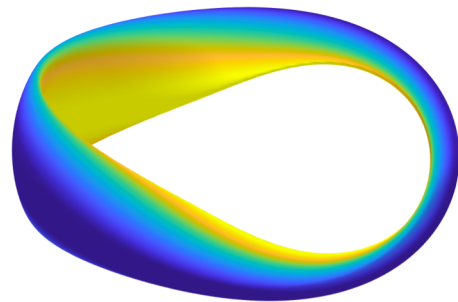
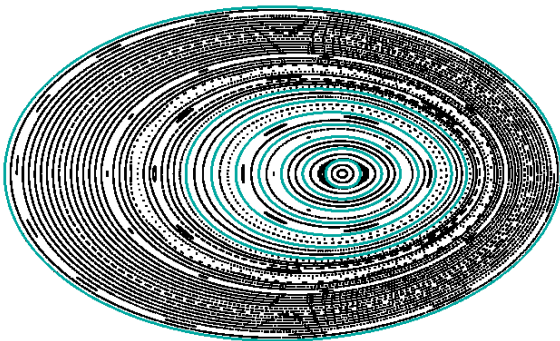
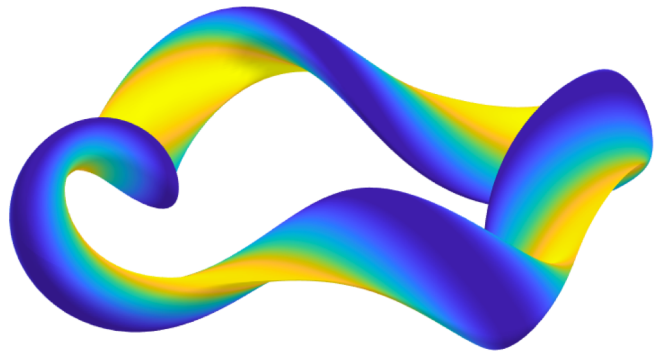


# Update on the simsopt stellarator optimization framework

Simons Hour, April 29, 2021



Matt Landreman, University of Maryland

Australian National University: Zhisong Qu

Cornell: David Bindel, Misha Padidar

EPFL: Antoine Bailod, Joaquim Loizu

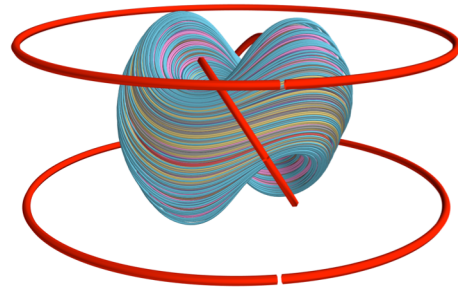
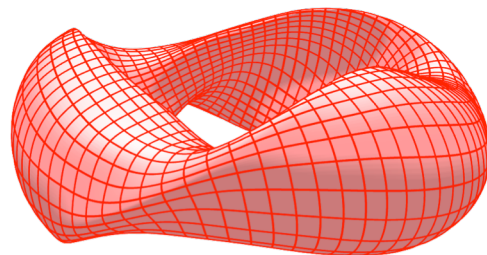
IPP: Jonathan Schilling

Maryland: Rogerio Jorge

NYU: Andrew Giuliani, Florian Wechsung

PPPL: Stuart Hudson, Bharat Medasani, Caoxiang Zhu

Wisconsin: Aaron Bader, Ben Faber, Thomas Kruger



- Vision
- Examples
- Design
- Next steps



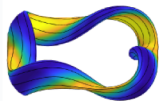
# Simsopt vision

- New stellarator optimization software that brings together expertise & code from the whole team & friends.
- Extensibility: It should be possible to add new codes and terms to an objective function without editing the core infrastructure or build system. Any edits to working code can potentially introduce bugs.
- Modularity: Physics modules that are not needed for your optimization problem do not need to be installed.
- Flexibility: Components can be used for many purposes, not just standard optimization.
- Thorough unit testing, regression testing, and continuous integration.

Project with similar goals at Wisconsin: Plasma Equilibrium Toolkit & LASSO.

# Simsopt components

- Interfaces to physics codes
- Surface & curve objects, with several parameterizations
- Tools for defining objective function & parameter space, e.g. fixed vs free degrees of freedom
- Fast Biot-Savart implementation, with derivatives
- Parallelized finite differences
- Plotting & graphics



# The Hidden Symmetries and Fusion Energy Collaboration

<https://hiddensymmetries.princeton...>

Repositories 7 Packages People 15 Teams Projects Settings

Type ▾

Language ▾

Sort ▾

Customize pins

New

## simsopt

Simons Stellarator Optimizer Code

plasma optimization fusion plasma-physics nuclear-fusion  
stellarator stellarators

Python LGPL-3.0 7 5 0 0 Updated 16 hours ago



## booz\_xform

Calculates Boozer coordinates for toroidal MHD equilibria, including stellarators and tokamaks.

C++ BSD-2-Clause 0 1 0 0 Updated 2 days ago



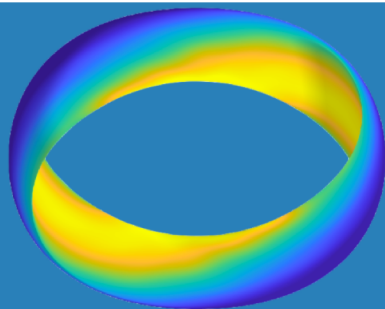
### Top languages

- C++ Python Fortran
- Jupyter Notebook

### People

15 >





latest

Search docs

## CONTENTS

- ▢ Getting started
- ▢ Concepts
- ▢ Defining optimization problems
- ▢ Testing

Source code on GitHub

## EXAMPLES

- ▢ Optimizing an equilibrium code
- Optimizing for quasisymmetry
- Eliminating magnetic islands

📖 Read the Docs

v: latest ▼

# Simsopt documentation

`simsopt` is a system for optimizing [stellarators](#). The high-level routines are in python, with calls to C++ or fortran where needed for performance. Several types of components are included:

- Interfaces to physics codes, e.g. for MHD equilibrium.
- Tools for defining objective functions and parameter spaces for optimization.
- Geometric objects that are important for stellarators – surfaces and curves – with several available parameterizations.
- An efficient implementation of the Biot-Savart law, including derivatives.
- Tools for parallelized finite-difference gradient calculations.

Some of the physics modules with compiled code reside in separate repositories. These separate modules include

- [VMEC](#), for MHD equilibrium.
- [SPEC](#), for MHD equilibrium. (This repository is private.)
- [booz\\_xform](#), for Boozer coordinates and quasisymmetry.

The design of `simsopt` is guided by several principles:

- Thorough unit testing, regression testing, and continuous integration.
- Extensibility: It should be possible to add new codes and terms to the objective function without editing modules that already work, i.e. the [open-closed principle](#). This is because any edits to working code can potentially introduce bugs.
- Modularity: Physics modules that are not needed for your optimization problem do not need to be installed. For instance, to optimize SPEC equilibria, the VMEC module need not be installed.

# How to get involved

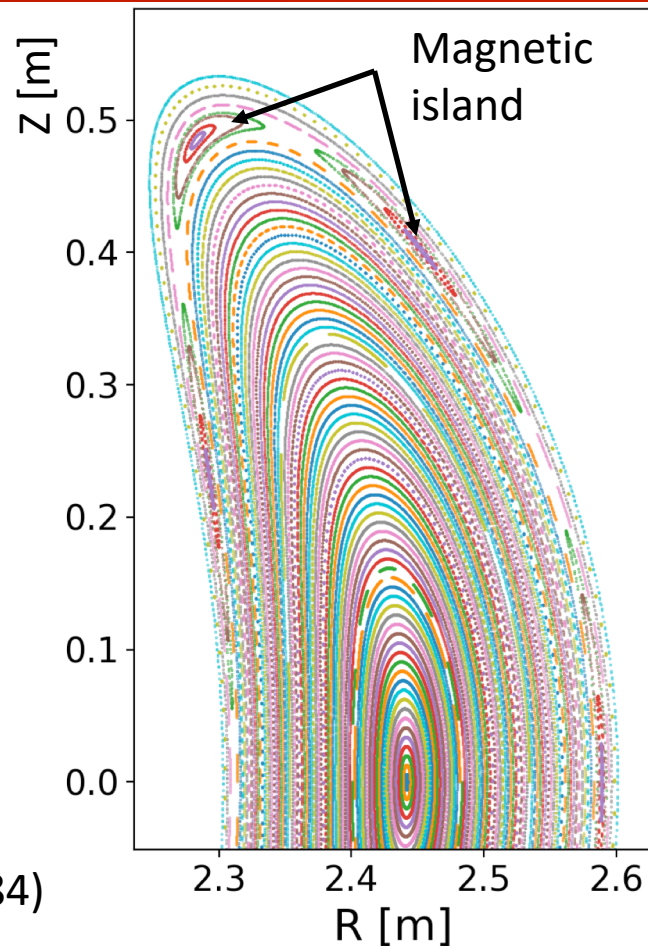
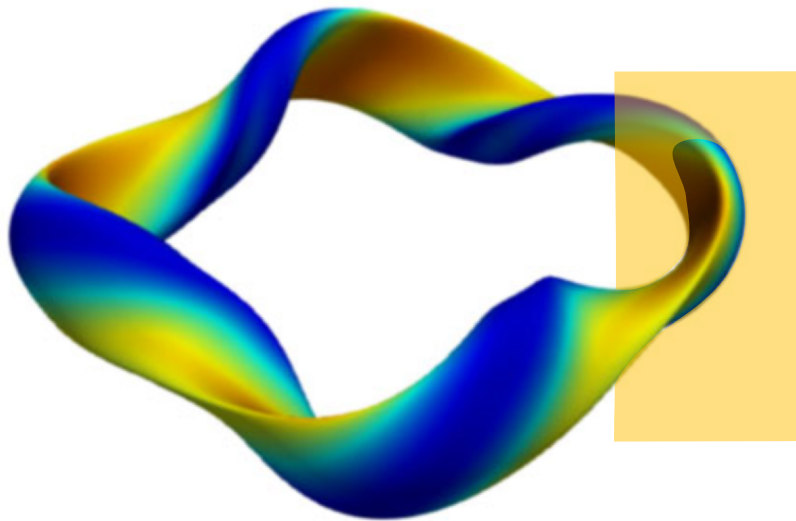
- <https://github.com/hiddenSymmetries/simsopt>
- Everyone is welcome at development meetings: Mondays @ 9am NY, 3pm Europe
- simsopt slack workspace, #developers channel
- Play around with the code and examples
- Fork the repository, add a feature, submit pull request



- Vision
- Examples
- Design
- Next steps

# Simsopt can optimize SPEC configurations to eliminate magnetic islands

Starting point: a quasi-helically symmetric configuration from Wisconsin [Bader et al (2020)]



We'll minimize Greene's residue, similar to Hanson & Cary (1984)

# Islands can be eliminated by optimizing Greene's residue with simsopt using a high-level python script

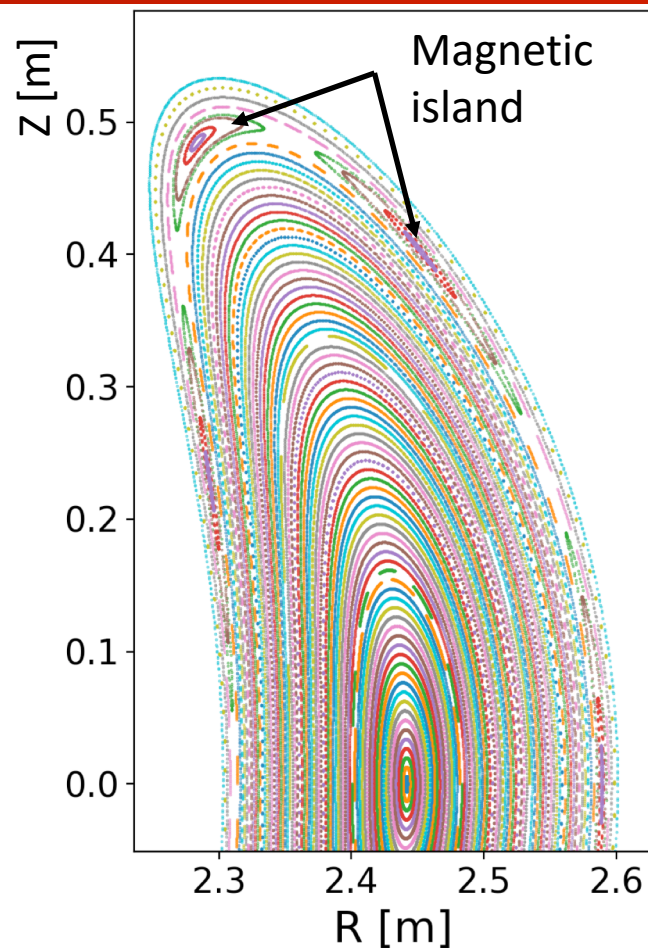
```
# Create an equilibrium object to optimize:
spec = Spec('QH-residues.sp')
spec.boundary.change_resolution(mpol=12, ntor=12)

# Define the parameter space for optimization:
spec.boundary.all_fixed()
spec.boundary.set_fixed('zs(6,1)', False)

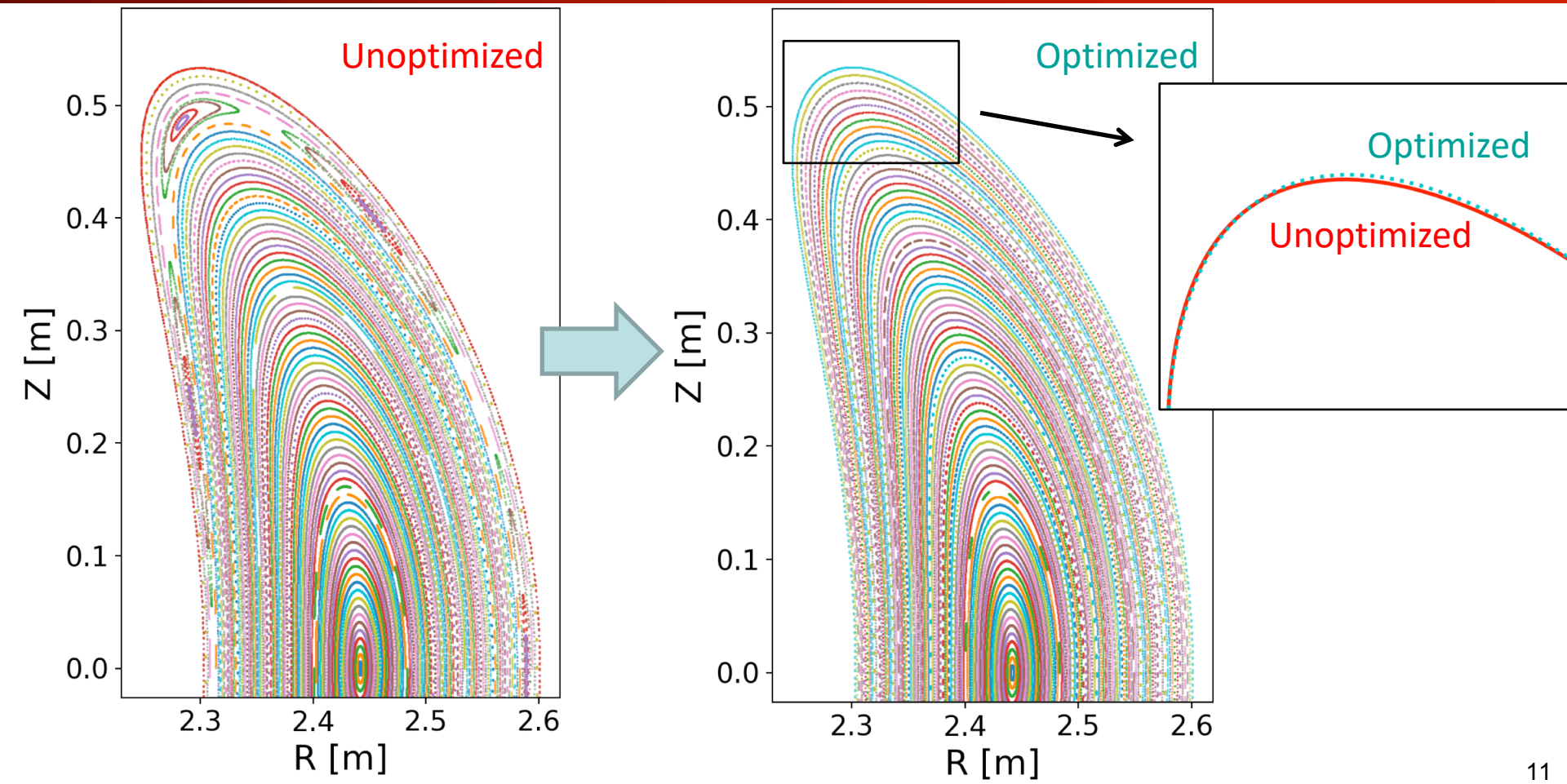
# Main island chain has helicity  $iota = 8/7$ :
residue1 = Residue(spec, 8, 7)
residue2 = Residue(spec, 12, 11)

# Objective function is  $\sum_j \text{residue}_j ** 2$ 
prob = LeastSquaresProblem([(residue1, 0, 1),
                             (residue2, 0, 1)])

least_squares_serial_solve(prob)
```



# Optimization of boundary shape successfully results in good surfaces

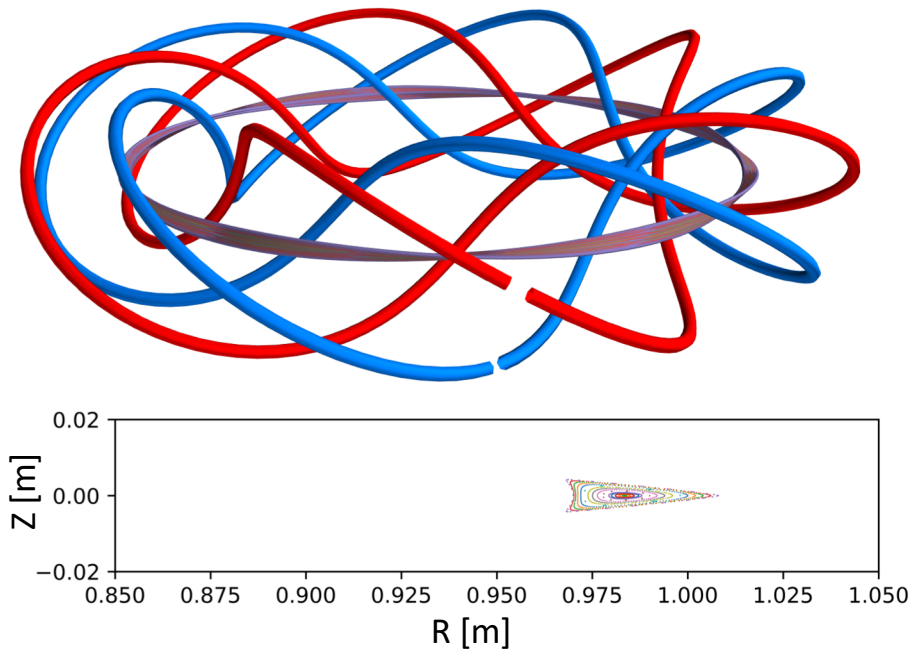




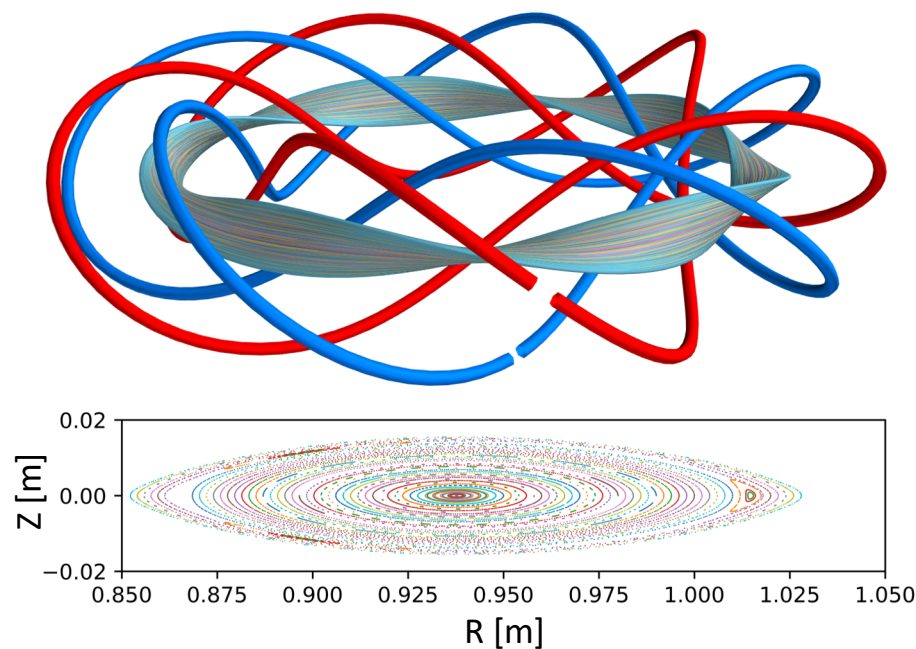
# Optimization for good surfaces can also be done in the parameter space of coil shapes rather than the boundary shape

Reproduction of Cary & Hanson (1986) with simsopt by Rogerio Jorge

Unoptimized



Optimized

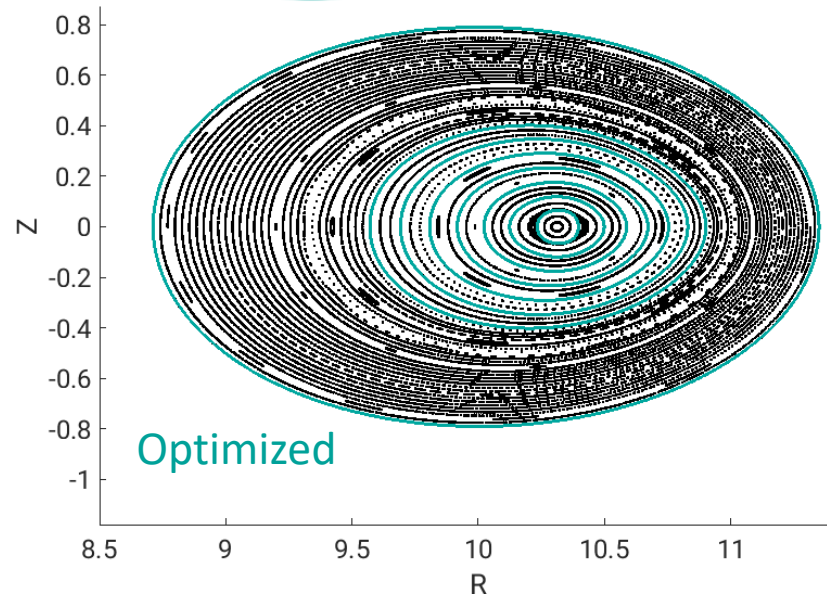
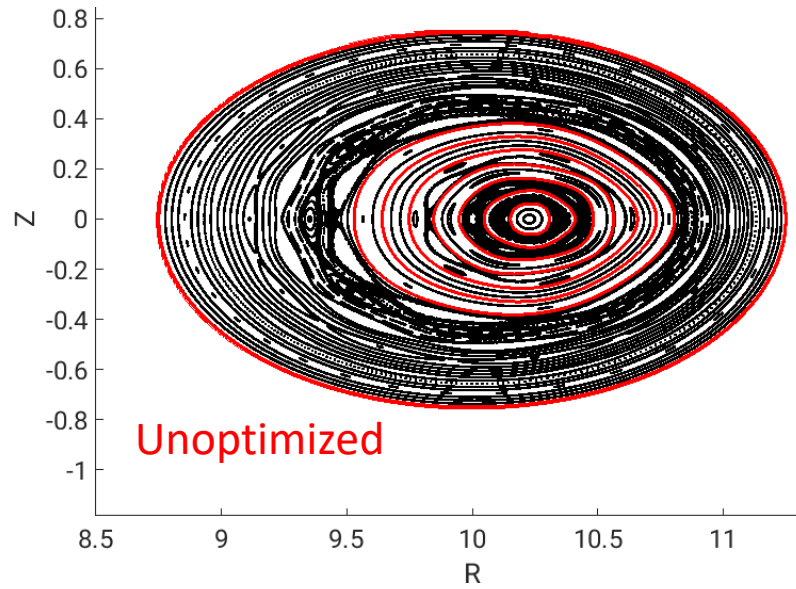
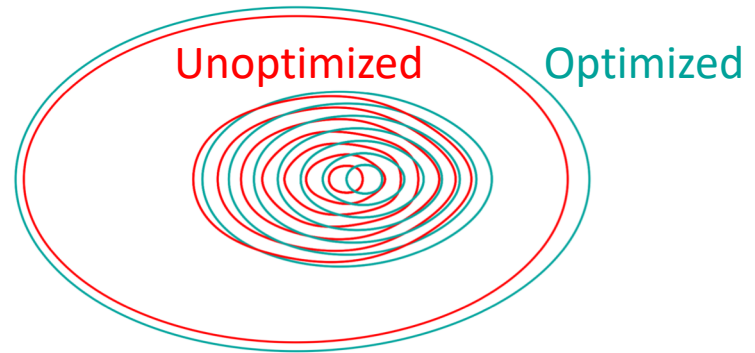
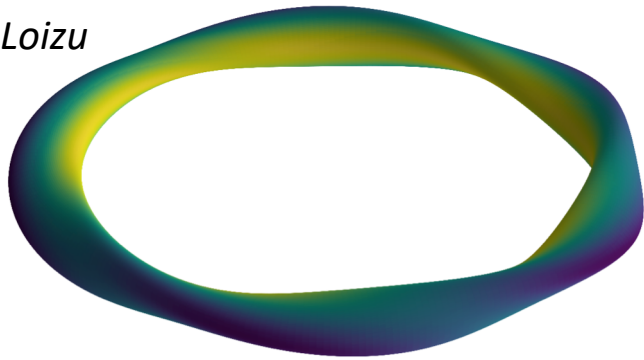


Good testbed for converse KAM, Mather's  $\Delta W$ , etc



# We are now optimizing for integrability with plasma current & pressure

*Bailod & Loizu*



# Like STELOPT & ROSE, SIMSOPT can optimize VMEC configurations for quasisymmetry

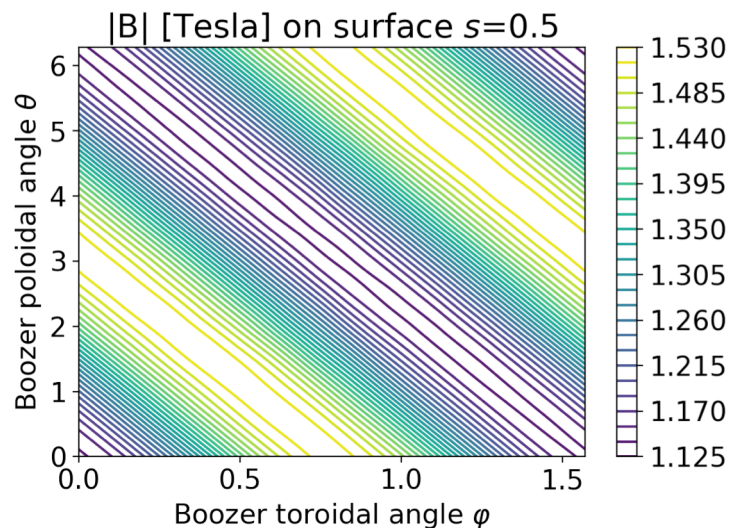
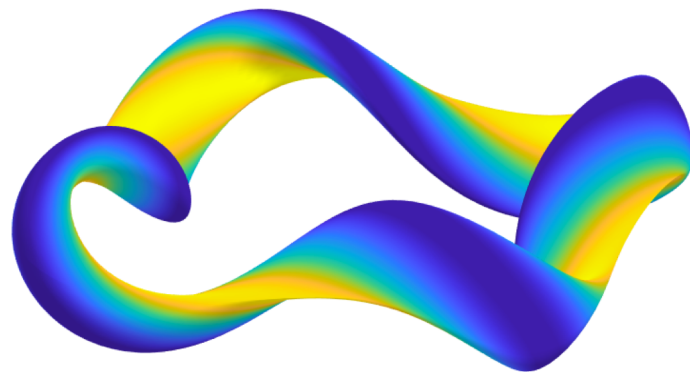
```
mpi = MpiPartition()
vmec = Vmec("input.nfp4_QH", mpi)

# Define parameter space:
surf = vmec.boundary
surf.all_fixed()
surf.fixed_range(mmin=0, mmax=3,
                 nmin=-3, nmax=3, fixed=False)
surf.set_fixed("rc(0,0)") # Average major radius

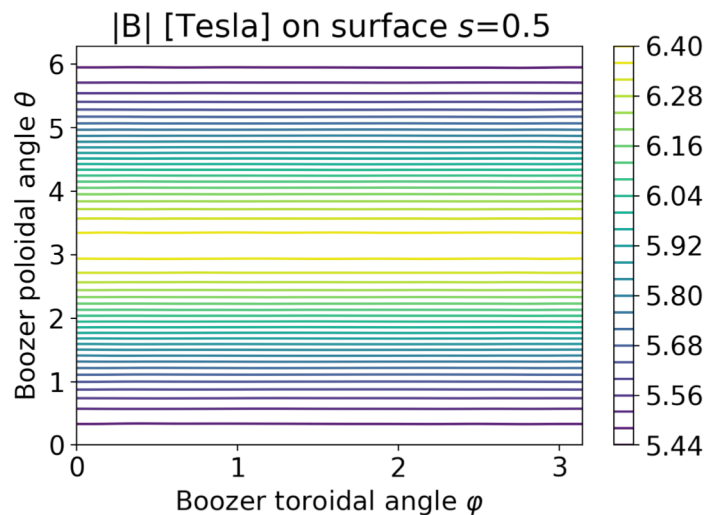
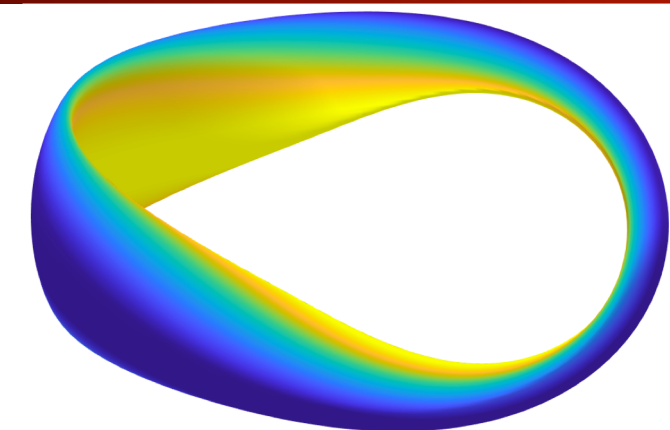
# Set parameters for quasisymmetry objective:
qs = Quasisymmetry(Boozer(vmec),
                   0.5, # Radius to target.
                   1, 1) # (M, N) you want in |B|.

# Define objective function:
prob = LeastSquaresProblem([(qs, 0, 1),
                           (vmec.aspect, 7, 1)])

least_squares_mpi_solve(prob, mpi, grad=True)
```



# Scripting allows dynamic increase in resolution during optimization



```
mpi = MpiPartition()
vmec = Vmec("input.nfp2_QA", mpi=mpi)
surf = vmec.boundary

# Configure quasisymmetry objective:
boozer = Boozer(vmec)
qs = Quasisymmetry(boozer, 0.5, # Radius to target
                   1, 0) # (M, N) you want in |B|

# Define objective function:
prob = LeastSquaresProblem([(vmec.aspect, 6, 1),
                             (vmec.iota_axis, 0.465, 1),
                             (vmec.iota_edge, 0.495, 1),
                             (qs, 0, 1)])

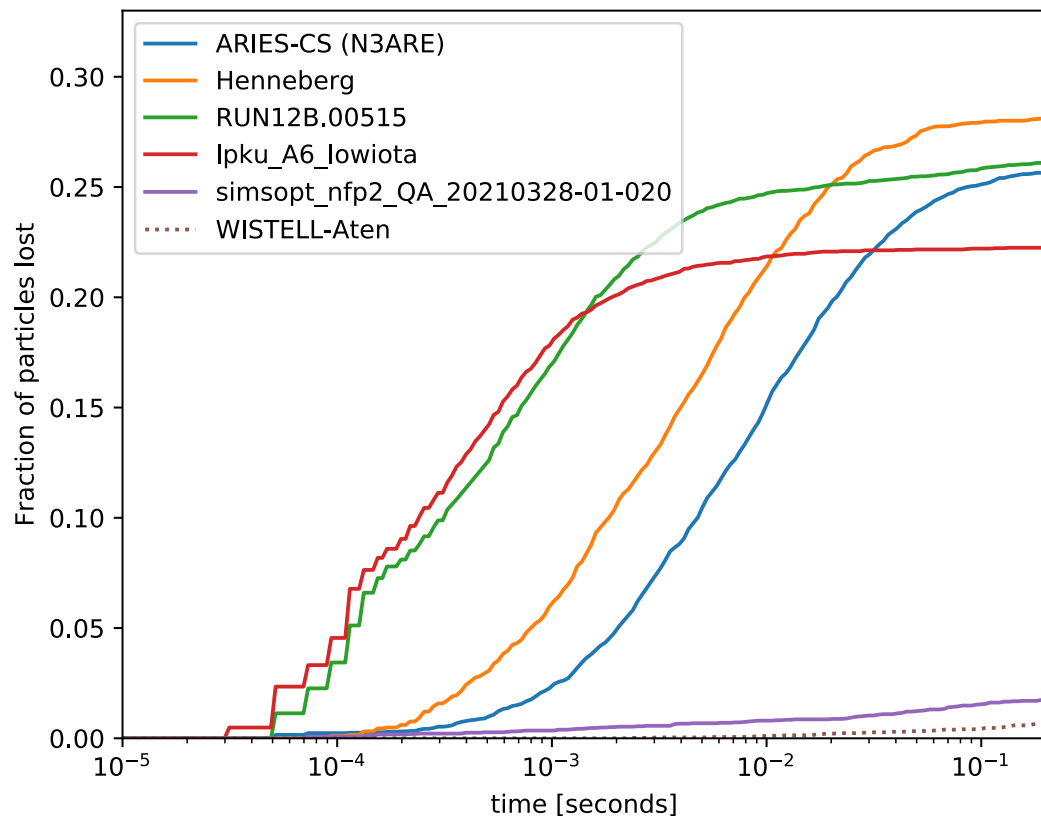
for step in range(4):
    vmec.indata.mpol = 3 + step # Increase resolution
    vmec.indata.ntor = vmec.indata.mpol
    boozer.mpol = 16 + step * 8 # Increase resolution
    boozer.ntor = boozer.mpol

    # Parameter space gets larger each step:
    surf.all_fixed()
    max_mode = step + 1
    surf.fixed_range(mmin=0, mmax=max_mode,
                    nmin=-max_mode, nmax=max_mode,
                    fixed=False)
    surf.set_fixed("rc(0,0)") # Major radius

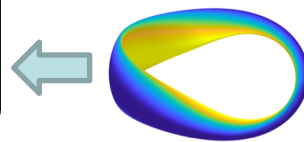
    least_squares_mpi_solve(prob, mpi, grad=True)
```

# This QA configuration from simsopt has decent alpha particle confinement

Collisionless losses of alpha particles for particles born at radius  $s = 0.3$



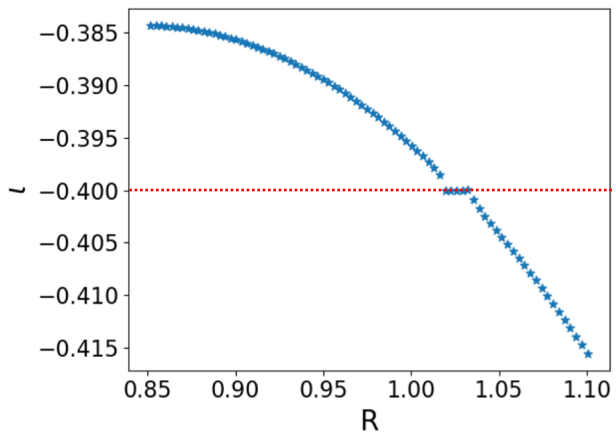
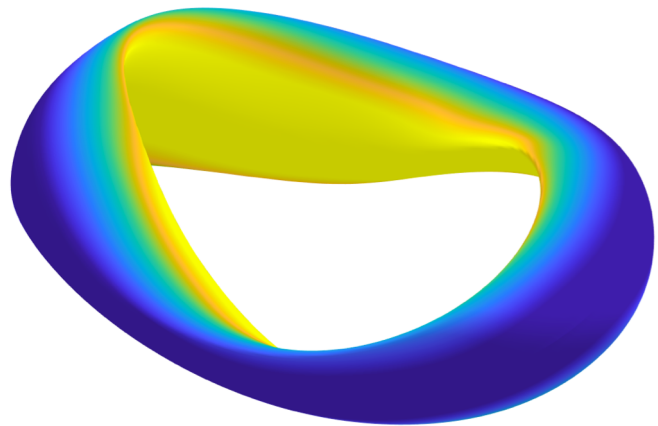
(Unfair: Haven't checked  
MHD stability or coils.)



Configurations scaled to the ARIES-CS volume and average  $|B|$ .

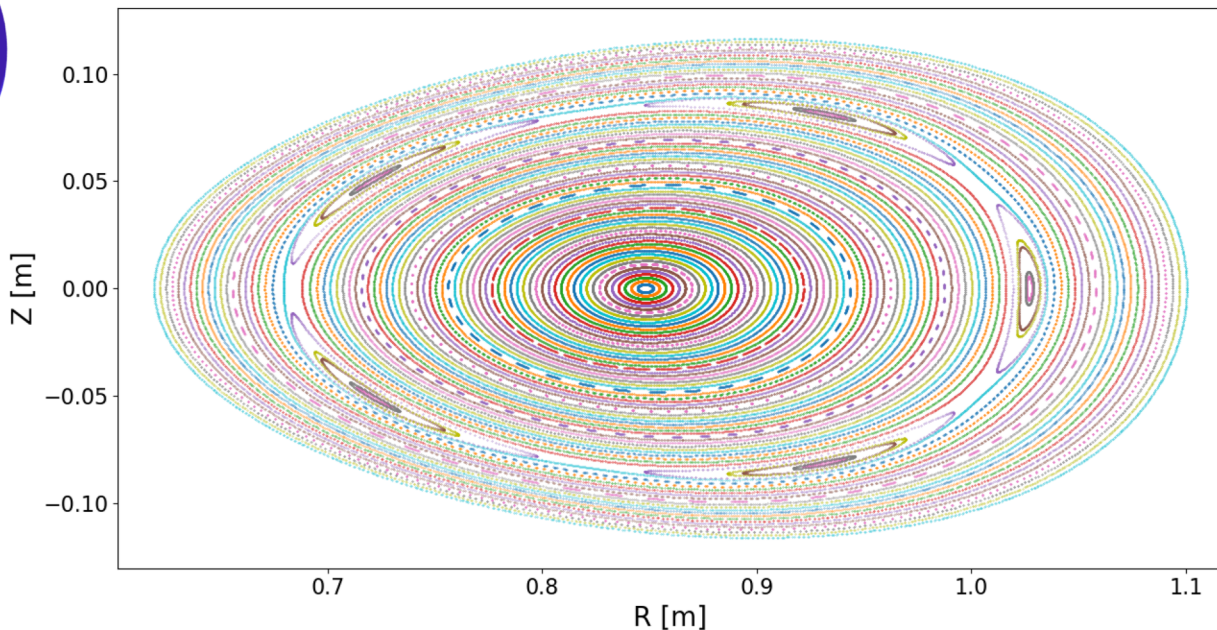
With Aaron Bader, Sophia Henneberg, Neil Pomphrey

Simsopt can also optimize for quasisymmetry & good surfaces simultaneously,  
with both VMEC and SPEC in the loop.



Starting point: nfp=2, QA, aspect = 6,

Island chain at iota =  $2/5 = 0.4$





## Simsopt driver script applied:

SPEC told to use the same boundary surface object as VMEC.

```
vmec = Vmec("input.nfp2_QA")
surf = vmec.boundary

spec = Spec("nfp2_QA.sp")
spec.boundary = surf

# Define parameter space:
surf.all_fixed()
surf.fixed_range(mmin=0, mmax=4,
                 nmin=-3, nmax=3, fixed=False)
surf.set_fixed("rc(0,0)") # Major radius

# Configure quasisymmetry objective:
qs = Quasisymmetry(Boozer(vmec),
                   0.5, # Radius s to target
                   1, 0) # (M, N) you want in |B|

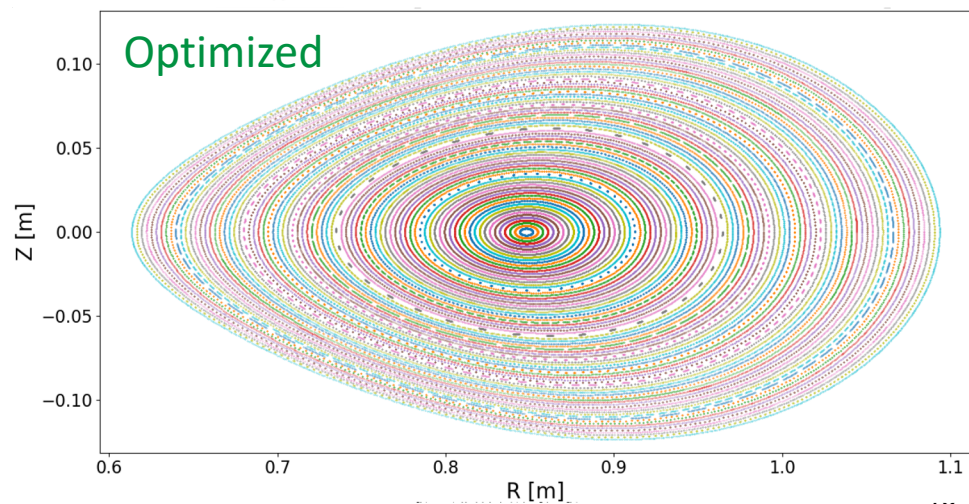
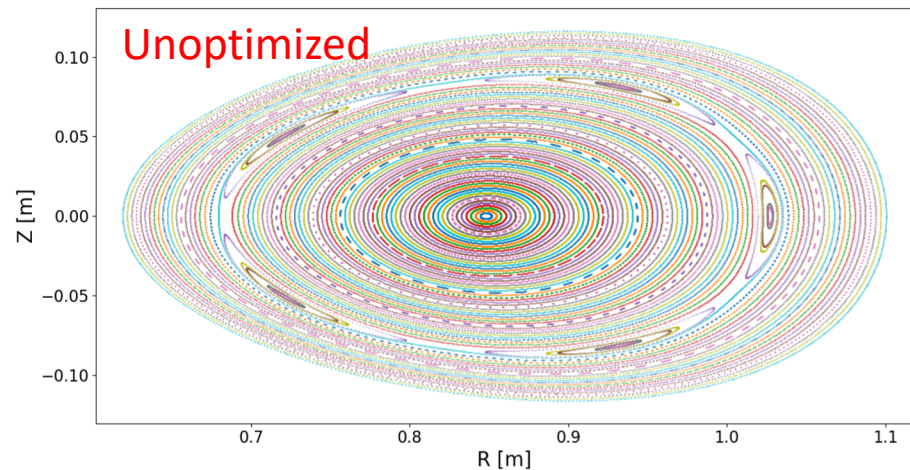
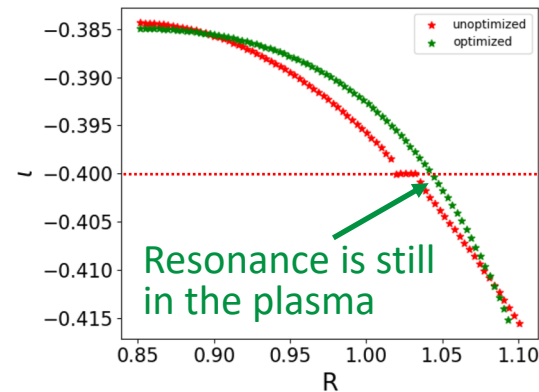
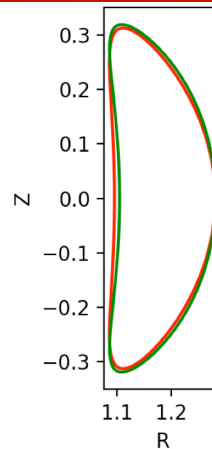
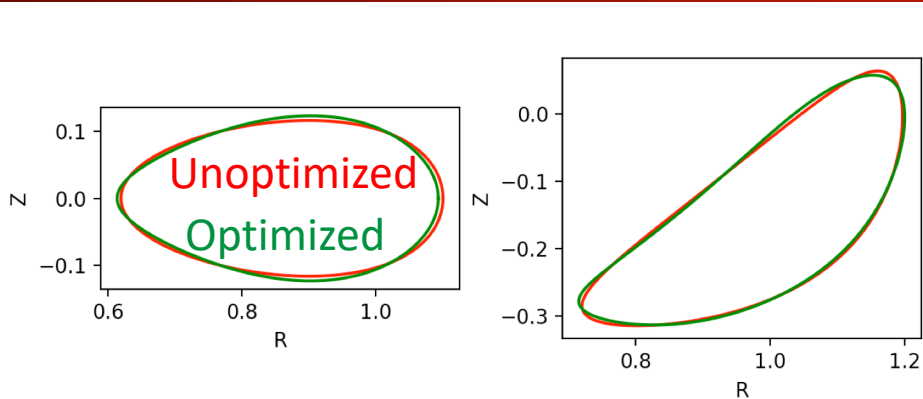
# Specify resonant iota = p / q
p = -2; q = 5
residue1 = Residue(spec, p, q)
residue2 = Residue(spec, p, q, theta=np.pi)

# Define objective function
prob = LeastSquaresProblem([(vmec.aspect, 6, 1),
                             (vmec.iota_axis, 0.385, 1),
                             (vmec.iota_edge, 0.415, 1),
                             (qs, 0, 1),
                             (residue1, 0, 2),
                             (residue2, 0, 2)])

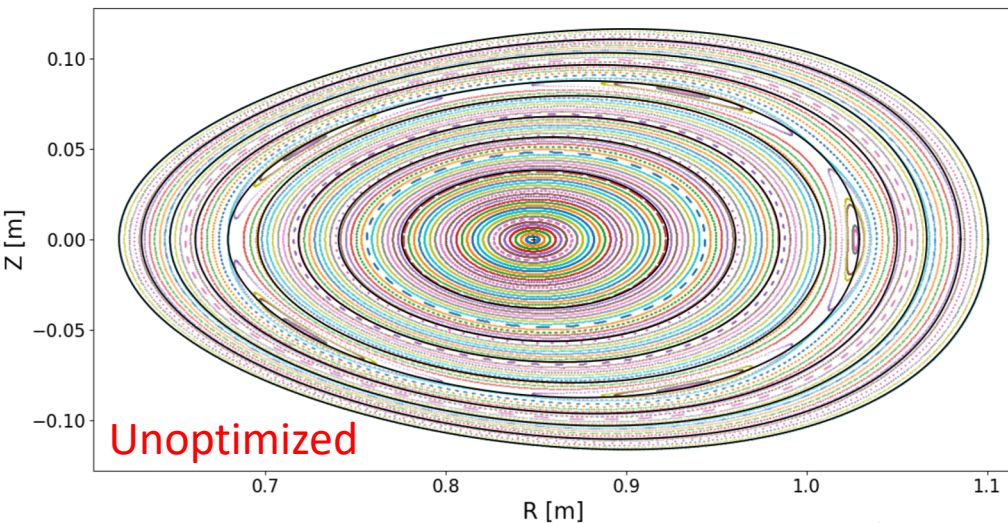
least_squares_serial_solve(prob)
```

Objective function includes both quasisymmetry from VMEC + booz\_xform and residues from SPEC + pyoculus.

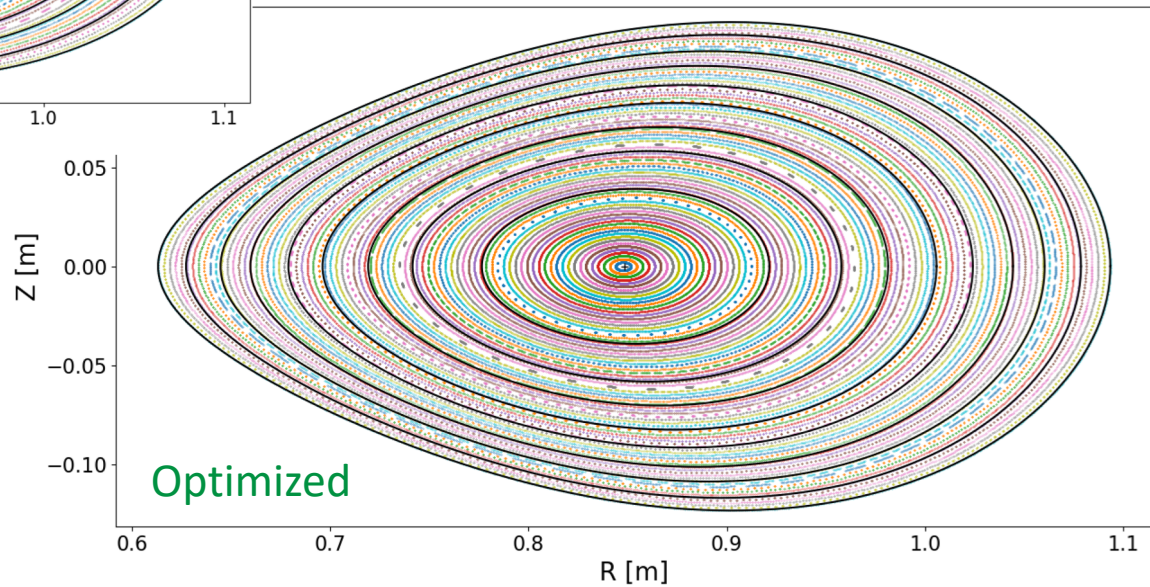
# The optimization eliminates the islands



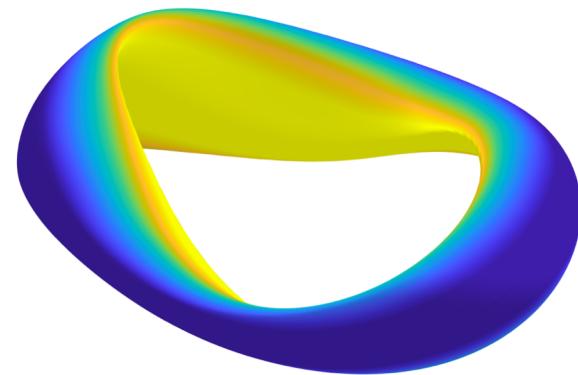
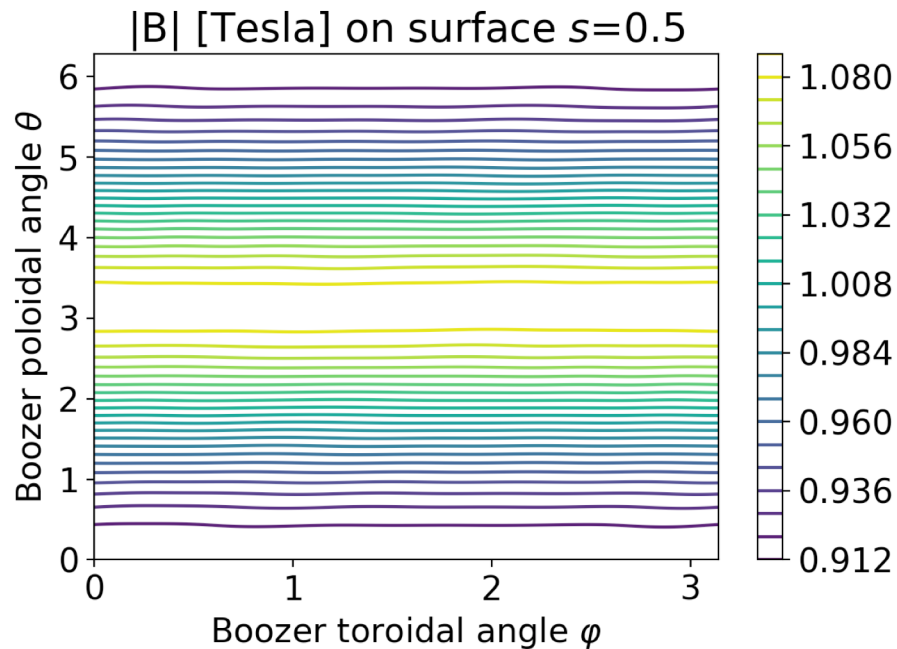
# VMEC & SPEC solutions match at the optimum



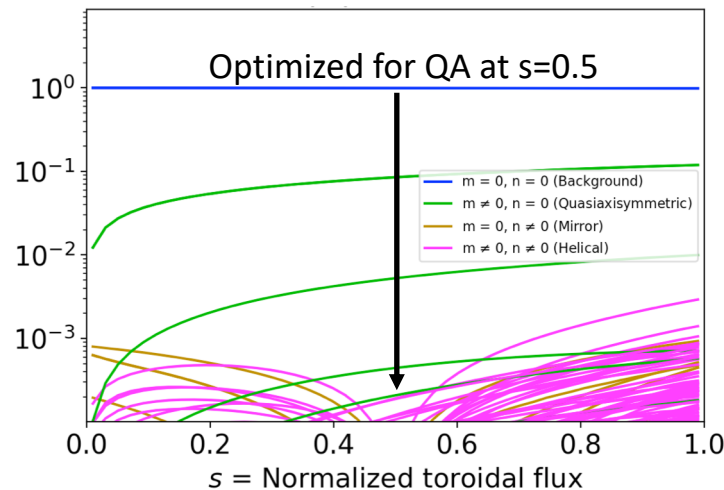
— VMEC  
... SPEC



# Quasisymmetry is simultaneously improved during the optimization

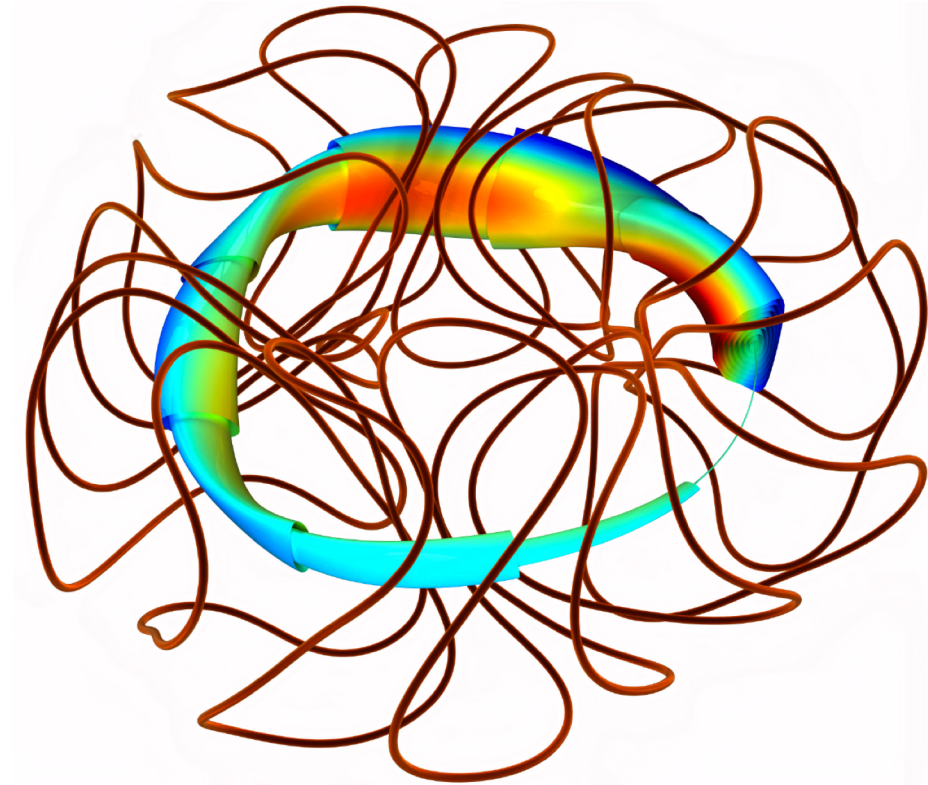


Modes of  $|B|$  in Boozer coordinates

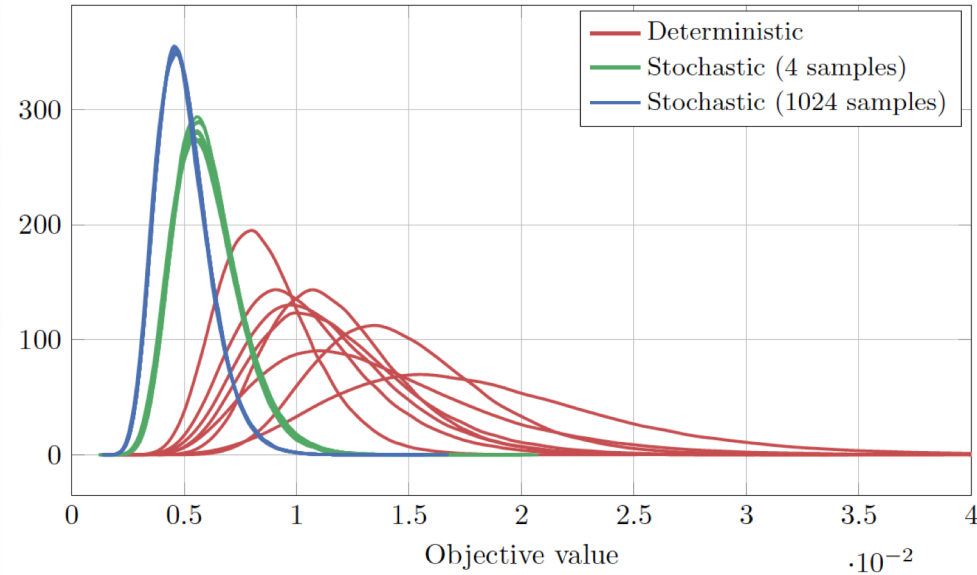




# Simsopt is used for the NYU derivative-based stochastic coil optimization



Distribution of objective function  
given coil shape errors



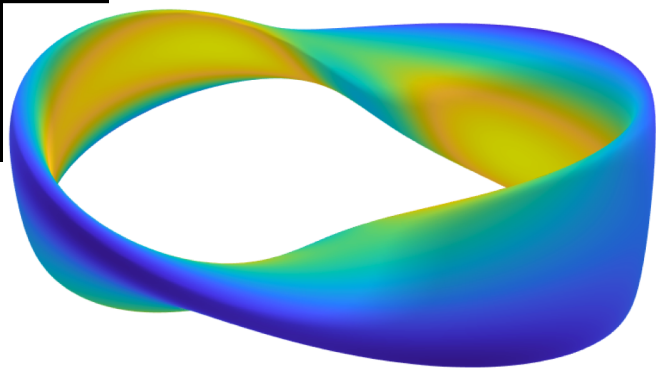


# Simsopt objective functions can be plugged into outside libraries

```
# E.g., Bayesian global optimization library
# from David Bindel's student:
from turbo import TurboM
from simsopt import LeastSquaresProblem, ...

# Define objective function as in previous slides:
...
prob = LeastSquaresProblem([(vmec.aspect, 6, 1),
                           (vmec.iota_axis, 0.465, 1),
                           (vmec.iota_edge, 0.495, 1),
                           (qs, 0, 1)])

turbo = TurboM(prob.objective, ...)
turbo.optimize()
```



- Vision
- Examples
- Design
- Next steps

# Design aspects

- Driver and infrastructure is in python.
- New compiled code is in C++, via pybind11.
- Fortran codes (e.g. VMEC, SPEC) interfaced via f90wrap.

# A variety of geometric objects and B field types have been implemented

## Curve subclasses:

*CurveXYZFourier, CurveRZFourier, CurveHelical,  
JaxCurveXYZFourier, RotatedCurve*

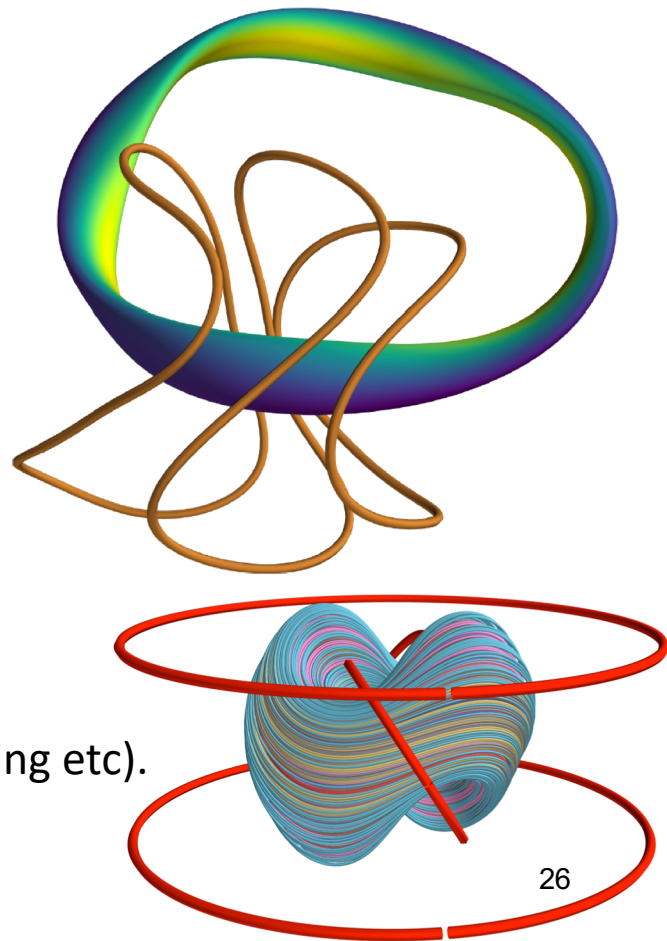
## Surface subclasses:

*SurfaceRZFourier, SurfaceGarabedian,  
SurfaceXYZFourier, SurfaceXYZTensorFourier*

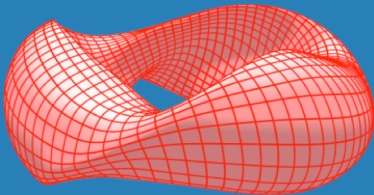

## MagneticField subclasses:

*BiotSavart, ToroidalField, CircularCoil, Dommaschk,  
ScalarPotentialRZMagneticField, MagneticFieldSum*

- All include 1 or 2 derivatives.
- All can include code in C++ (for speed) & python (for plotting etc).
- All have caching to avoid repeated calculations.
- Example with automatic differentiation is included.



# As an example of a new standalone physics module, `booz_xform` has been re-written



Search docs

## CONTENTS

- Getting started
- Theory and numerical implementation
- Typical usage
  - Plotting
- API Reference
- Developer notes
- Source code on GitHub

## booz\_xform documentation

`booz_xform` is a package for computing Boozer coordinates in toroidal magnetohydrodynamic equilibria, including both stellarators and tokamaks. The package described here follows the same algorithm as the fortran 77 code of the same name in [Stellopt](#). However the package here is written in C++, with python bindings. The package here is also written so as to allow input from equilibrium codes other than VMEC, it is parallelized using OpenMP, and it includes functions for plotting output. It is also equipped with unit and regression tests and continuous integration.

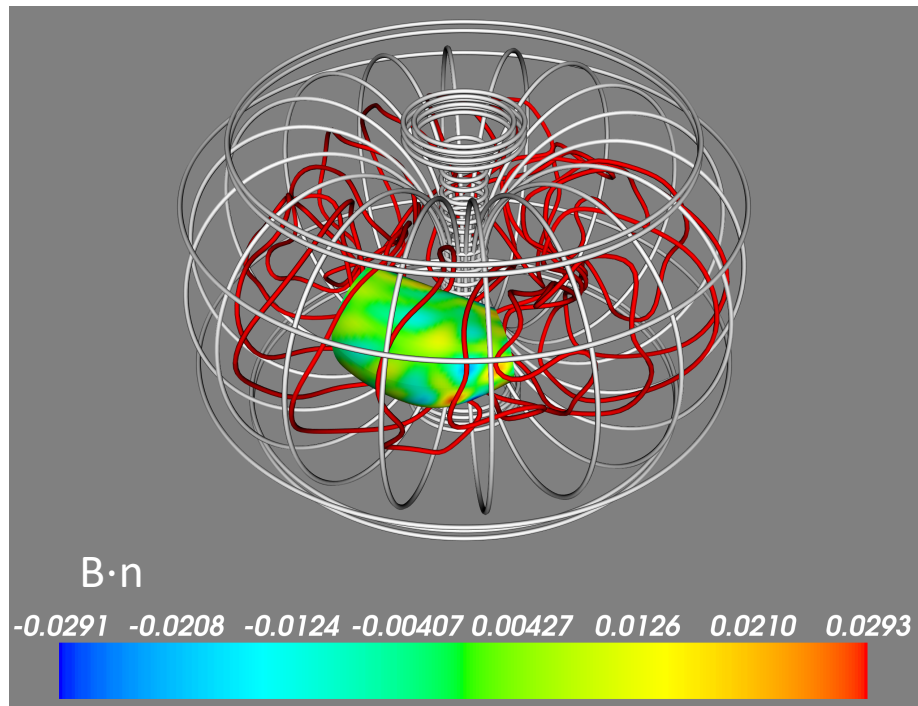
## Contents

- Getting started
  - Requirements
  - Installation
    - 1. Installation from PyPI
    - 2. Installation from a local copy of the repository
    - 3. Installation without pip from a local copy of the repository
    - 4. Building outside of the python package system
- Theory and numerical implementation
  - Theory
    - 1. Determining the toroidal angle difference
    - 2. Transforming other quantities
  - Implementation details



# Simsopt is being designed to handle both derivative-free and derivative-based problems

- Curves, surfaces, and magnetic fields all have 1-2 derivatives implemented.
- Curve types with automatic differentiation are implemented.
- Stage-2 coil optimization with derivatives (FOCUS) with simsopt classes works.
- Presently, simsopt handles naïve multiplication of Jacobians for the chain rule.
- In process of updating the system for defining problems with derivatives, to allow user-controlled forward vs reverse-mode chain rule.



# Comprehensive tests are automatically run after every push to GitHub

hiddenSymmetries / simsopt

Unwatch

3

Star

6

Fork

7

<> Code

Issues 7

Pull requests 6

Actions

Projects

Wiki

Security

Insights

Settings

✓ CI CI #579

Re-run jobs

Summary

Triggered via push 9 hours ago

Status

Total duration

Artifacts

landreman

48405c3

master

Success

15m 4s

-

Jobs

✓ CI (3.7, unit)

✓ CI (3.7, integrated)

✓ CI (3.8, unit)

✓ CI (3.8, integrated)

✓ CI (3.9, unit)

✓ CI (3.9, integrated)

ci.yml

on: push

Matrix: CI

✓ 6 jobs completed

Show all jobs

> 100 unit test cases

16 integrated tests

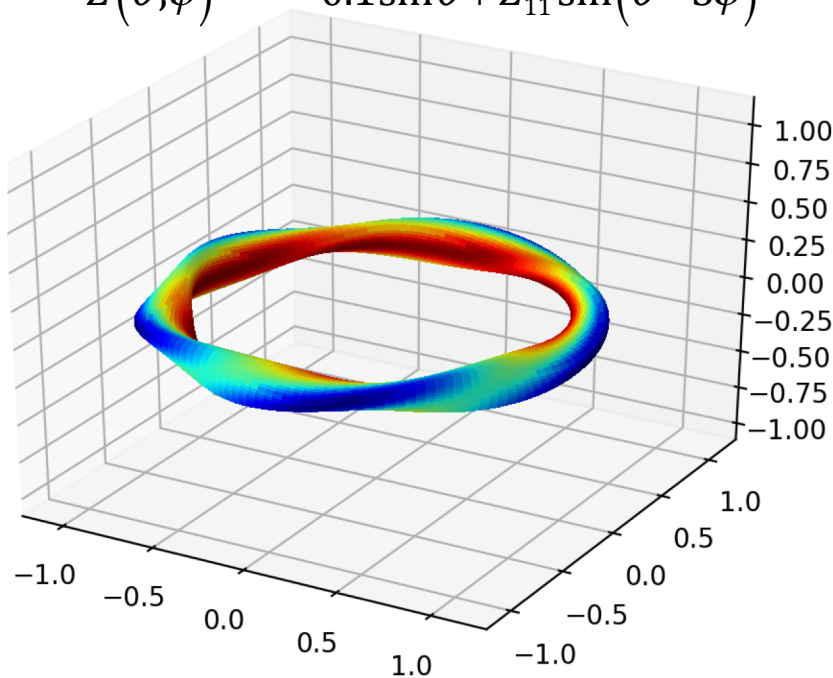
Several benchmark problems have been characterized.  
Simsomt+spec, simsopt+vmec, and stelopt give the same answer.

[github.com/landreman/stellopt\\_scenarios](https://github.com/landreman/stellopt_scenarios)

Boundary:

$$R(\theta, \varphi) = 1 + 0.1 \cos \theta + R_{11} \cos(\theta - 5\varphi),$$

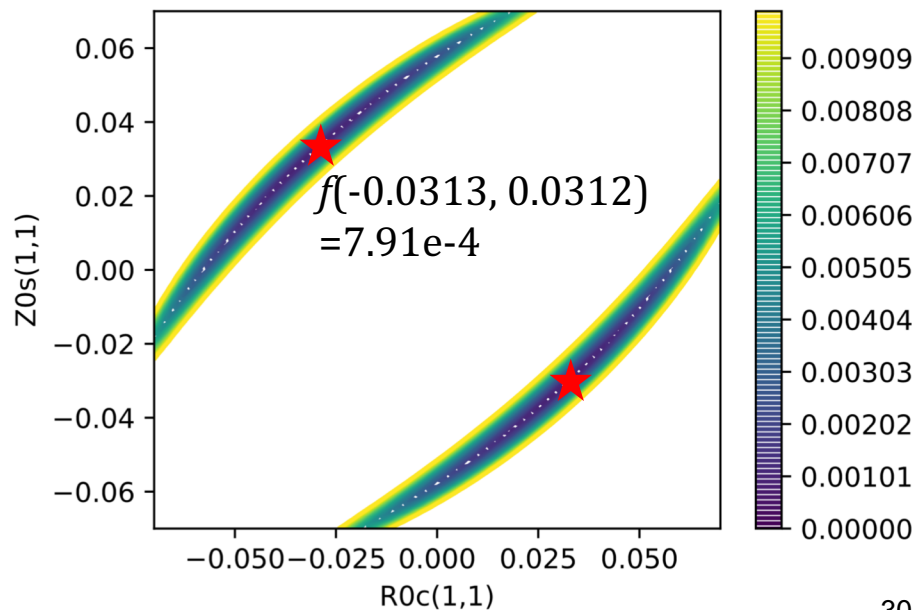
$$Z(\theta, \varphi) = 0.1 \sin \theta + Z_{11} \sin(\theta - 5\varphi)$$



Independent variables:  $\{R_{11}, Z_{11}\}$

Objective function:

$$f = (\iota_0 - 0.41)^2 + (\text{volume} - 0.15)^2$$



- Vision
- Examples
- Design
- Next steps

# There are many ways you could contribute

- Make sure the infrastructure can handle likely use cases
- Parallel global optimization algorithms
- Multi objective algorithms
- Code up adjoint problems by E Paul & A Geraldini
- Set up a container
- Write mgrid files, free boundary vmec
- Implement more measures of integrability
- Interface more physics objectives & codes
- Connect more equilibrium codes: GVEC, DESC, BIEST
- Interface NYU replacement for BNORM
- Guiding center trajectory integration
- Add more graphing tools



# Closing questions

Any suggested modifications to the structure?

What to prioritize next?

*Global optimization, multi-objective optimization, adjoints, ...*

What other use cases would you like to be able to handle?

Join the project!

- *Development meetings: Mondays @ 9am NY, 3pm Europe*
- *simsopt slack workspace, #developers channel*
- *mattland@umd.edu*