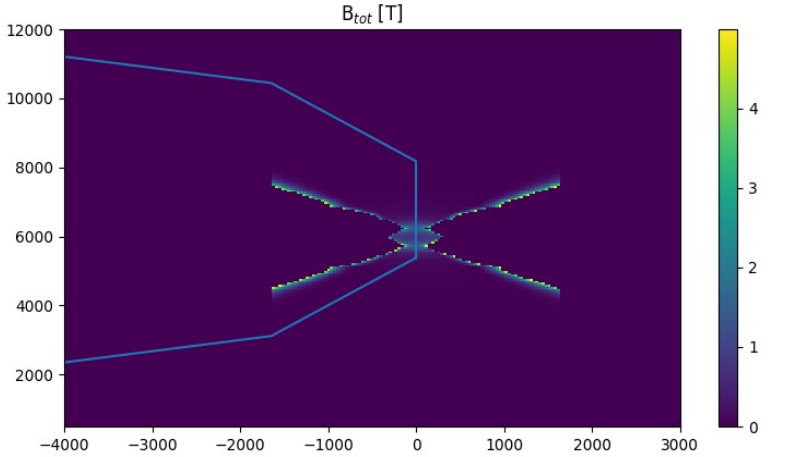
Example PyOpalObject

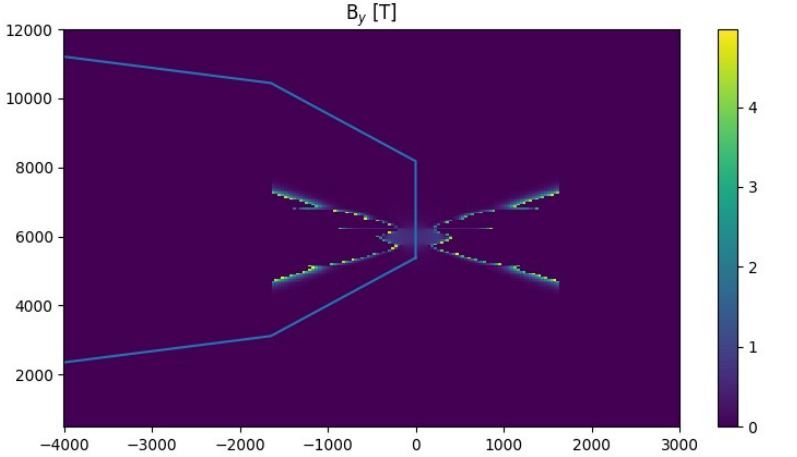
```
BOOST_PYTHON_MODULE(vertical_ffa_magnet) {
    ExceptionTranslation::registerExceptions();
    PyOpalObjectNS::PyOpalObject<OpalVerticalFFAMagnet> element;
    auto elementClass = element.make_class("VerticalFFAMagnet");
    element.addGetOpalElement(elementClass);
    elementClass.def("get_field_value", &PyOpalObjectNS::getFieldValue<OpalVerticalFFAMag
}
```

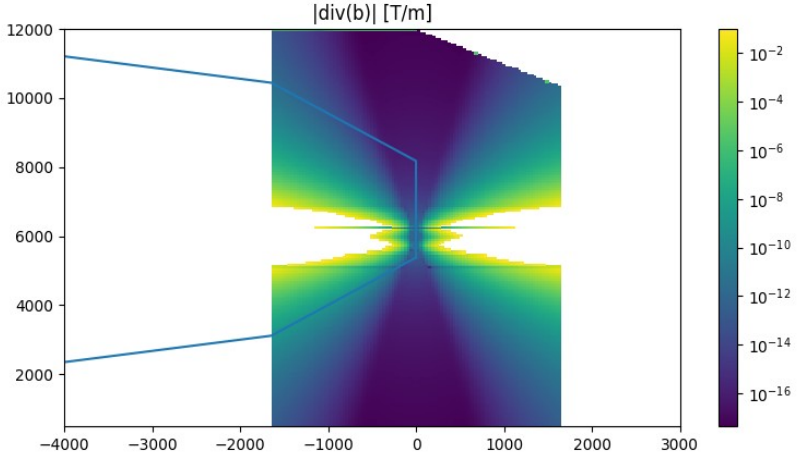
Example Python script

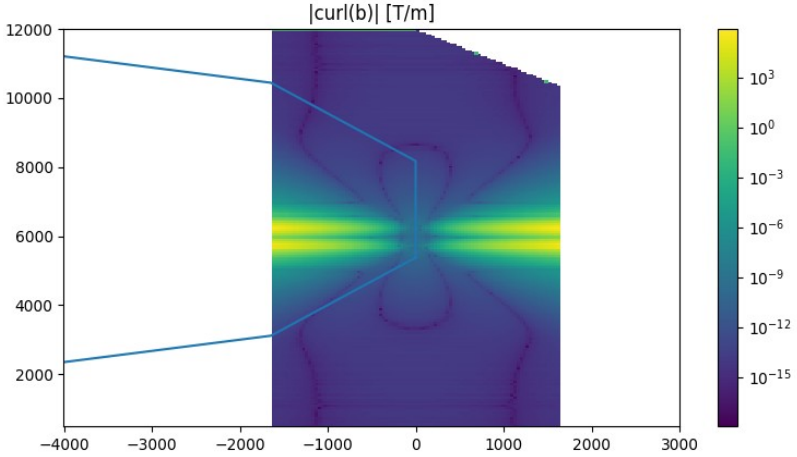
```
def main():
    magnet = PyOpal.vertical_ffa_magnet.VerticalFFAMagnet()
    magnet.b0 = 1.0
    magnet.field_index = 1.31
    magnet.max_horizontal_power = 12
    magnet.centre_length = 0.5
    magnet.end_length = 0.15
    magnet.end_field_model = "arctan"
    magnet.width = 2*math.sin(math.radians(36))*2.8
    magnet.height_neg_extent = 1.0
    magnet.height_pos_extent = 1.0
    magnet.bb_length = 12.0
    magnet.enge_parameters = [1.0, 2.0, 3.0]
    magnet.bb_start_position = [0.0, 0.0, 0.0]
    magnet.bb_start_normal = [0.0, 0.0, 1.0]
    magnet.bb_end_position = [0.0, 0.0, 12.0]
    magnet.bb_end_normal = [0.5, 0.0, 0.5]

    print("VFFA lookup")
    magnet.get_field_value(100.0, 0.0, 6000.0-250.0, 0.0)
```

Plots!


B_{tot} [T]
12000
10000
8000
6000
4000
2000
−4000
−3000
−2000
−1000
0
1000
2000
3000
4
3
2
1
0


B_y [T]
12000
10000
8000
6000
4000
2000
−4000
−3000
−2000
−1000
0
1000
2000
3000
4
3
2
1
0


|div(b)| [T/m]
12000
10000
8000
6000
4000
2000
−4000
−3000
−2000
−1000
0
1000
2000
3000
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}


|curl(b)| [T/m]
12000
10000
8000
6000
4000
2000
−4000
−3000
−2000
−1000
0
1000
2000
3000
10^{3}
10^{0}
10^{-3}
10^{-6}
10^{-9}
10^{-12}
10^{-15}

Line (looks like a list)

```cpp
#include "PyOpal/ExceptionTranslation.h"
#include "Lines/Line.h"
#include "PyOpal/PyLine.h"

//using namespace boost::python;
namespace PyOpal {
namespace PyLineNS {

const char* module_docstring =
"The line module handles building of a line of elements in OPAL";

template <>
std::vector<PyOpalObjectNS::AttributeDef> PyOpalObjectNS::PyOpalObject<TBeamline<Element> >:
    {"L", "length", "", PyOpalObjectNS::DOUBLE},
    {"ORIGIN", "origin", "", PyOpalObjectNS::STRING},
    {"ORIENTATION", "orientation", "", PyOpalObjectNS::STRING},
    {"X", "x", "", PyOpalObjectNS::DOUBLE},
    {"Y", "y", "", PyOpalObjectNS::DOUBLE},
    {"Z", "z", "", PyOpalObjectNS::DOUBLE},
    {"THETA", "theta", "", PyOpalObjectNS::DOUBLE},
    {"PHI", "phi", "", PyOpalObjectNS::DOUBLE},
    {"PSI", "psi", "", PyOpalObjectNS::DOUBLE}
};

template <>
std::string PyOpalObjectNS::PyOpalObject<TBeamline<Element> >::classDocstring = "";

BOOST_PYTHON_MODULE(line) {
    ExceptionTranslation::registerExceptions();
    PyLine element;
    auto lineClass = element.make_class("Line");
    // https://docs.python.org/3/library/collections.abc.html
    // I tried to pull everything from:
    // mutable sequence, sequence, reversible, iterator, iterable, container
    // so should look like a python list
    lineClass
        .def("__len__", &PyLine::getLength)
        .def("__getitem__", &PyLine::getElement)
        .def("__setitem__", &PyLine::setElement)
        .def("append", &PyLine::append)
```

Tracking – minimal tracking example

```
print("Running track_run test")
beam = PyOpal.beam.Beam()
beam.momentum = 0.1
beam.number_of_slices = 10
beam.register()

distribution = PyOpal.distribution.Distribution()
distribution.type = "FROMFILE"
root_dir = os.environ["OPAL_BUILD_PATH"]
print(root_dir)
distribution.fname = os.path.join(root_dir,"tests/opal_s
distribution.register()

line = make_line()
field_solver = make_field_solver()

track = PyOpal.track.Track()
track.line = "LINE"
track.beam = "BEAM"
run = PyOpal.track_run.TrackRun()
run.method = "CYCLOTRON-T"
print("set beam")
run.beam_name = "BEAM"
PyOpal.parser.list_objects()
print("get beam")
print(track.beam)
print("done")
print("TRACK")
track.execute()
print("TRACK RUN")
run.execute()
```

```
def make_field_solver():
    field_solver = PyOpal.field_solver.FieldSolver()
    field_solver.field_solver_type = "NONE"
    field_solver.mesh_size_x = 5
    field_solver.mesh_size_y = 5
    field_solver.mesh_size_t = 5
    field_solver.parallelize_x = False
    field_solver.parallelize_y = False
    field_solver.parallelize_t = False
    field_solver.boundary_x = "open"
    field_solver.boundary_y = "open"
    field_solver.boundary_t = "open"
    field_solver.bounding_box_increase = 2

    field_solver.register()
    return field_solver
```

Tracking – minimal tracking example

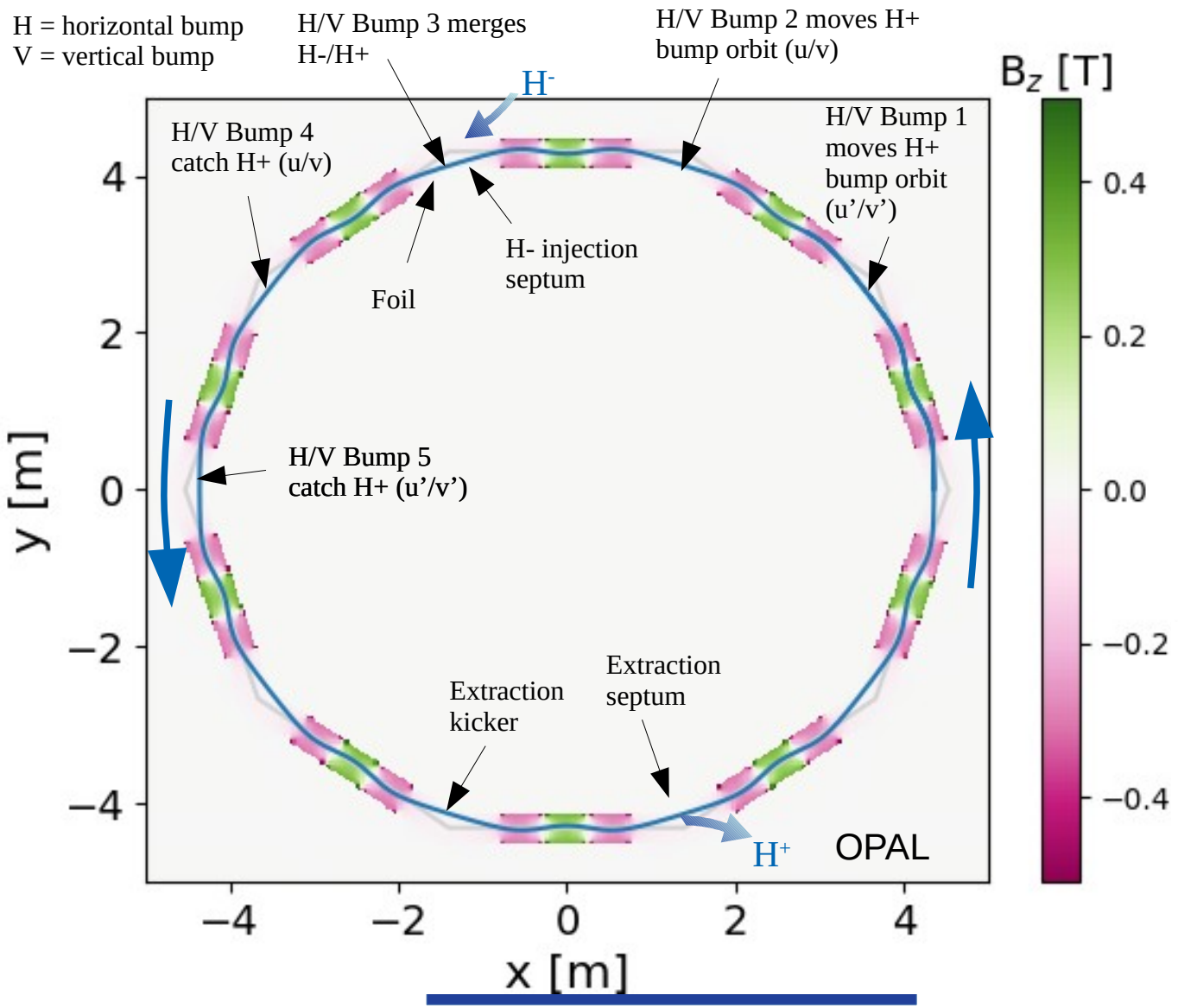
```
injectBeam 2
injectBeam 3
injectBeam 4
injectBeam 5
injectBeam 6
injectBeam 7
*** The MPI_Allreduce() function was called before MPI_INIT was invoked.
*** This is disallowed by the MPI standard.
*** Your MPI job will now abort.
[NDW1765:9995] Local abort before MPI_INIT completed successfully; not able to aggregate error messages, and not able to guarantee that all other processes were killed!
```

- Oh Noes!

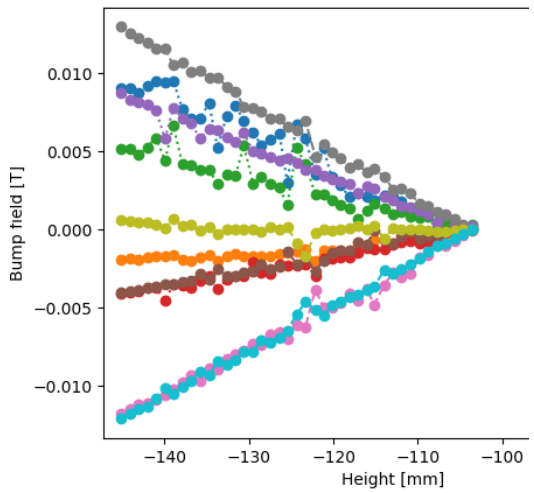
Also note on vFFA

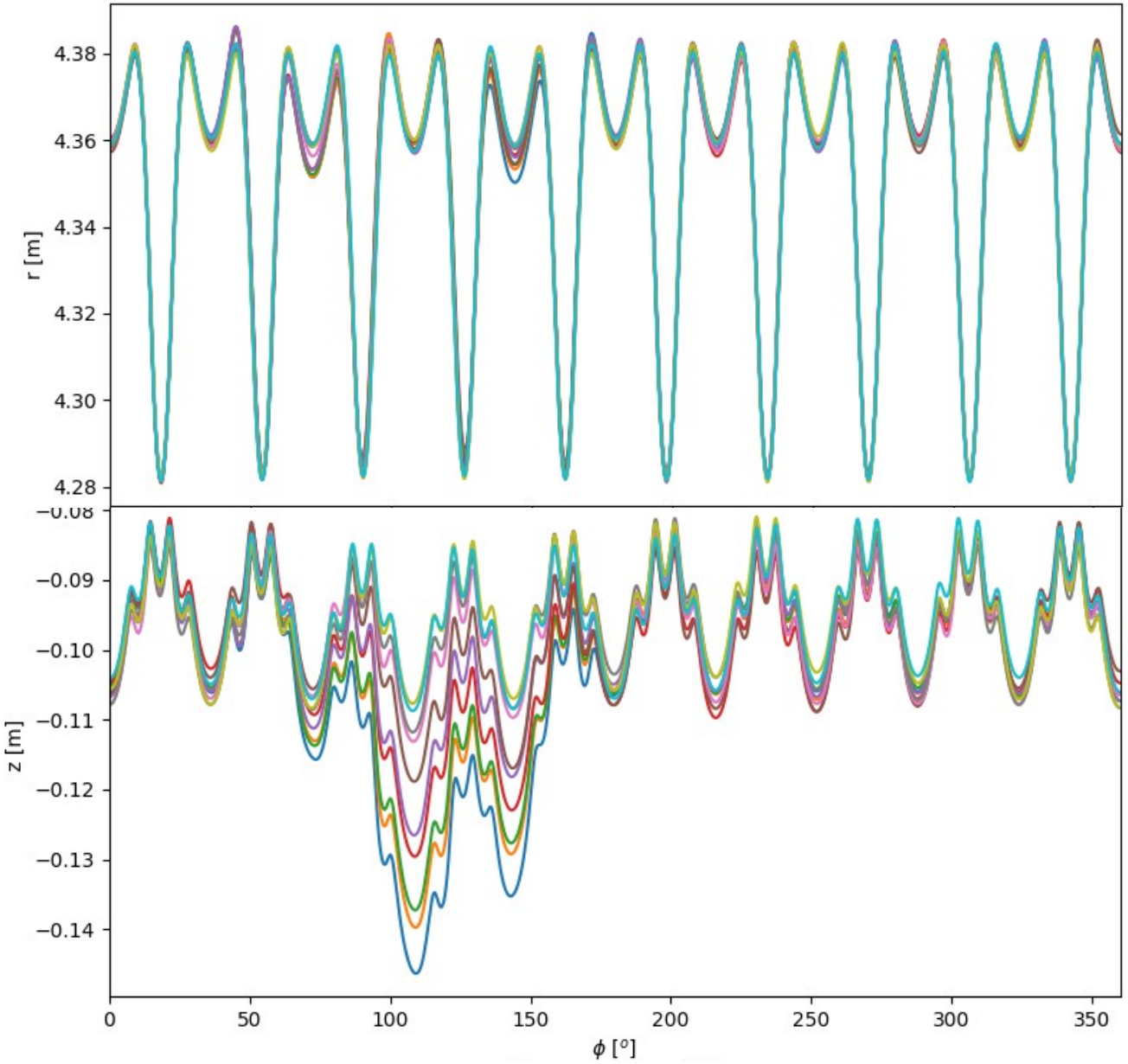
- Various things implemented but not committed (bad me):
  - Time dependence of dipoles
  - Full 3D placement in OPAL-Cycl/RingDefinition
  - OutputPlane routine
    - Improves PROBE using RK4 to estimate track position when it crosses a physical plane
  - (Hacky time-dependent beam injection)

Proposed Injection/Extraction System



Also note on vFFA





Milestones

- VFFA Milestones
  - Merge code!
  - Look at space charge simulation (note beam aspect ratio)
- PyOpal milestones
  - Library interface/development framework
    - Merge was never approved
  - First full tracking
  - Wrapping of various elements, actions, options, etc