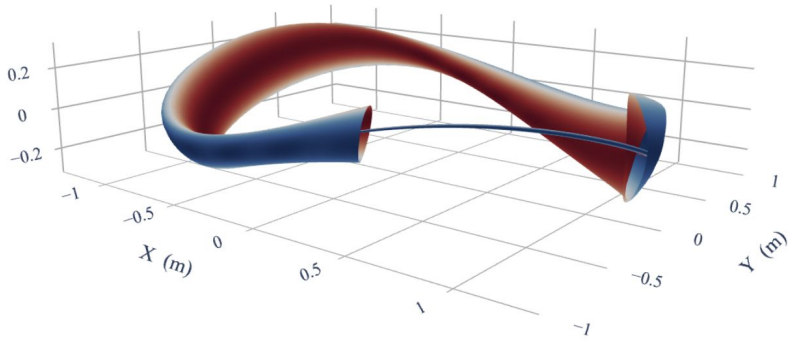
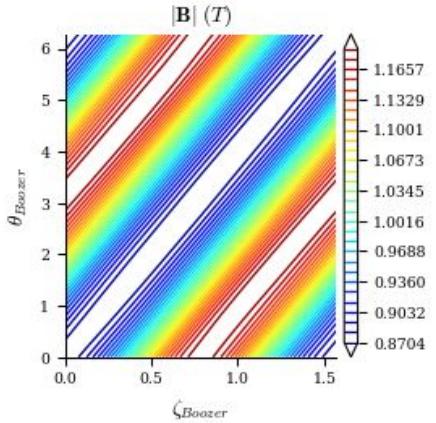
Stellarator Equilibrium and Optimization with DESC

Rory Conlin[2], Daniel Dudt[3], Yigit Gunsur Elmacioglu[1], Rahul Gaur[1], Kian Orr[1], Dario Panici[1], Kaya Unalmis[1], Egemen Kolemen[1]



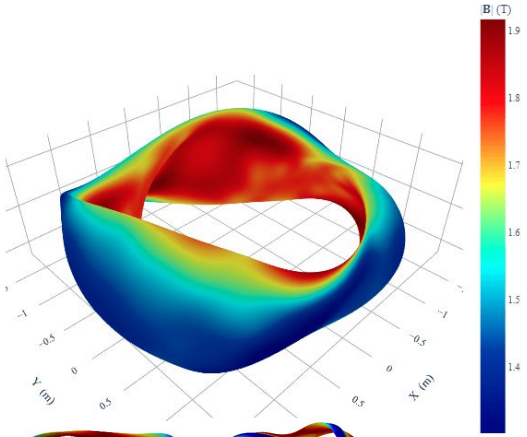
1. Princeton MAE
2. University of Maryland
3. Thea Energy



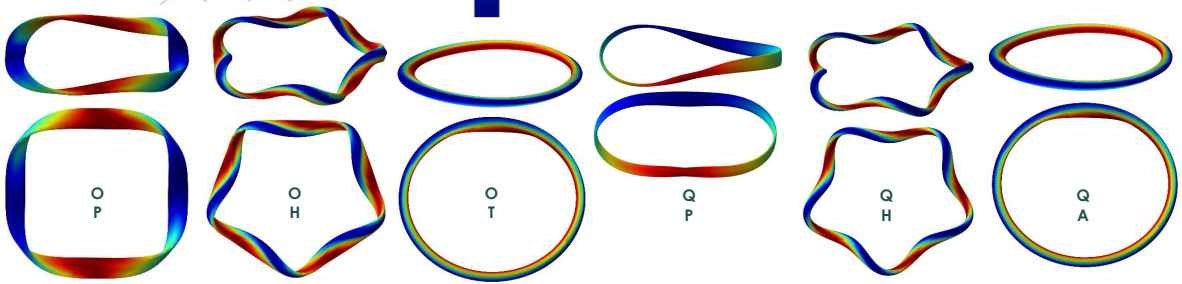
Plasma Control

DESC is a new(ish) tool for stellarator optimization

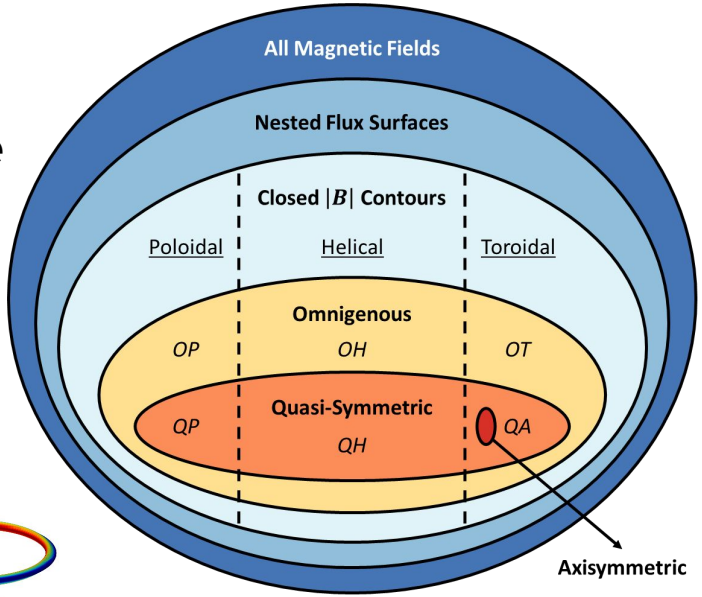
Accurate Equilibria



- Stellarator equilibria are complicated
- Design space is much larger than tokamaks

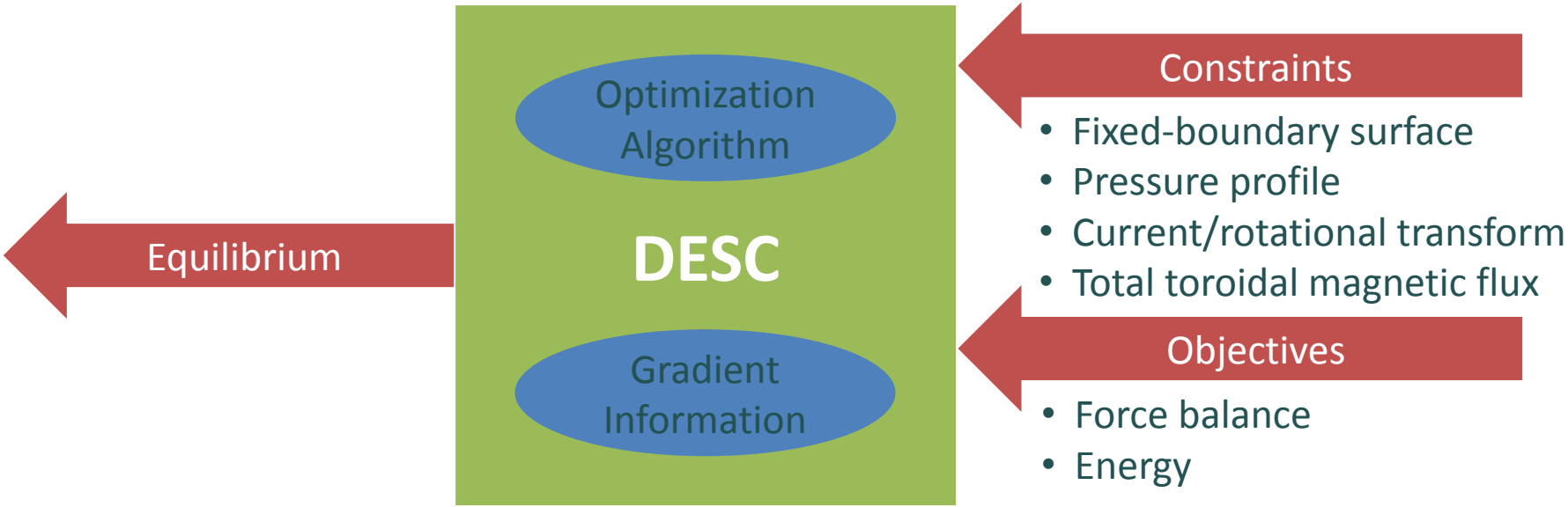


Fast Optimization

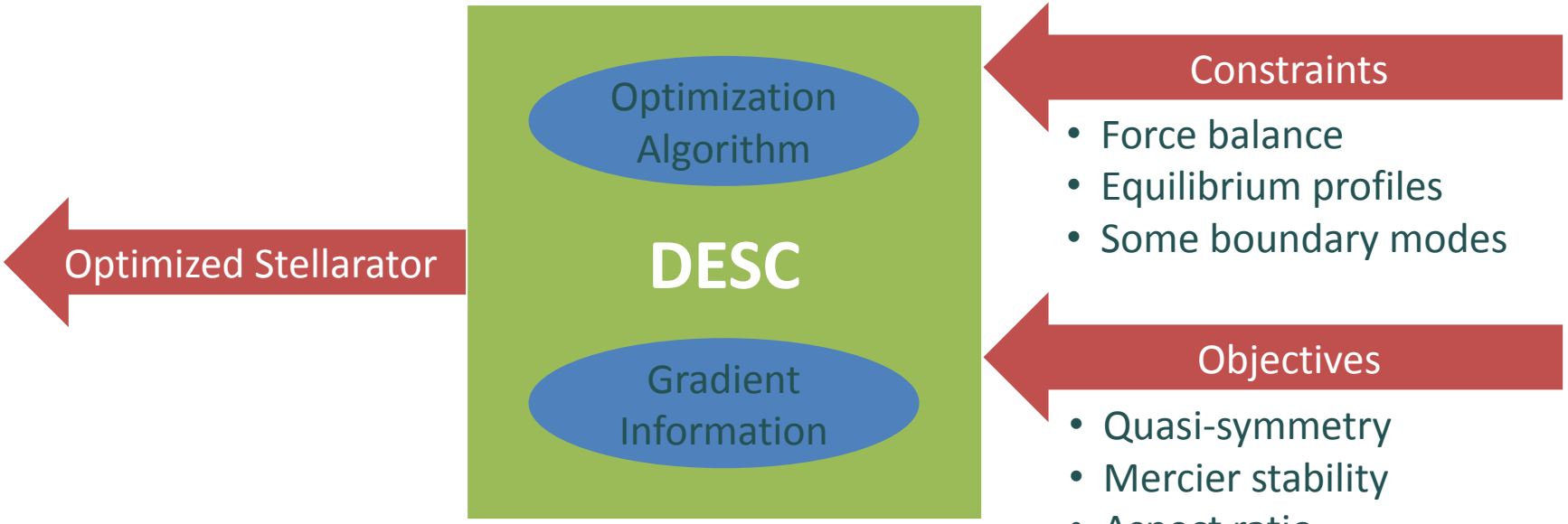


Plasma Control

A flexible stellarator optimization suite

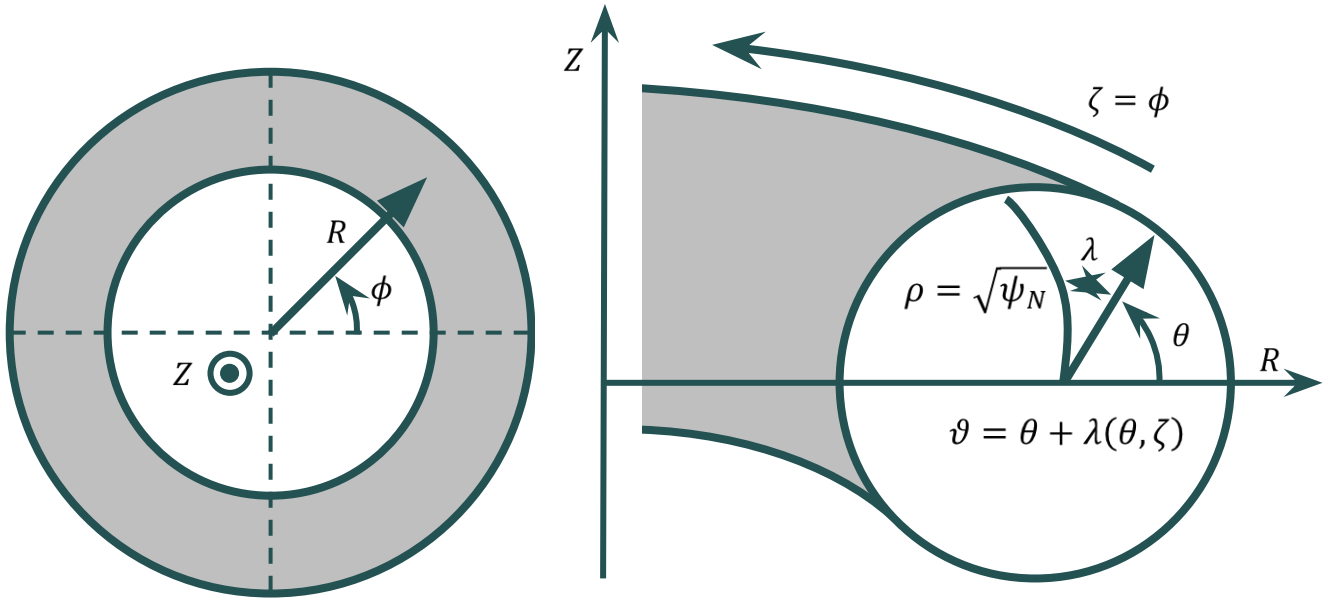


A flexible stellarator optimization suite



Solve the “inverse” equilibrium problem for the shapes of the flux surfaces

Instead of solving for **B** directly in lab coordinates, pick a coordinate system (flux coordinates) where **B** is trivial, then solve for mapping between flux coordinates and lab coordinates



Magnetic field form

$$\boldsymbol{B} = B^{\zeta}\left(\iota \boldsymbol{e}_{\vartheta} + \boldsymbol{e}_{\zeta}\right)$$

assumes

$$\boldsymbol{\nabla} \cdot \boldsymbol{B} = 0, \boldsymbol{B} \cdot \boldsymbol{\nabla}\rho = 0$$

Coordinate map:

$$R(\rho, \theta, \zeta)$$
$$Z(\rho, \theta, \zeta)$$
$$\lambda(\rho, \theta, \zeta)$$

Plasma Control

Solution is represented with global Fourier-Zernike[1,2] spectral basis functions

spectral coefficients

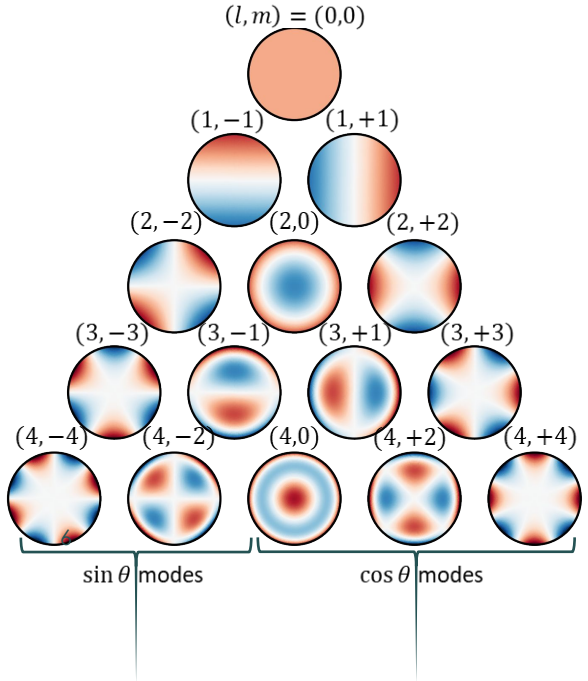
$$R(\rho,\theta,\zeta) = \sum_{lmn} R_{lmn} \mathcal{Z}_l^m(\rho,\theta)\mathcal{F}^n(\zeta)$$

Zernike polynomials

Fourier series

$$\mathcal{Z}_l^m(\rho,\theta) = \begin{cases} \mathcal{R}_l^{|m|}\cos(|m|\theta) \text{ for } m \geq 0 \\ \mathcal{R}_l^{|m|}\sin(|m|\theta) \text{ for } m < 0 \end{cases}$$

one-sided Jacobi polynomials



- Similar discretization for $Z(\rho,\theta,\zeta)$ and $\lambda(\rho,\theta,\zeta)$

[1]Zernike, *Mon. Not. R. Astron. Soc.* (1934).
[2]Loomis, *ASTM STP* (1978).

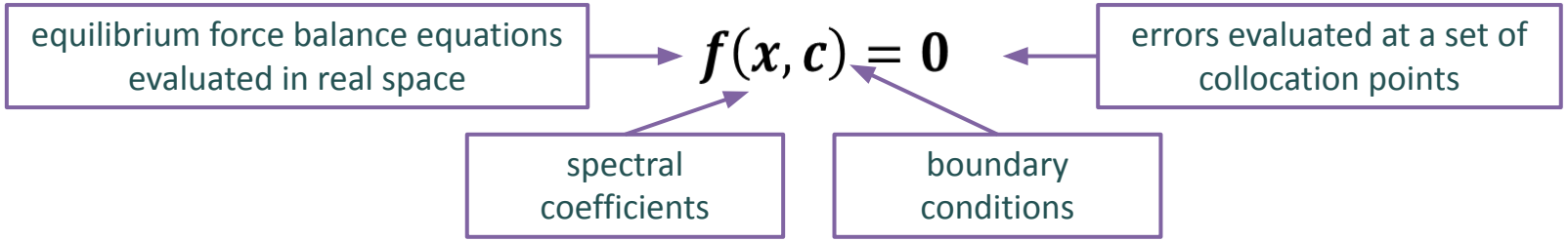
Plasma Control

MHD equilibrium is solved using a pseudo-spectral collocation method

Substituting the spectral expansions into the original PDE

$$\boldsymbol{F} \equiv (\boldsymbol{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B} - \mu_0 \boldsymbol{\nabla} p = \boldsymbol{0}$$

reduces it to a system of nonlinear algebraic equations



which is then solved by a Gauss-Newton method (super-linear convergence)

$$\boldsymbol{x}^* = \mathrm{argmin}_{\boldsymbol{x}}\left(\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c})\|^2\right) \qquad \dim(\boldsymbol{f}) \geq \dim(\boldsymbol{x})$$

Optimize subject to equilibrium constraint

- Equilibrium condition: $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c}) \approx \boldsymbol{0}$
- Optimization objective: $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{c}) \approx \boldsymbol{0}$

} Residuals evaluated at collocation points

- $\boldsymbol{x}$ = spectral coefficients of R, Z, λ (equilibrium solution)
- $\boldsymbol{c}$ = spectral coefficients of R^b, Z^b, p, ι, Ψ (optimization variables)

Optimization problem formulation

- Find the optimal stellarator parameters: $\boldsymbol{c}^* = \mathrm{argmin}_{\boldsymbol{c}}\left(\|\boldsymbol{g}(\boldsymbol{x}^*, \boldsymbol{c})\|^2\right)$
- Subject to the equilibrium constraint: $\boldsymbol{x}^* = \mathrm{argmin}_{\boldsymbol{x}}\left(\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c})\|^2\right)$

Gradient computations are the bottleneck of traditional stellarator optimization

- $g(\boldsymbol{c}) =$ cost function to be minimized; $\boldsymbol{c} =$ optimization variables
- Gradient descent optimization:

$$\boldsymbol{c}_{n+1} = \boldsymbol{c}_n - \gamma \nabla g(\boldsymbol{c}_n)$$

Finite Differences:

- Requires $\geq \dim(\boldsymbol{c})$ equilibrium solves
- Inaccurate and sensitive to step size

Adjoint methods:

- Not applicable to all objectives
- Laborious to implement

Efficient computing with the ease of Python

Automatic Differentiation (AD)

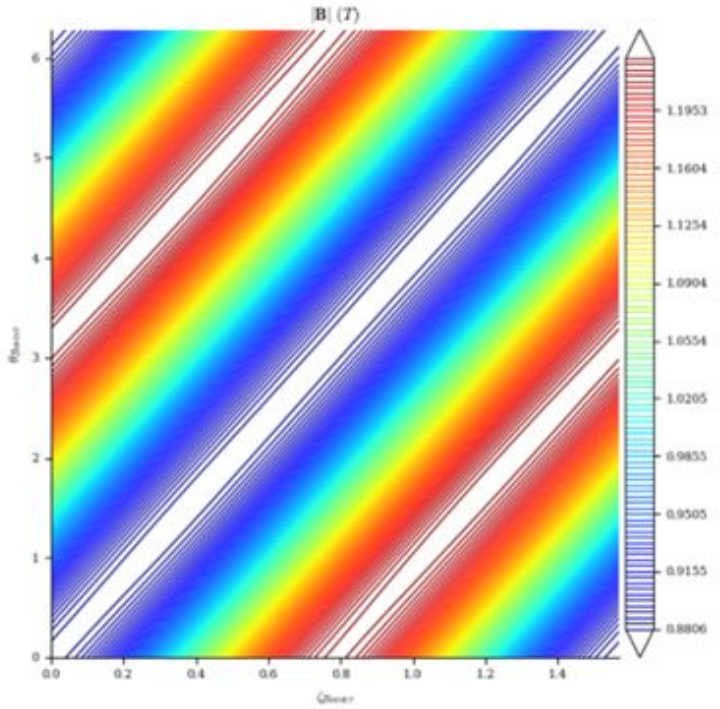
- Optimization requires derivative information
- Exact derivatives of arbitrary functions to any order

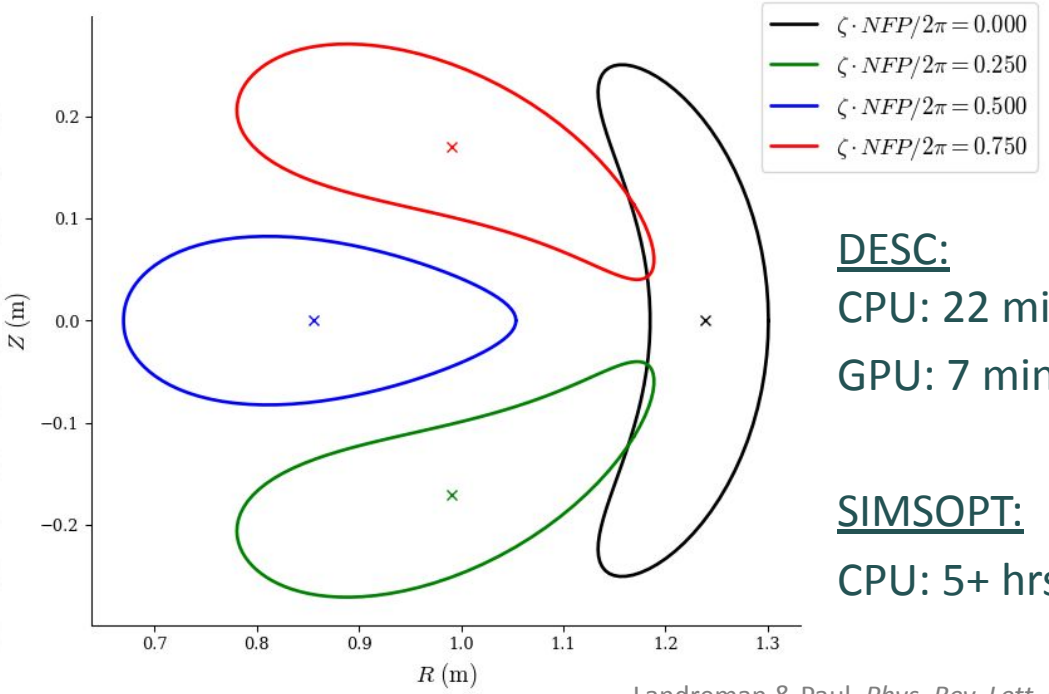
Just-In-Time (JIT) Compilation

- Comparable speed to C or Fortran compiled languages
- Hardware agnostic (CPU, GPU, TPU)

Requires specific code structure, but easy to implement: `import jax.numpy as jnp`

Can find “precise quasi-symmetry” & more





DESC:

CPU: 22 min

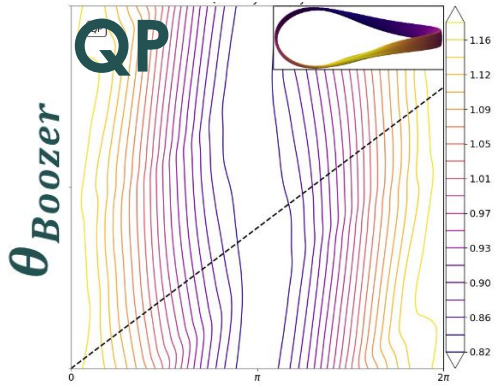
GPU: 7 min

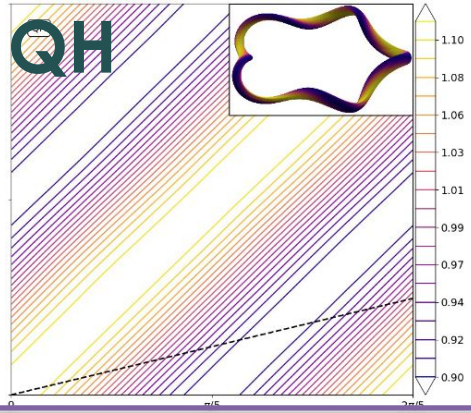
SIMSOPT:

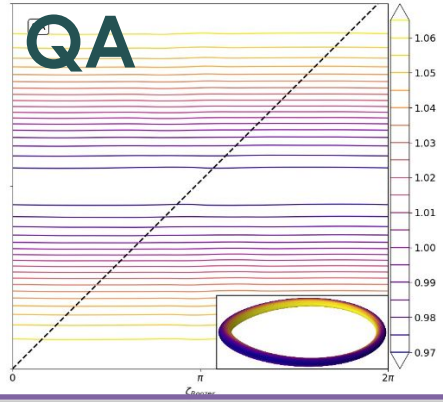
CPU: 5+ hrs

Landreman & Paul, *Phys. Rev. Lett.* (2022)

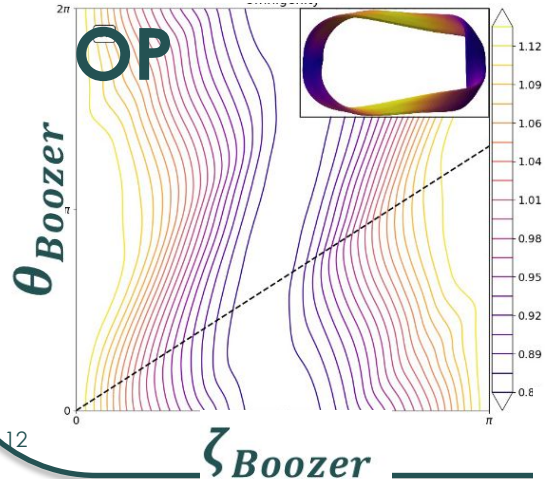
DESC can find equilibria with any omnigenity type

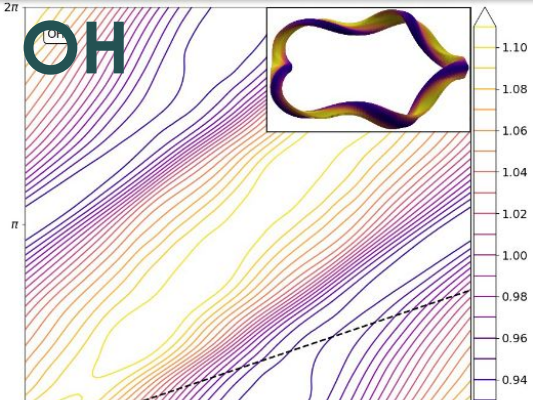


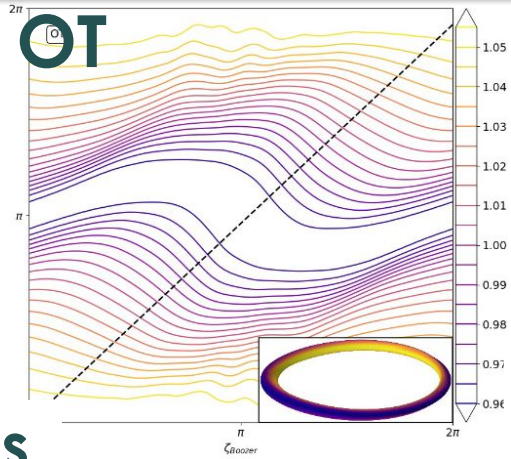




QS Fields

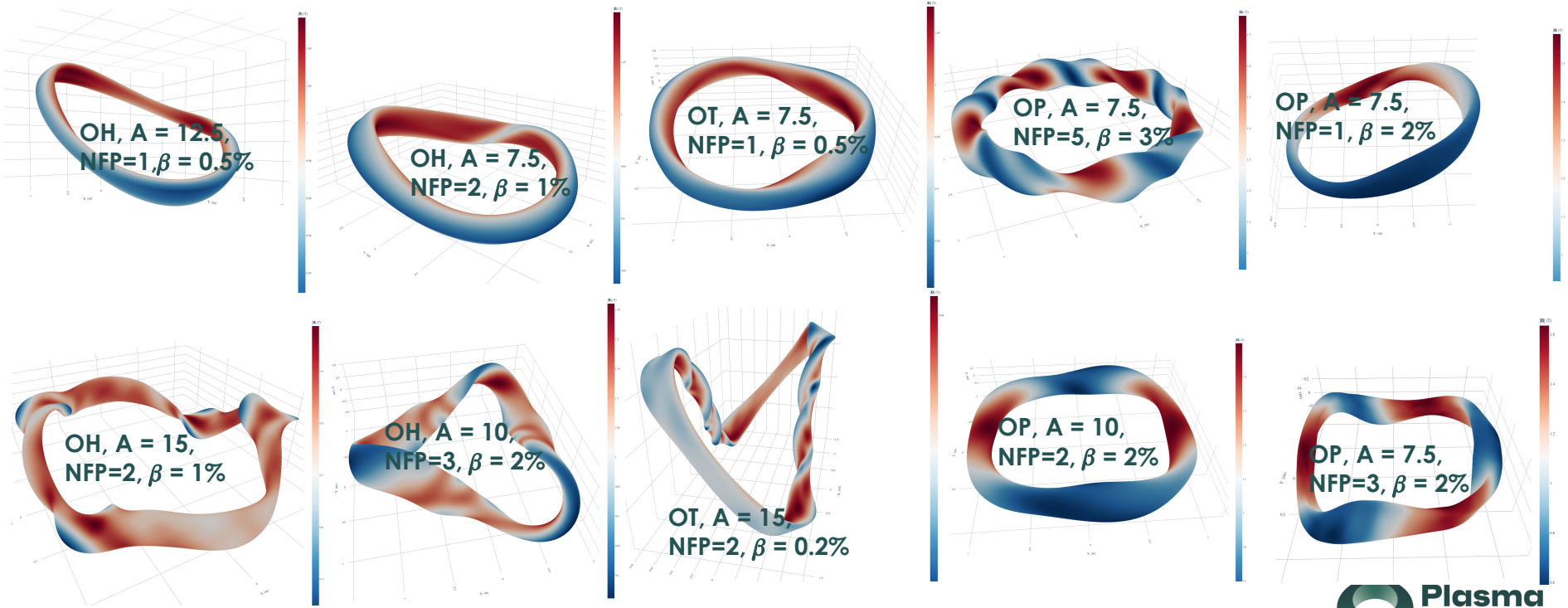






Omnigenous Fields

Lots of possible designs



Switch to notebooks/code examples:

1. https://desc-docs.readthedocs.io/en/stable/notebooks/tutorials/basic_optimization.html
2. https://desc-docs.readthedocs.io/en/stable/notebooks/tutorials/advanced_optimization.html
3. https://desc-docs.readthedocs.io/en/stable/notebooks/tutorials/coil_stage_two_optimization.html
4. https://desc-docs.readthedocs.io/en/stable/notebooks/tutorials/omnigenity.html

Some problems to try (some “easy”, some not)

1. Which of the different types of Quasisymmetry (QA, QH, QP) are "easier" to optimize for?
2. Which of the metrics for Quasisymmetry (`QuasisymmetryBoozer`, `QuasisymmetryTripleProduct`, `QuasisymmetryTwoTerm`), work best? Are some faster? Do some give better results?
3. Does performing a “multigrid” optimization (starting with low resolution and then increasing) improve the results for coil optimization?
4. How small can you make the symmetry breaking Boozer harmonics? Can you get it less than the earths geomagnetic field?
5. What sorts of "trivial" optima exist for QS optimization? How can we avoid them?
6. How can we avoid strong shaping that can be difficult to product with external coils?
7. How much do initial conditions matter? Are there certain initial guesses that lead to much better solutions? How does this depend on what we're optimizing for?
8. Can you make a coilset for a stellarator using only planar coils? What about only circular coils?
9. How does varying the number of field periods change the achievable level of QS?
10. How many coils is “enough”? Is there a point where adding more coils doesn’t help?
11. How does adding pressure change the equilibrium? Does it affect quasisymmetry?
12. Do certain flavors of QS "prefer" different values of rotational transform?
13. What if we want to include some objective for MHD stability? (`MagneticWell` , `MercierStability`) Does this make the optimization harder? easier? Is there a tradeoff between QS and stability?
14. If you only optimize for QS on a single surface, which surface would you pick? Is there a difference?
15. What if we fix the geometry of the coils and just allow the currents to change, does this work? (Try using different ways to avoid the trivial solution for the vacuum case)
16. Can you modify the precise QA coil example in the coil tutorial to setup a single stage optimization problem, where you optimize both the coils and the plasma at the same time? Does this improve the QS or normal field error?