4th E-TASC Scientific Board
Monitoring of ENR-MOD 2021 activities

Energetic particle optimization of stellarator devices using near-axis magnetic fields

ENR-MOD.01.IST-T001

Team: Rogerio Jorge, Paulo Rodrigues, Jorge Ferreira, António Figueiredo, Rui Coelho

IST - Instituto Superior Técnico, Universidade de Lisboa, Portugal

November 26, 2021

Outline

- Motivation
- Objectives
- Tasks
- 2022 Plan
- Preliminary Results
- Deliverables for 2022
- Possible Revisions for 2022

Motivation

PAPER

Modeling of energetic particle transport in optimized stellarators

A. Bader[7,1], D.T. Anderson[1], M. Drevlak[2], B.J. Faber[1], C.C. Hegna[1], S. Henneberg[2], M. Landreman[3], J.C. Schmitt[4], Y. Suzuki[5] and A. Ware[6]

Published 14 October 2021 •

Nuclear Fusion, Volume 61, Number 11

Citation A. Bader *et al* 2021 *Nucl. Fusion* **61** 116060

Author affiliations

[1] University of Wisconsin-Madison, Madison, WI, United States of America

[2] Max-Planck-Institut für Plasmaphysik, Greifswald, Germany

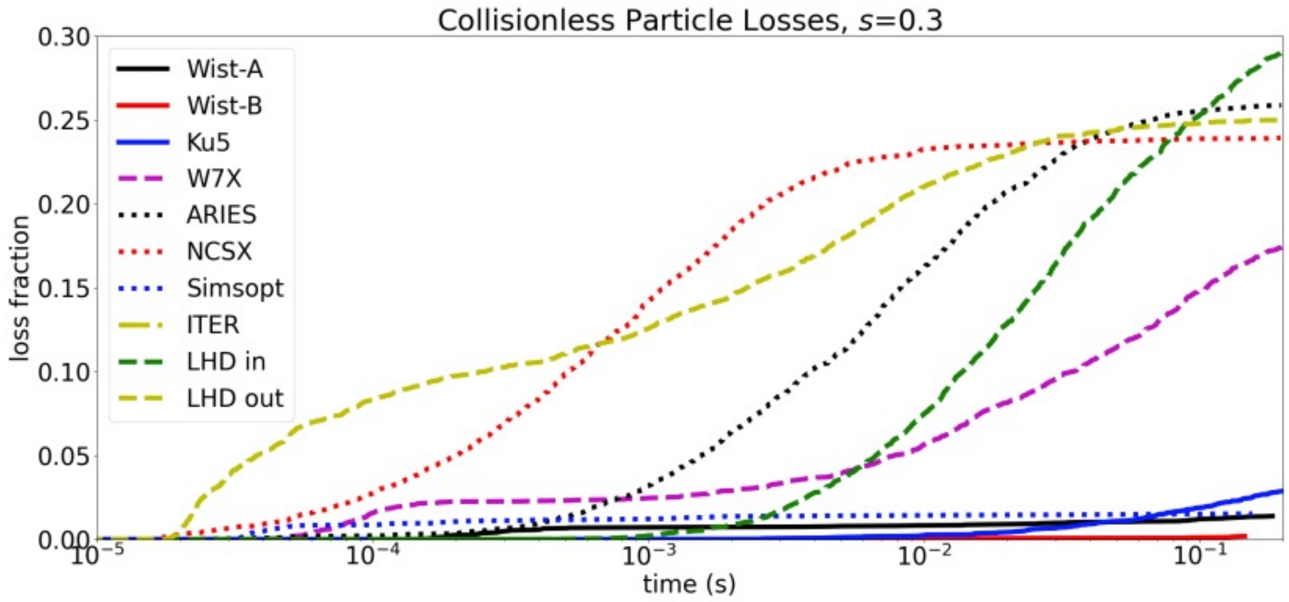
[3] University of Maryland, College Park, MD, United States of America

[4] Auburn University, Auburn, AL, United States of America

[5] Hiroshima University, Higashi-Hiroshima, Japan

[6] University of Montana, Missoula, MT, United states of America

Figure 4: Particle loss as a function of time for all configurations for particles born on the s=0.3 surface.



Why some stellarators confine particles better than others?

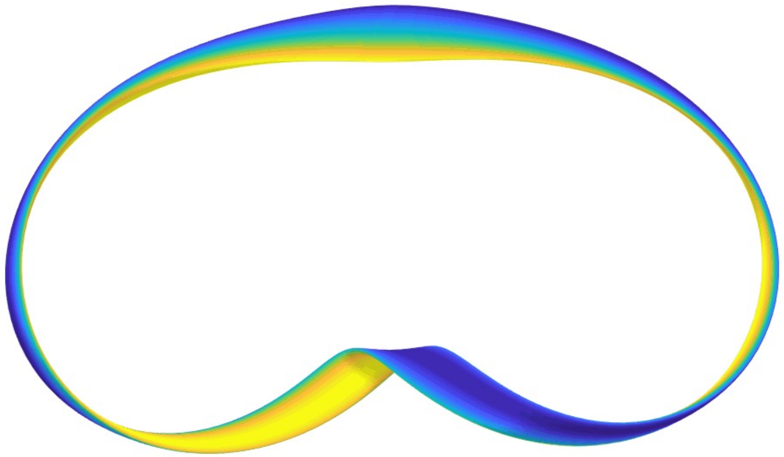
How to increase the performance of optimized stellarators?

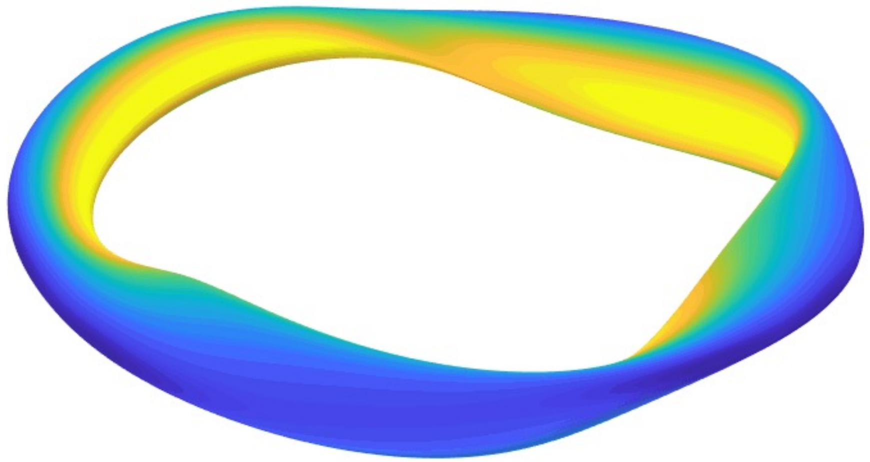
Can we survey the phase-space of possible stellarator shapes?

Objectives

Find reactor relevant stellarator shapes in a reliable and efficient manner

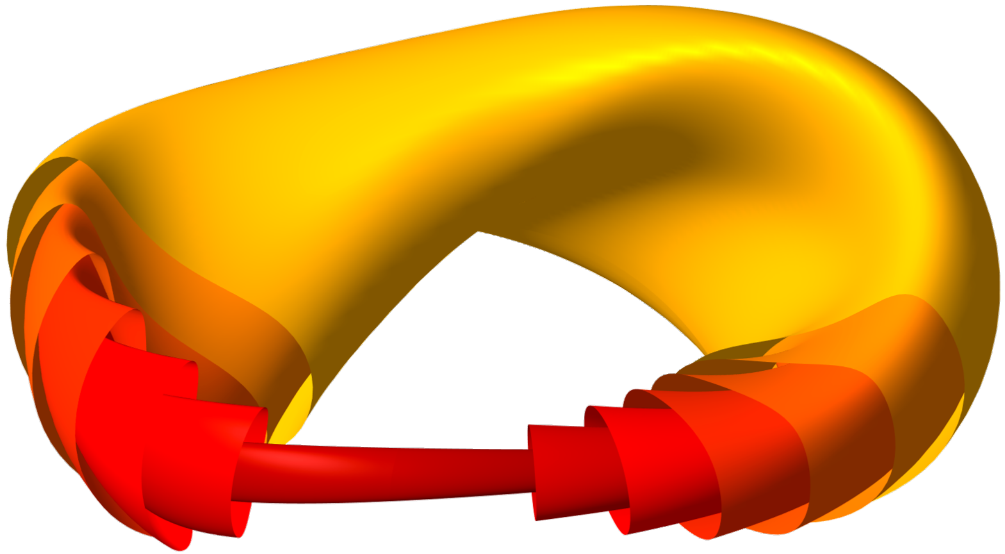
- Use the near-axis expansion to get a stellarator shape (~1 ms)
- Obtain particle confinement using particle tracer codes (~10 s)
- Optimize in the space of near-axis configurations (~10 min)

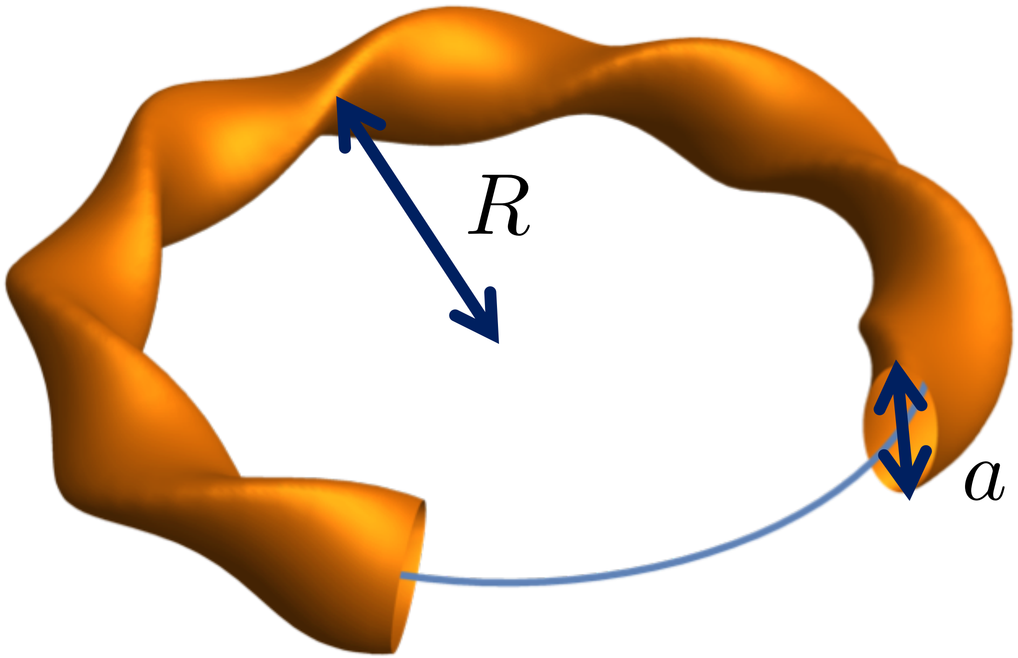




The near-axis expansion

Solve ideal MHD equations for small $\epsilon = a/R$





Direct approach

[R. Jorge et al, JPP 86 (2020)]

Inverse approach

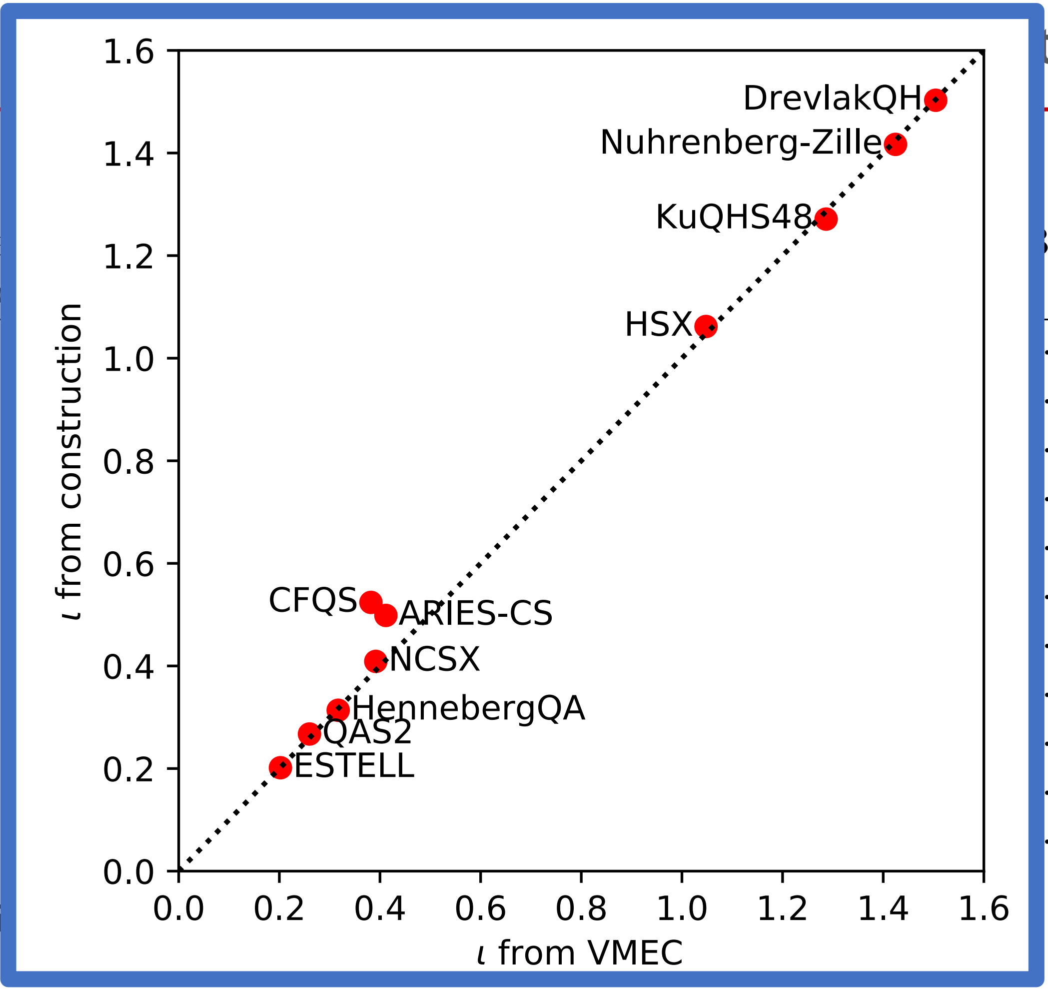
[M. Landreman et al, JPP 86 (2018)]

Every quasisymmetric design available

- Construct a near-axis representation for each design
- Compare VM tion

Configuration	Aspe ratio	est-fit [m^{-1}]	B_0 [T]
NZ1988	12	.157	0.205
HSX	10	.28	1.00
KuQHS48	8.1	.147	1.20
WISTELL-A	6.7	.791	2.54
Drevlak	8.6	.0899	3.97
NCSX	4.4	.408	1.55
ARIES-CS	4.5	.0740	5.69
QAS2	2.6	.347	1.79
ESTELL	5.3	.570	1.00
CFQS	4.3	.586	0.933
Henneberg	3.4	.302	2.41

Table 1: Quasi



Jorge & Landreman, *Plasma Phys. Control. Fusion* **63** (2021)

Tasks 2022

WP1 – Particle Tracer Code for Arbitrary Geometries

Solve guiding-center equations in several geometries (analytical/numerical):

- Near-axis direct approach (Mercier)
- Near-axis indirect approach (Garren-Boozer)
- VMEC
- Helena (benchmarking)

WP2 – Measure and Optimize Confinement

- Numerical and analytical metrics for fast-particle confinement
- Integration with SIMSOPT

Tasks 2023

WP3 – Optimized Stellarator Equilibria

Find optimized designs (not necessarily quasisymmetric or quasi-isodynamic) using a SIMSOPT based optimization

WP4 – Physics Study of Nemov's Γ_c criterion

Study the viability of using the parameter Γ_c as a cost function in fast-particle confinement stellarator studies

WP5 – Orbits in Free-Boundary Magnetic Fields

Using SPEC or VMEC, study particle orbits in coil-based optimized stellarators

2022 plan – Particle Tracing

::gyronimo:: - gyromotion for the people, by the people -

An object-oriented library for gyromotion applications in plasma physics.

Philosophy and purpose:

Have you ever had a bright and promising idea about gyromotion in plasmas that just faded away the moment you realised the amount of non-trivial, tedious, unrewarding, non-physics details you would have to implement before you could get a simple glimpse over the results?

P. Rodrigues, *github.com/prodrigs/gyronimo*

Solve guiding-center equations of motion in a general coordinate system

$$\left[1+\frac{\tilde{v}_\parallel}{\tilde{\Omega}}\left(\mathbf{b}\cdot\tilde{\nabla}\times\mathbf{b}\right)\right]\frac{d\mathbf{X}}{d\tau}=\tilde{v}_\parallel\mathbf{b}+\frac{1}{\tilde{\Omega}}\left[\tilde{v}_\parallel^2\tilde{\nabla}\times\mathbf{b}+\mathbf{b}\times\left(\tilde{v}_\parallel\partial_\tau\mathbf{b}+\frac{1}{2}\tilde{\mu}\tilde{\nabla}\tilde{B}-\tilde{\mathbf{E}}\right)\right]$$

$$\left[1+\frac{\tilde{v}_\parallel}{\tilde{\Omega}}\left(\mathbf{b}\cdot\tilde{\nabla}\times\mathbf{b}\right)\right]\frac{d\tilde{v}_\parallel}{d\tau}=-\left(\mathbf{b}+\frac{\tilde{v}_\parallel}{\tilde{\Omega}}\tilde{\nabla}\times\mathbf{b}\right)\cdot\left(\frac{1}{2}\tilde{\mu}\tilde{\nabla}\tilde{B}+\tilde{v}_\parallel\partial_\tau\mathbf{b}-\tilde{\mathbf{E}}\right).$$

Test near axis direct approach, indirect, VMEC, SPEC, Bio—Savart, etc…

2022 plan

WP1

C++ Library with documentation on GitHub

Python interface with near-axis expansion coordinates

C++ interface with VMEC equilibria

Benchmark with other particle tracer codes such as BEAMS3D

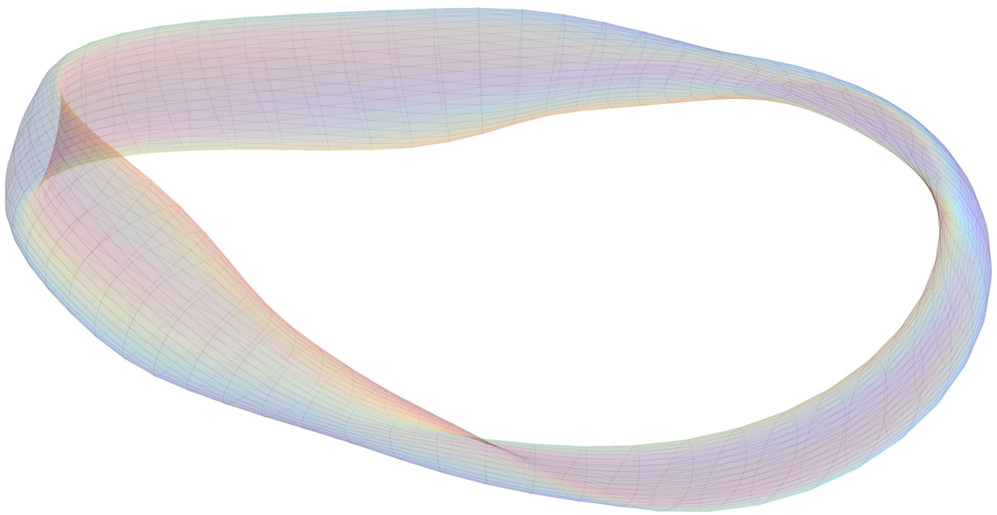
WP2

Analytical/numerical objective function to measure fast particle confinement

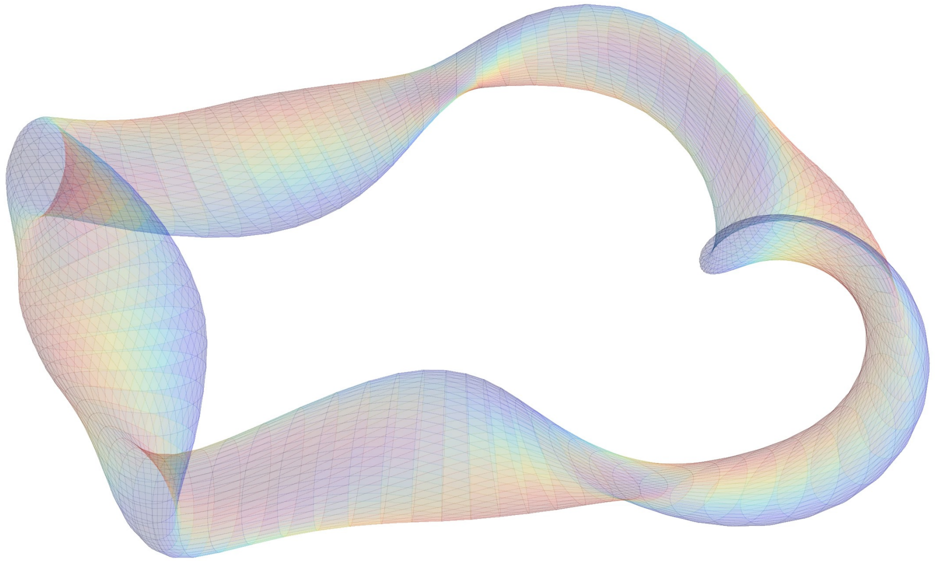
Python interface with SIMSOPT/STELLOPT

Preliminary Results

Second order near-axis geometry



Quasi-axisymmetric design of [1], section 5.2



Quasi-helically symmetric design of [1], section 5.4

[1] Landreman & Sengupta (2019), 85(6), JPP

Preliminary Results

WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓

C++ interface with VMEC equilibria

Benchmark with other particle tracer codes such as BEAMS3D ✓

WP2

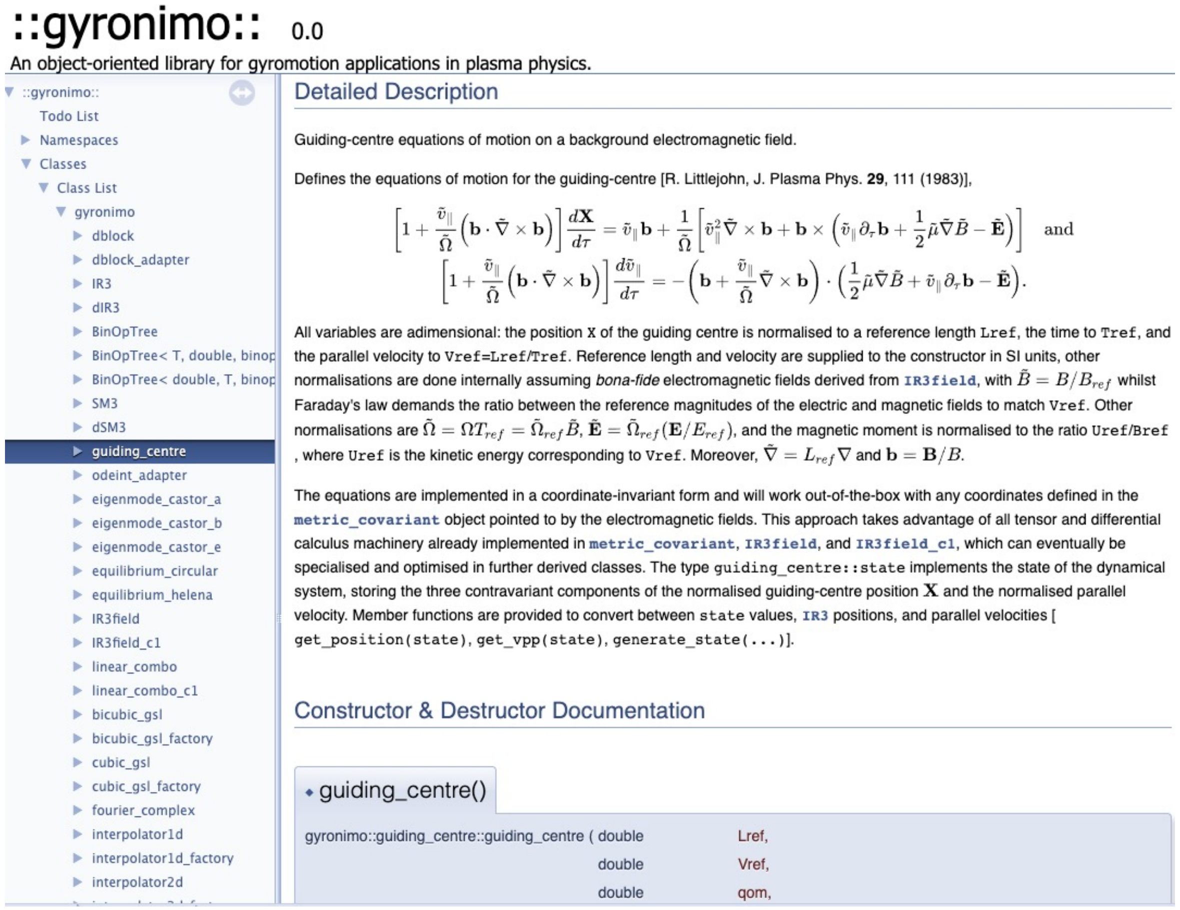
Analytical/numerical objective function to measure fast particle confinement

Python interface with SIMSOPT/STELLOPT ✓

Preliminary Results

WP1

C++ Library with documentation on GitHub ✓



::gyronimo:: 0.0

An object-oriented library for gyromotion applications in plasma physics.

- ::gyronimo::
  - Todo List
  - Namespaces
  - Classes
    - Class List
      - gyronimo
        - dblock
        - dblock_adapter
        - IR3
        - dIR3
        - BinOpTree
        - BinOpTree< T, double, binop
        - BinOpTree< double, T, binop
        - SM3
        - dSM3
        - guiding_centre
        - odeint_adapter
        - eigenmode_castor_a
        - eigenmode_castor_b
        - eigenmode_castor_e
        - equilibrium_circular
        - equilibrium_helena
        - IR3field
        - IR3field_c1
        - linear_combo
        - linear_combo_c1
        - bicubic_gsl
        - bicubic_gsl_factory
        - cubic_gsl
        - cubic_gsl_factory
        - fourier_complex
        - interpolator1d
        - interpolator1d_factory
        - interpolator2d

Detailed Description

Guiding-centre equations of motion on a background electromagnetic field.

Defines the equations of motion for the guiding-centre [R. Littlejohn, J. Plasma Phys. **29**, 111 (1983)],

$$\left[1+\frac{\tilde{v}_\parallel}{\tilde{\Omega}}\left(\mathbf{b}\cdot\tilde{\nabla}\times\mathbf{b}\right)\right]\frac{d\mathbf{X}}{d\tau}=\tilde{v}_\parallel\mathbf{b}+\frac{1}{\tilde{\Omega}}\left[\tilde{v}_\parallel^2\tilde{\nabla}\times\mathbf{b}+\mathbf{b}\times\left(\tilde{v}_\parallel\partial_\tau\mathbf{b}+\frac{1}{2}\tilde{\mu}\tilde{\nabla}\tilde{B}-\tilde{\mathbf{E}}\right)\right]\quad\text{and}$$

$$\left[1+\frac{\tilde{v}_\parallel}{\tilde{\Omega}}\left(\mathbf{b}\cdot\tilde{\nabla}\times\mathbf{b}\right)\right]\frac{d\tilde{v}_\parallel}{d\tau}=-\left(\mathbf{b}+\frac{\tilde{v}_\parallel}{\tilde{\Omega}}\tilde{\nabla}\times\mathbf{b}\right)\cdot\left(\frac{1}{2}\tilde{\mu}\tilde{\nabla}\tilde{B}+\tilde{v}_\parallel\partial_\tau\mathbf{b}-\tilde{\mathbf{E}}\right).$$

All variables are adimensional: the position `X` of the guiding centre is normalised to a reference length `Lref`, the time to `Tref`, and the parallel velocity to `Vref=Lref/Tref`. Reference length and velocity are supplied to the constructor in SI units, other normalisations are done internally assuming *bona-fide* electromagnetic fields derived from `IR3field`, with $\tilde{B}=B/B_{ref}$ whilst Faraday's law demands the ratio between the reference magnitudes of the electric and magnetic fields to match `Vref`. Other normalisations are $\tilde{\Omega}=\Omega T_{ref}=\tilde{\Omega}_{ref}\tilde{B}$, $\tilde{\mathbf{E}}=\tilde{\Omega}_{ref}(\mathbf{E}/E_{ref})$, and the magnetic moment is normalised to the ratio `Uref/Bref` , where `Uref` is the kinetic energy corresponding to `Vref`. Moreover, $\tilde{\nabla}=L_{ref}\nabla$ and $\mathbf{b}=\mathbf{B}/B$.

The equations are implemented in a coordinate-invariant form and will work out-of-the-box with any coordinates defined in the `metric_covariant` object pointed to by the electromagnetic fields. This approach takes advantage of all tensor and differential calculus machinery already implemented in `metric_covariant`, `IR3field`, and `IR3field_c1`, which can eventually be specialised and optimised in further derived classes. The type `guiding_centre::state` implements the state of the dynamical system, storing the three contravariant components of the normalised guiding-centre position $\mathbf{X}$ and the normalised parallel velocity. Member functions are provided to convert between `state` values, `IR3` positions, and parallel velocities [`get_position(state)`, `get_vpp(state)`, `generate_state(...)`].

Constructor & Destructor Documentation

◆ guiding_centre()

gyronimo::guiding_centre::guiding_centre (	double	Lref,
	double	Vref,
	double	qom,

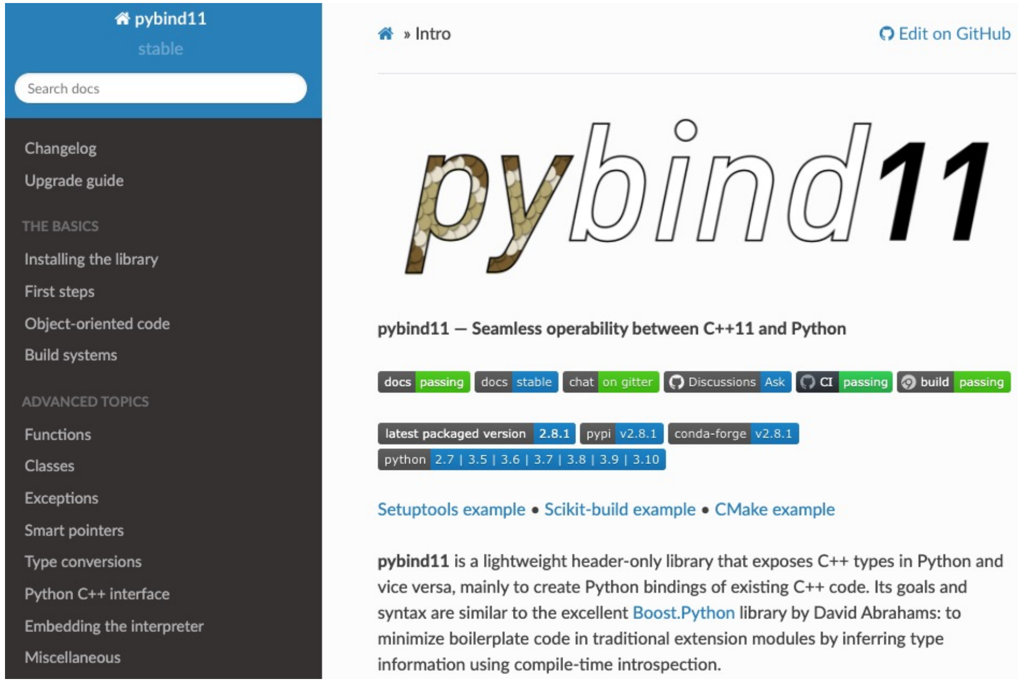
Work by Paulo Rodrigues, IST

Preliminary Results

WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓



PYBIND11: Seamless creation of Python bindings to C++ code

- Straight interfacing of gyronimo C++ libraries in Python
- Easy to connect with other stellarator codes
- Orchestrating multiple particle tracing
- Post-processing and plotting
- Lightweight

In other words…

Python – Slower, easy to read, write and plot data
C++ – Faster, core of the code, solves the equations

Preliminary Results

WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓

C++

```cpp
#include <../include/pybind11/pybind11.h>
#include <../include/pybind11/stl.h>
#include <gyronimo/interpolators/cubic_gsl.hh>
#include <gyronimo/metrics/metric_stellna.hh>
#include <gyronimo/core/dblock.hh>
```

```cpp
#include <gyronimo/dynamics/odeint_adapter.hh>
#include <gyronimo/core/codata.hh>
namespace py = pybind11;
using namespace gyronimo;

std::vector< std::vector<double>> gc_solver(
  int field_periods,
```

```cpp
  // Integrate for t in [0,Tfinal], with dt=Tfinal/nsamples, using RK4.
  boost::numeric::odeint::runge_kutta4<gyronimo::guiding_centre::state> integration_algorithm;
  boost::numeric::odeint::integrate_const(
      integration_algorithm, odeint_adapter(&gc),
      initial_state, 0.0, Tfinal, Tfinal/nsamples, push_back_state_and_time(x_vec,&qsc,&gc) );

  return x_vec;
}

// Python wrapper functions
PYBIND11_MODULE(NEAT, m) {
    m.doc() = "Gyronimo Wrapper for the Stellarator Near-Axis Expansion (STELLNA)";
    m.def("gc_solver",&gc_solver);
}
```

Python

```python
# Quasi-helically symmetric stellarator
r0=0.15
Lambda    = [0.86]
Tfinal    = 400
stel = Qsc.from_paper(4, nphi=nphi)
theta0    = [1.15]
phi0      = [0.0]
```

```python
# Call Gyronimo
sol = np.array(NEAT.gc_solver(int(stel.nfp),
```

```python
import mayavi.mlab as mlab
fig = mlab.figure(bgcolor=(1,1,1), size=(430,720))
[mlab.plot3d(rpos_cartesian[i][0],rpos_cartesian[i][1],rpos_cartesian[i][2],
```

Preliminary Results

WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓

Able to model QA (NCSX), QH (HSX) and QI (W7-X) stellarators

Python

```python
# Quasi-helically symmetric stellarator
r0=0.15
Lambda    = [0.86]
Tfinal    = 400
stel = Qsc.from_paper(4, nphi=nphi)
theta0    = [1.15]
phi0      = [0.0]
```

- QA and QH stellarators conserve canonical angular momentum

$$p_\chi = \left(\frac{\partial L}{\partial \dot{\zeta}}\right)_{\chi,\dot{\chi},\zeta} = q\frac{N}{M}\psi - q\psi_p + \frac{mv_{||}}{B}\left(G + \frac{N}{M}I\right)$$

$$\text{orbit width} \propto 1/|\iota - N|$$

- QI stellarators do not conserve canonical angular momentum, but estimates are possible

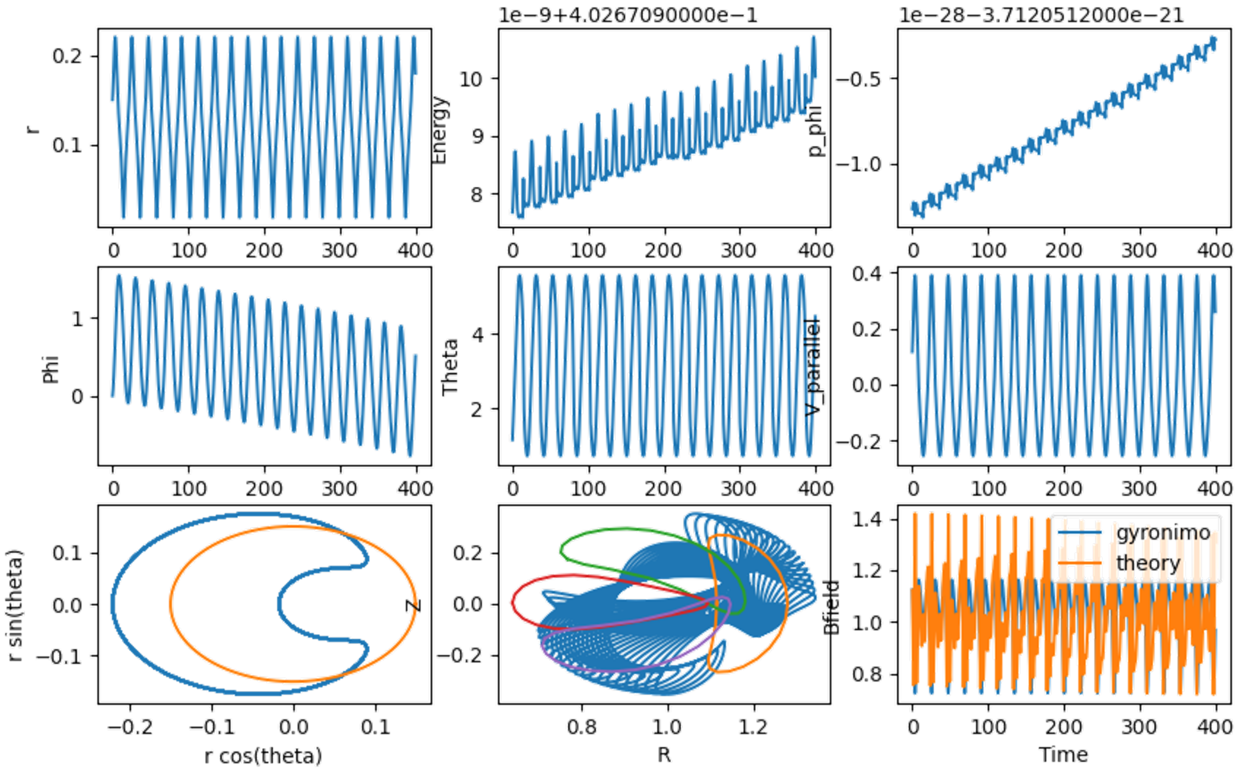
$$\Delta\psi = -\frac{\partial}{\partial\alpha}\int_{B(l_0)}^{B(l)} h\frac{\partial}{\partial B}\left(\frac{v_{\|}}{\Omega}\right)\mathrm{d}B$$

Preliminary Results

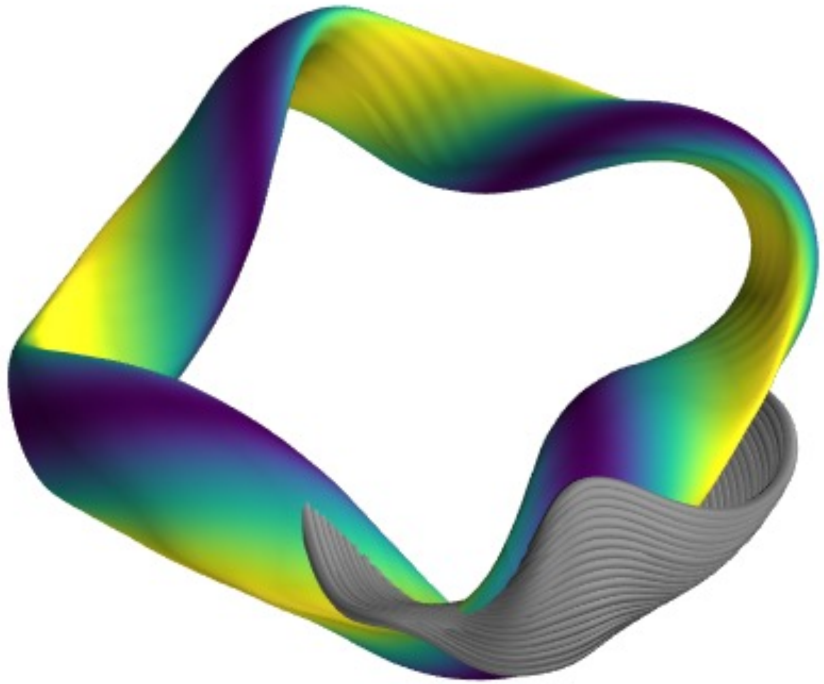
WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓



QH Stellarator

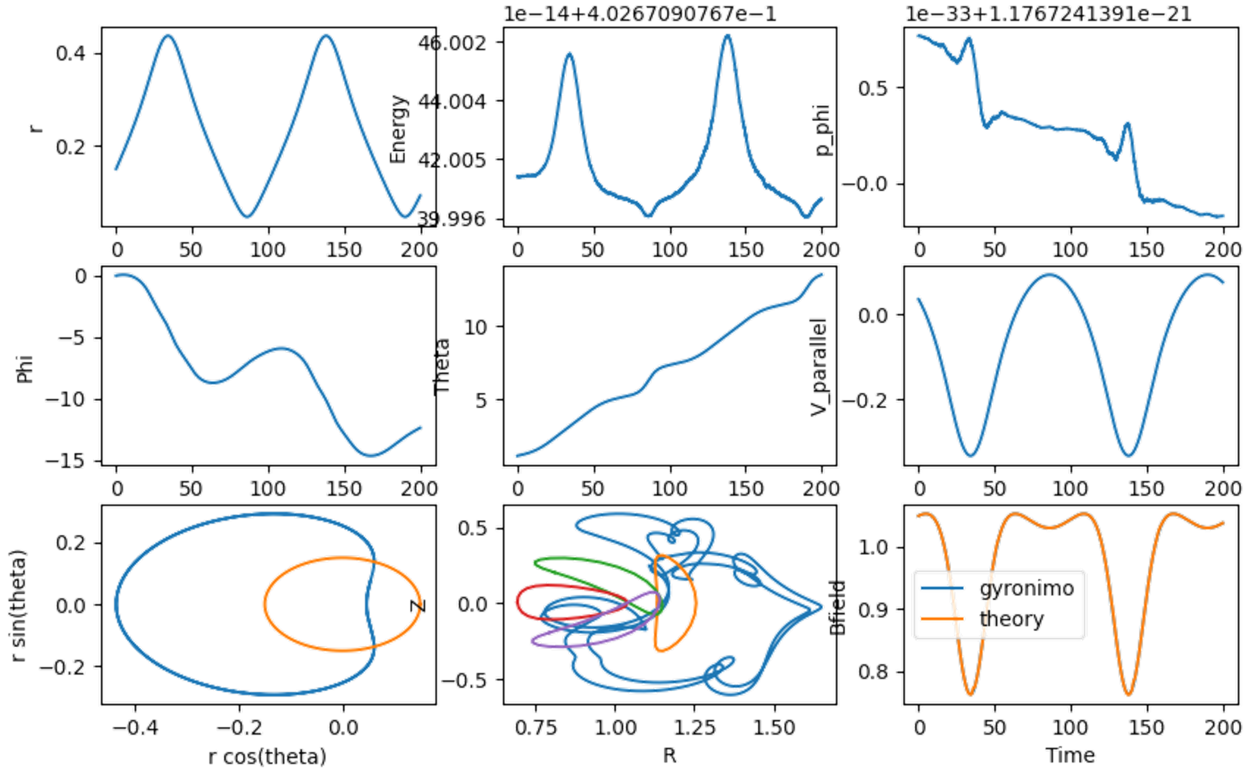


Preliminary Results

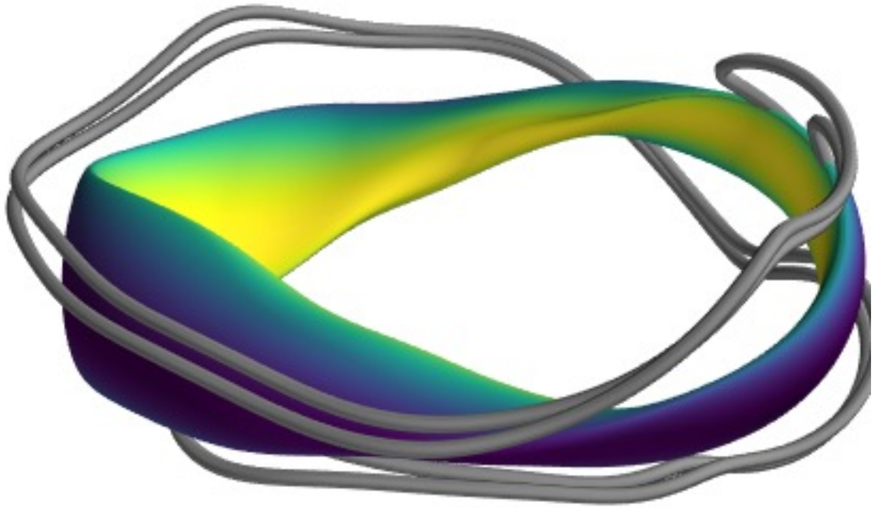
WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓



QA Stellarator

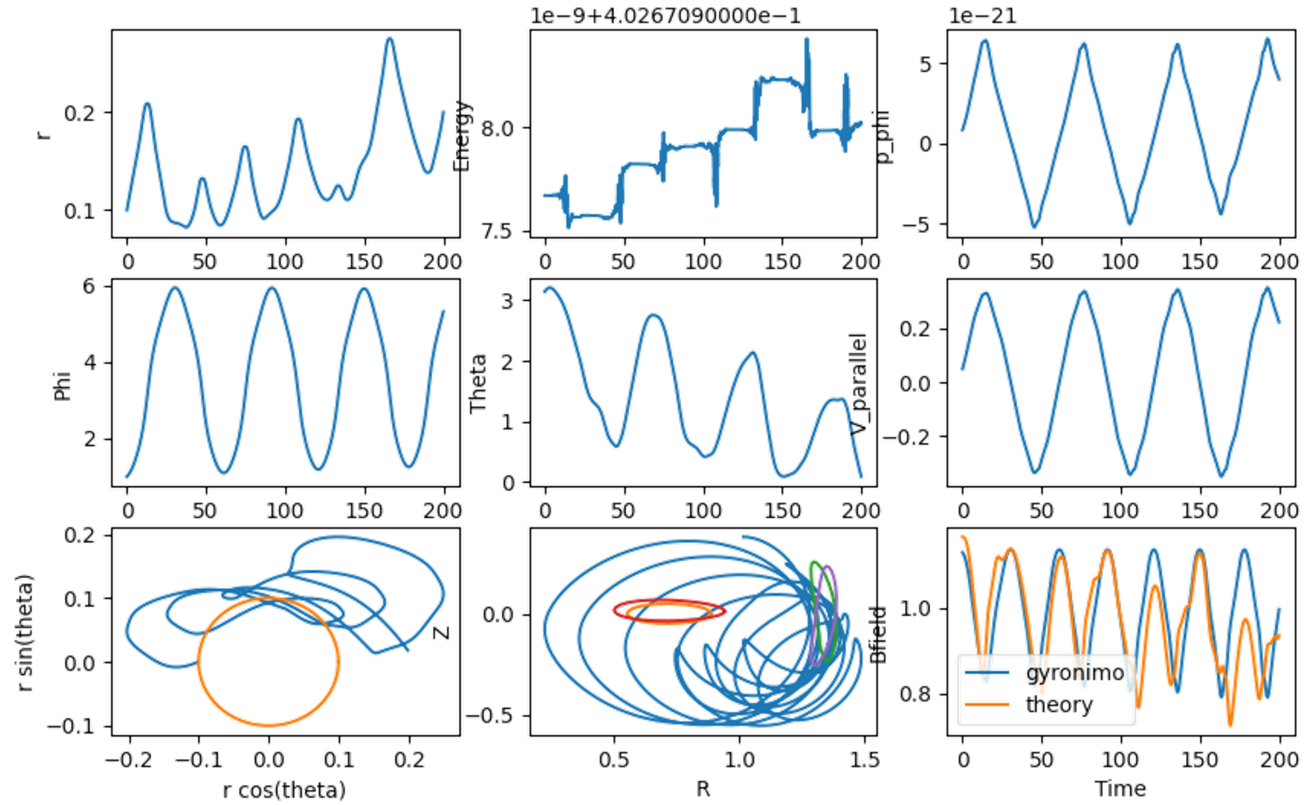


Preliminary Results

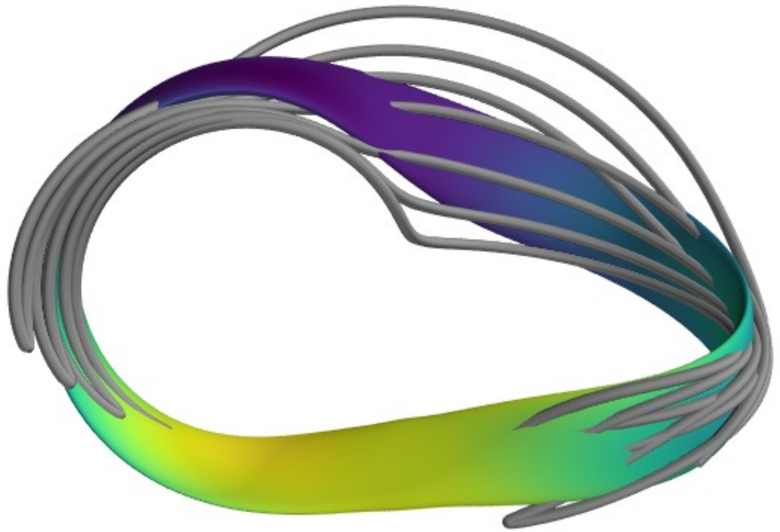
WP1

C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓



QI Stellarator



Preliminary Results

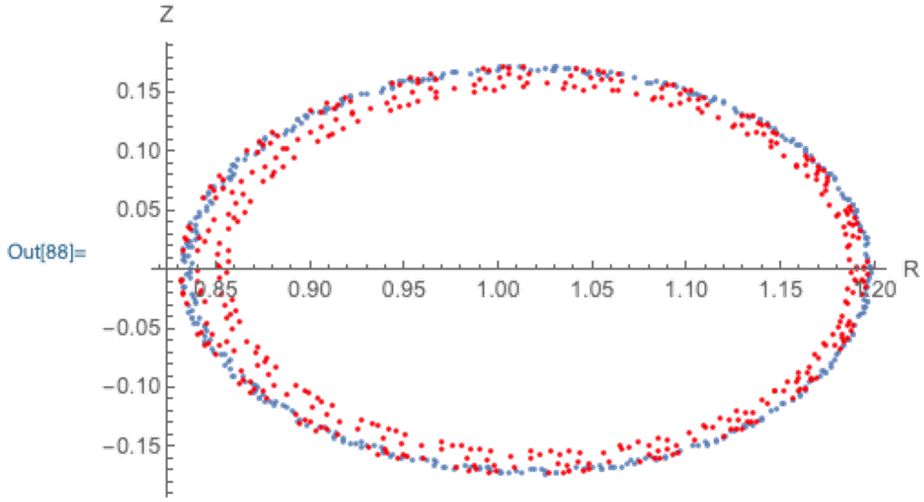
WP1

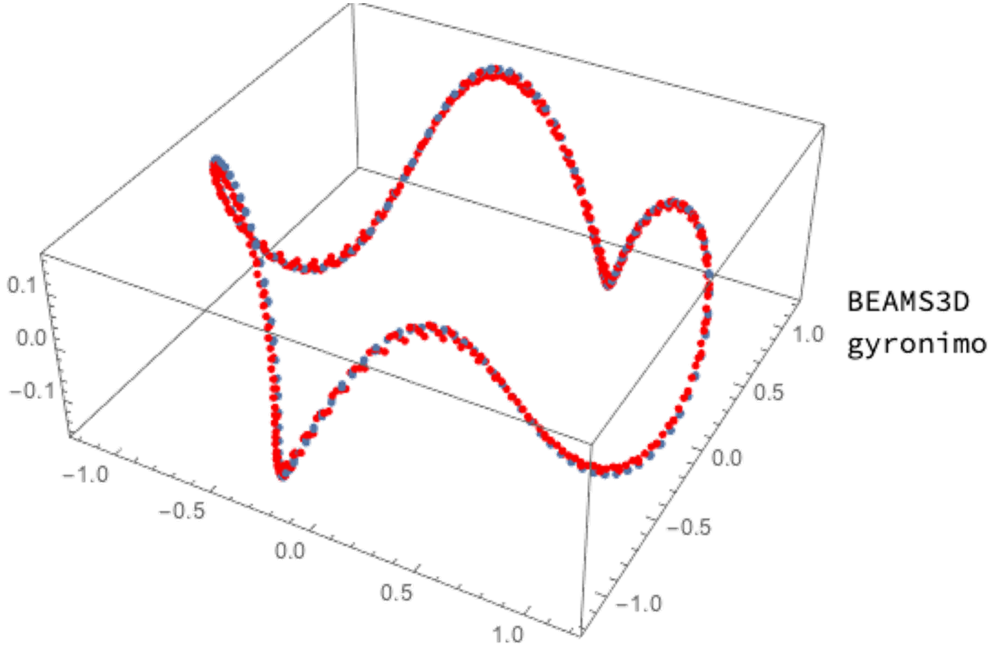
C++ Library with documentation on GitHub ✓

Python interface with near-axis expansion coordinates ✓

C++ interface with VMEC equilibria

Benchmark with other particle tracer codes such as BEAMS3D ✓





Need to extend to 10^5 particles

Preliminary Results

WP2

Analytical/numerical objective function to measure fast particle confinement

Python interface with SIMSOPT/STELLOPT ✓

SIMSOPT requires a *get*_dofs and a set_*dofs* function

In pyQSC

```
def get_dofs(self):
    """
    Return a 1D numpy vector of all possible optimizable
    degrees-of-freedom, for simsopt.
    """
    return np.concatenate((self.rc, self.zs, self.rs, self.zc,
                           np.array([self.etabar, self.sigma0, self.B2s, self.B2c,
                           self.p2, self.I2, self.c0, self.alpha0, self.delta]),
                           self.B0_vals, self.d_cvals, self.d_svals, self.alpha_cvals, self.alpha_svals)
```

```
def set_dofs(self, x):
    """
    For interaction with simsopt, set the optimizable degrees of
    freedom from a 1D numpy vector.
    """
    self.rc = x[self.nfourier * 0 : self.nfourier * 1]
    self.zs = x[self.nfourier * 1 : self.nfourier * 2]
    self.rs = x[self.nfourier * 2 : self.nfourier * 3]
    self.zc = x[self.nfourier * 3 : self.nfourier * 4]
    self.etabar = x[self.nfourier * 4 + 0]
    self.sigma0 = x[self.nfourier * 4 + 1]
    self.B2s = x[self.nfourier * 4 + 2]
    self.B2c = x[self.nfourier * 4 + 3]
    self.p2 = x[self.nfourier * 4 + 4]
    self.I2 = x[self.nfourier * 4 + 5]
```

Full script in:

https://github.com/rogeriojorge/NearAxis_Optimization

Deliverables for 2022

New open-source code

- Gyronimo (github.com/prodrigs/gyronimo)
- Extensive documentation and CI/CD

Scientific publications

- Gyronimo oriented study – convergence and verification studies
- Orbit differences between different optimized stellarators

Conference presentations

- EPS 2022
- Varenna 2022

Possible Revisions for 2022

Parallelization

Original proposal accounted for both OpenMP and CUDA in the first year

In the first year, only OpenMP may be implemented

Additional code development in 2023

Orbits within coils

Original proposal only had studies with coils at the end of the project

Magnetic fields from Biot-Savart projected to be added in mid-2022

However, realistic coil simulations only in 2023