Efficient Stellarator Optimization and Analysis with DESC

Daniel Dudt, Rory Conlin, Dario Panici,
Patrick Kim, Kaya Unalmis, Egemen Kolemen

APS DPP October 21, 2022

PPPL

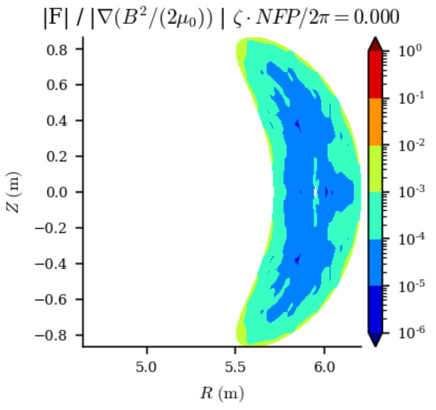
Princeton Plasma Control
control.princeton.edu

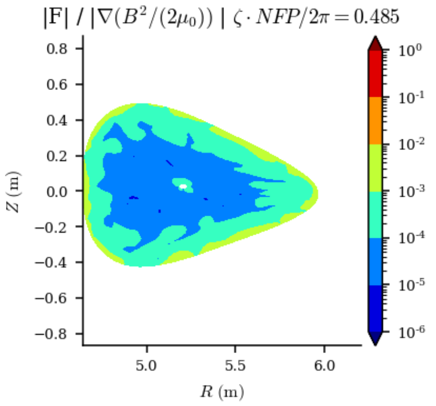
andlinger center
for energy+the environment

PRINCETON
UNIVERSITY

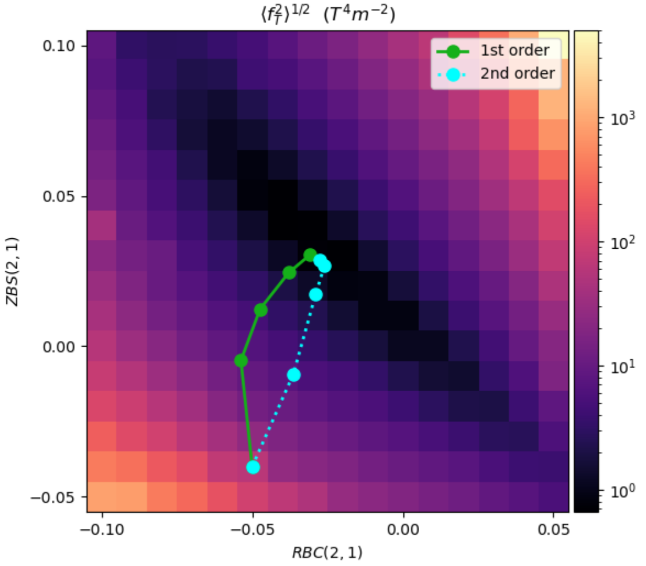
DESC is a new tool for stellarator optimization

Accurate Equilibria

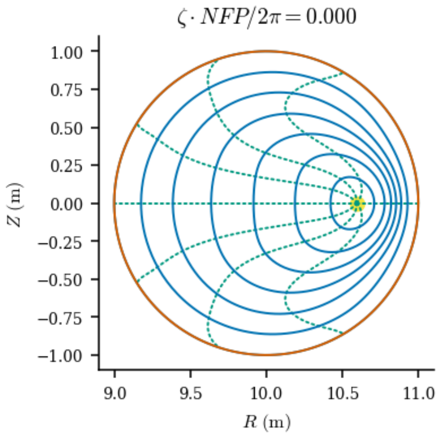


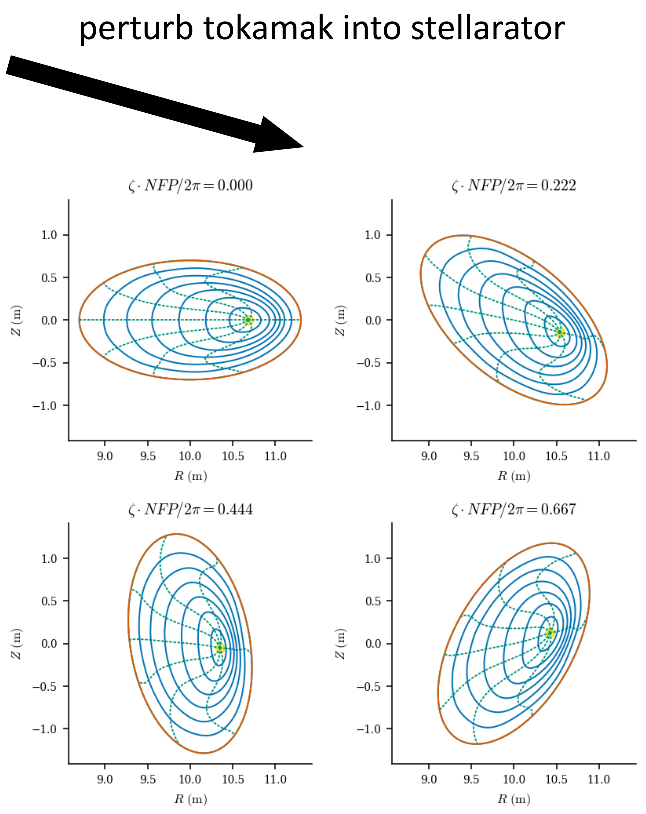


Fast Optimization



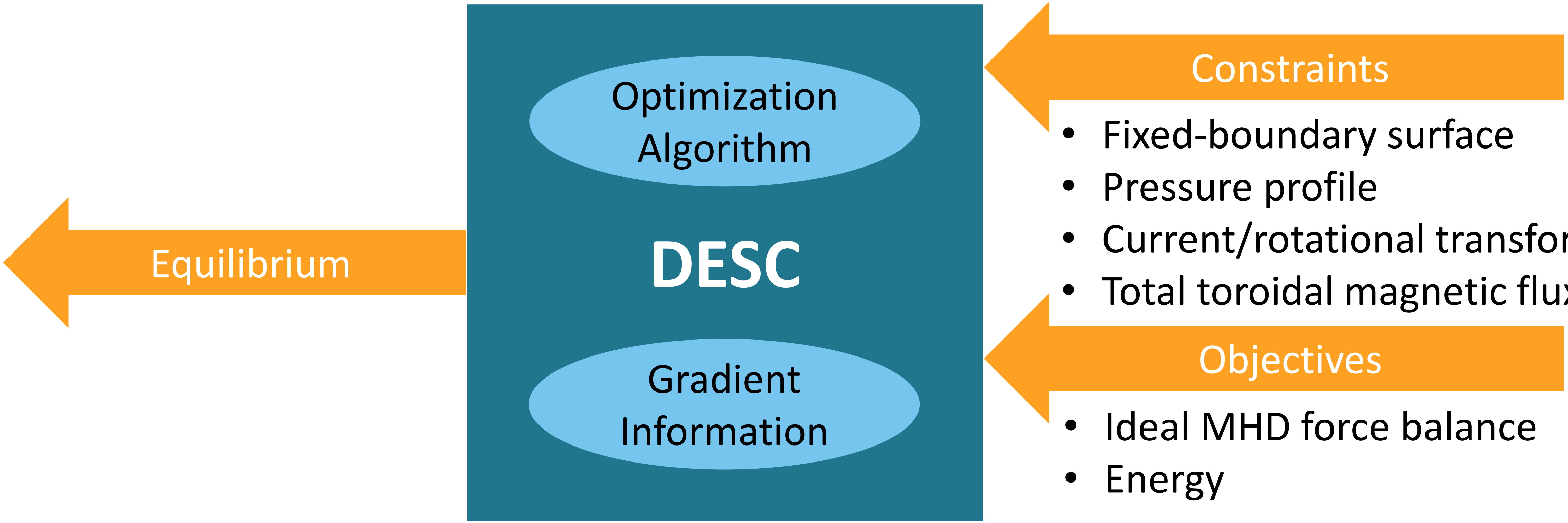
Phase-Space Connections





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

A flexible stellarator optimization suite



Constraints
- Fixed-boundary surface
- Pressure profile
- Current/rotational transform
- Total toroidal magnetic flux

Objectives
- Ideal MHD force balance
- Energy

A flexible stellarator optimization suite

DESC

Optimization Algorithm

Gradient Information

Optimized Stellarator

Constraints

- Ideal MHD force balance
- Equilibrium profiles
- Some boundary modes

Objectives

- Quasi-symmetry
- Mercier stability
- Aspect ratio
- etc.

Why do we need *another* stellarator code?

Equilibrium solvers: VMEC, NEAR, PIES, HINT, SPEC, GVEC, etc.

Optimization codes: STELLOPT, ROSE, WISTELL, SIMSOPT, etc.

1. Better understand the solution space of stellarator equilibria
2. Integrate the equilibrium solver with optimization tools
3. Avoid Jacobian approximations, near-axis expansions, low-β expansions, etc.
4. Use modern numerical methods and scientific computing practices

Developed with the following design principles:

1. **Simple user interface**
   - Open-source Python code
   - Well documented
   - High test coverage
   - Easy to install
2. **Local error quantification**
   - Pseudo-spectral (collocation) methods
3. **Properly resolve the magnetic axis**
   - Global basis functions
   - Zernike polynomials
4. **Exact derivatives of all objectives**
   - Automatic differentiation
5. **Hardware agnostic**
   - Run on CPUs, GPUs, and TPUs
6. **Extendable to new applications**
   - Modular & flexible code structure

Zernike spectral basis inherently satisfies boundary conditions at the magnetic axis

spectral coefficients

Zernike polynomials

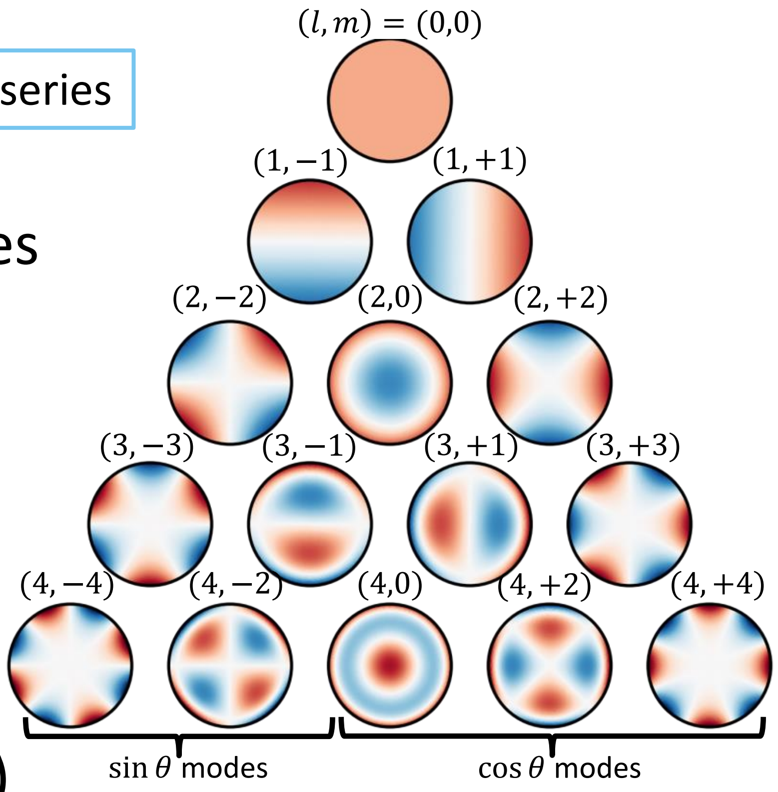
$$X(\rho,\theta,\zeta) = \sum_{lmn} X_{lmn} \mathcal{Z}_l^m(\rho,\theta)\mathcal{F}^n(\zeta)$$

Fourier series

- Periodic boundary conditions for poloidal & toroidal angles
- Satisfies analyticity conditions at the magnetic axis:

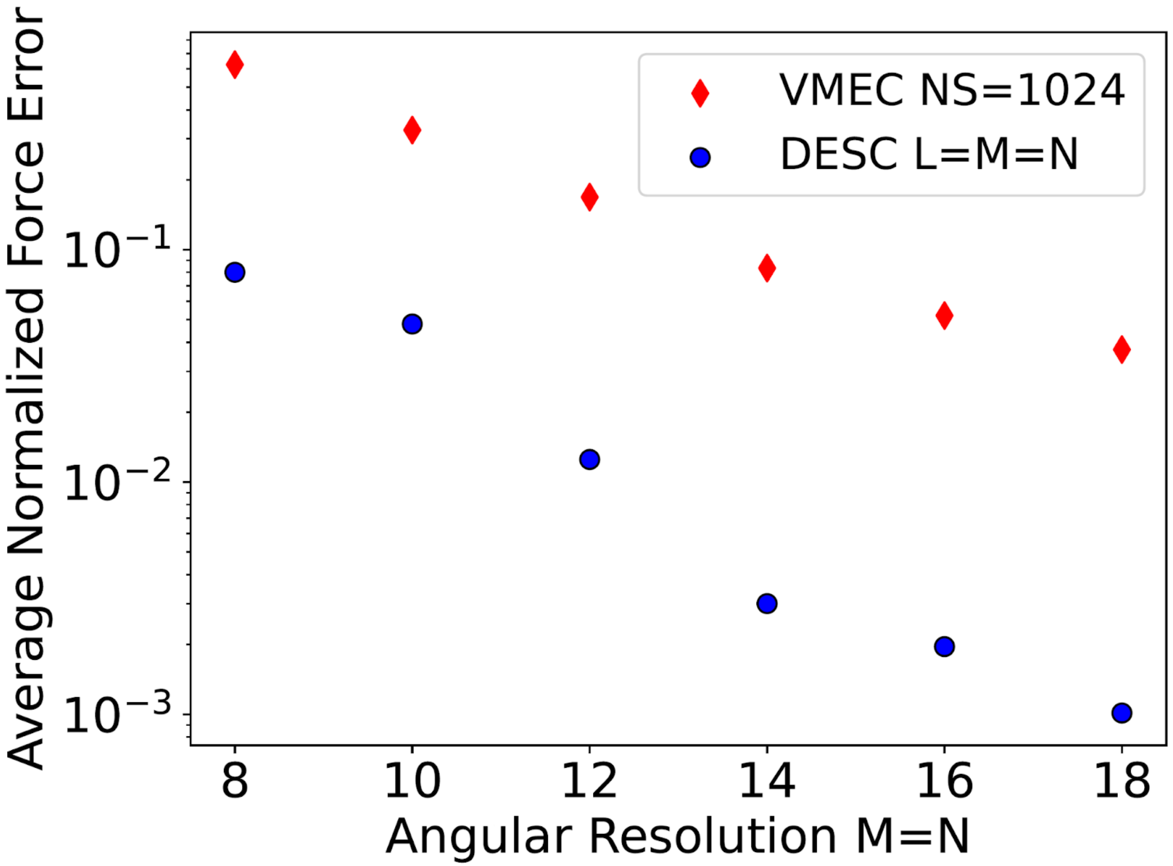
$$f(\rho,\theta) = \sum_m \rho^m\left(a_{m,0} + a_{m,2}\rho^2 + \cdots\right)\cos(m\theta) + \sum_m \rho^m\left(b_{m,0} + b_{m,2}\rho^2 + \cdots\right)\sin(m\theta)$$

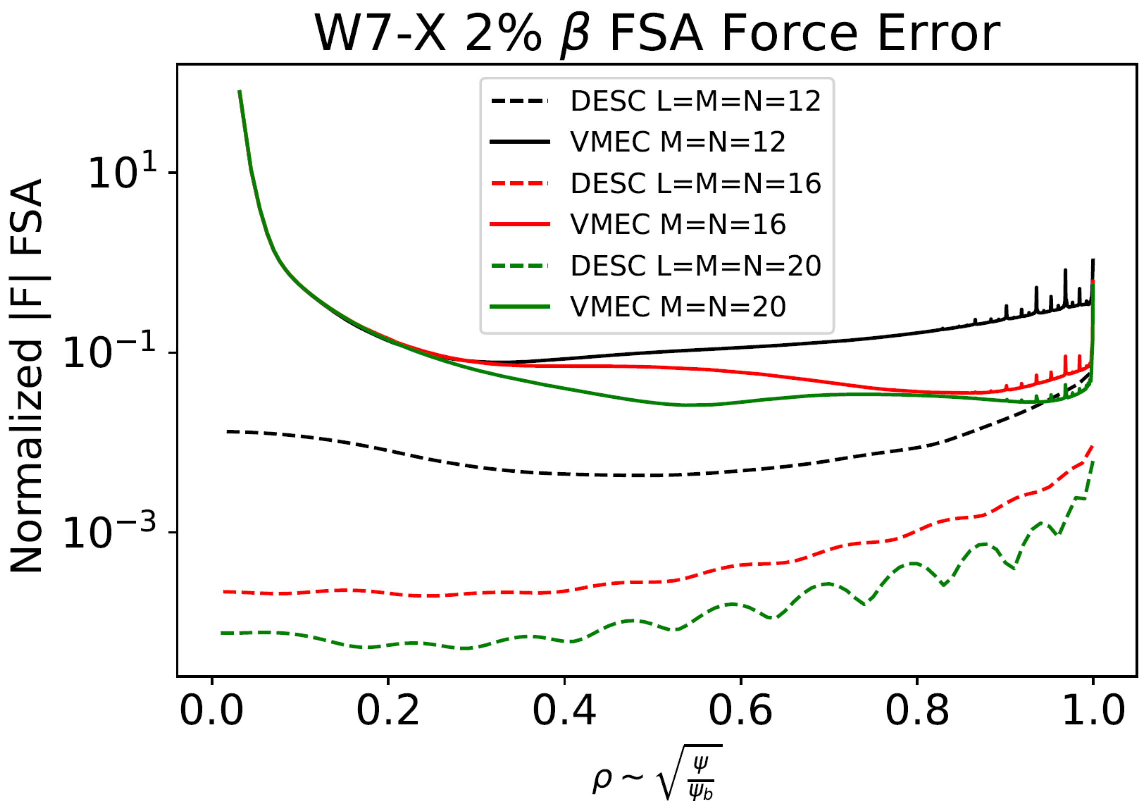
- Exponential convergence (if solution exists and is smooth)



Boyd et al., *J. Comput. Phys.* (2011)
Lewis et al., *J. Math. Phys.* (1990)

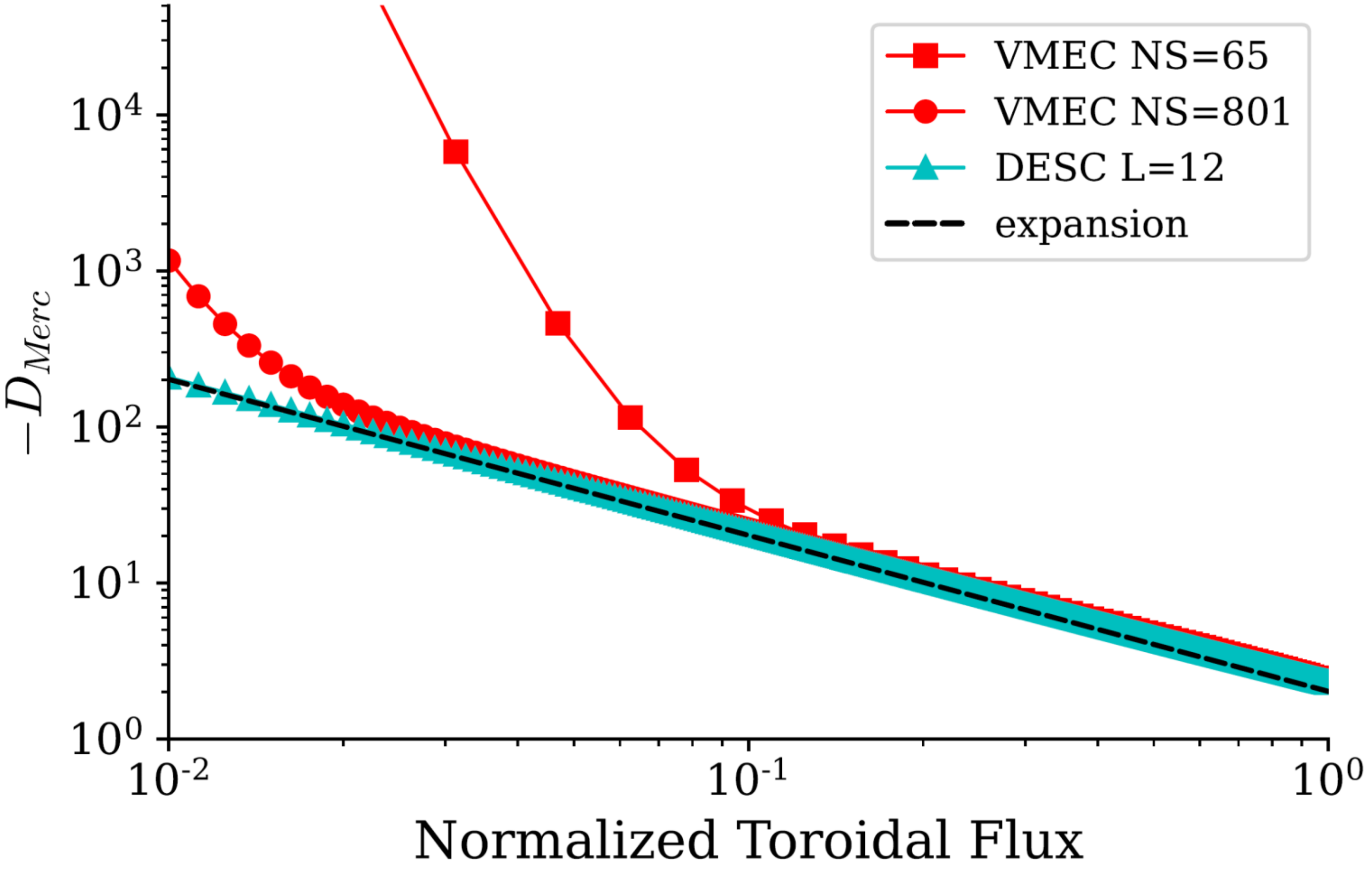
Spectral methods yield more accurate equilibrium solutions





Hirshman & Whitson, *Phys. Fluids* (1983)

Accurately resolving the magnetic axis is important for stability calculations



VMEC requires high radial resolution to resolve axis

<u>Run times:</u>

- DESC = 0.2 GPU-hours (NVIDIA A100)
- VMEC = 5.2 CPU-hours (AMD Opteron 6276)

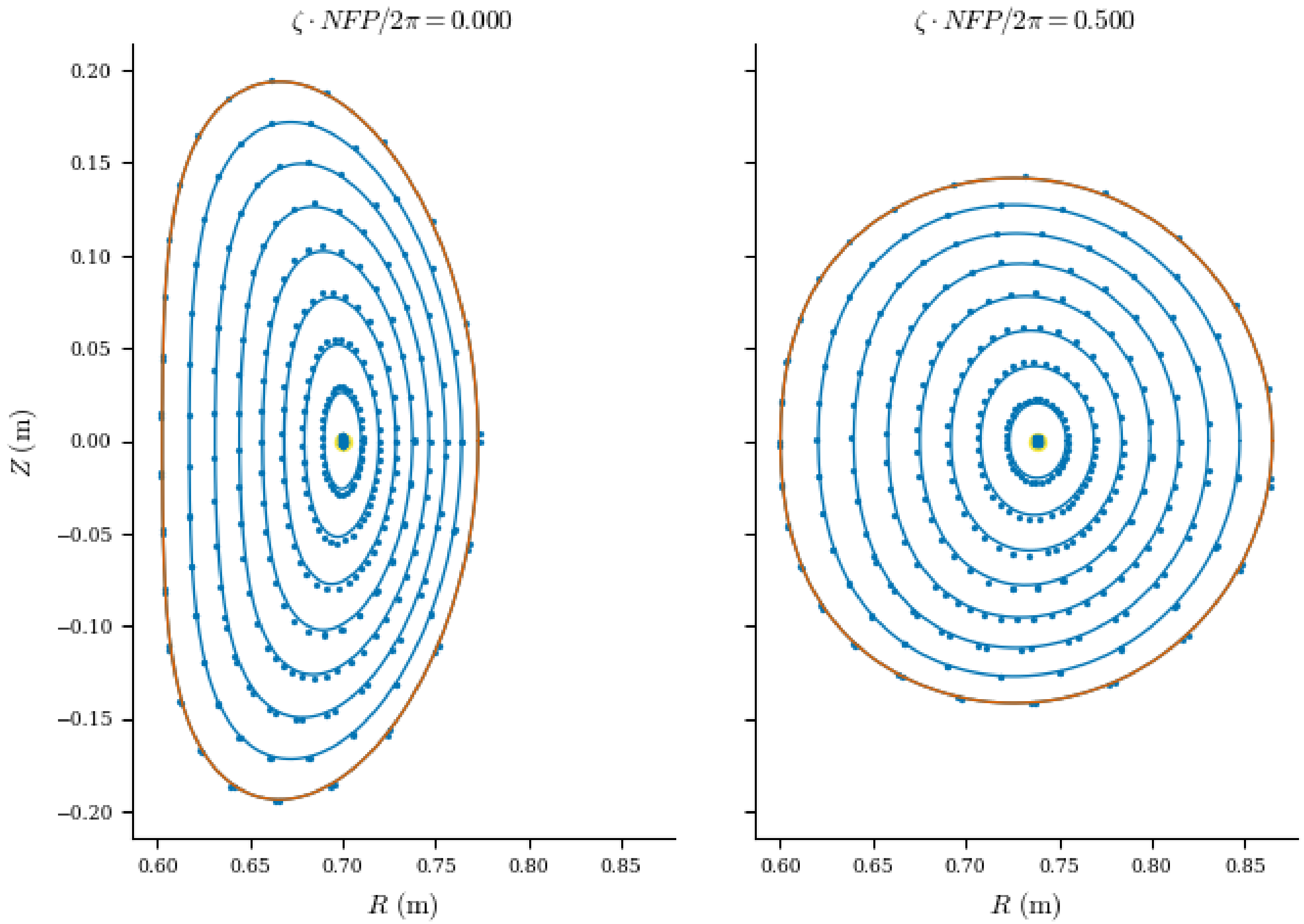
Landreman & Sengupta, *J. Plasma Phys.* (2019)

Extended to free-boundary equilibria

<u>Algorithms:</u>

- NESTOR implemented
- BIEST in progress
- Testing other methods

- Vacuum solution agrees with field-line tracing
- Benchmarked against VMEC



Merkel, *J. Comp. Phys.* (1986)
Malhotra et al., *J. Comp. Phys.* (2019)

Novel boundary conditions to better parameterize stellarator design space

<u>Conventional boundary condition:</u>

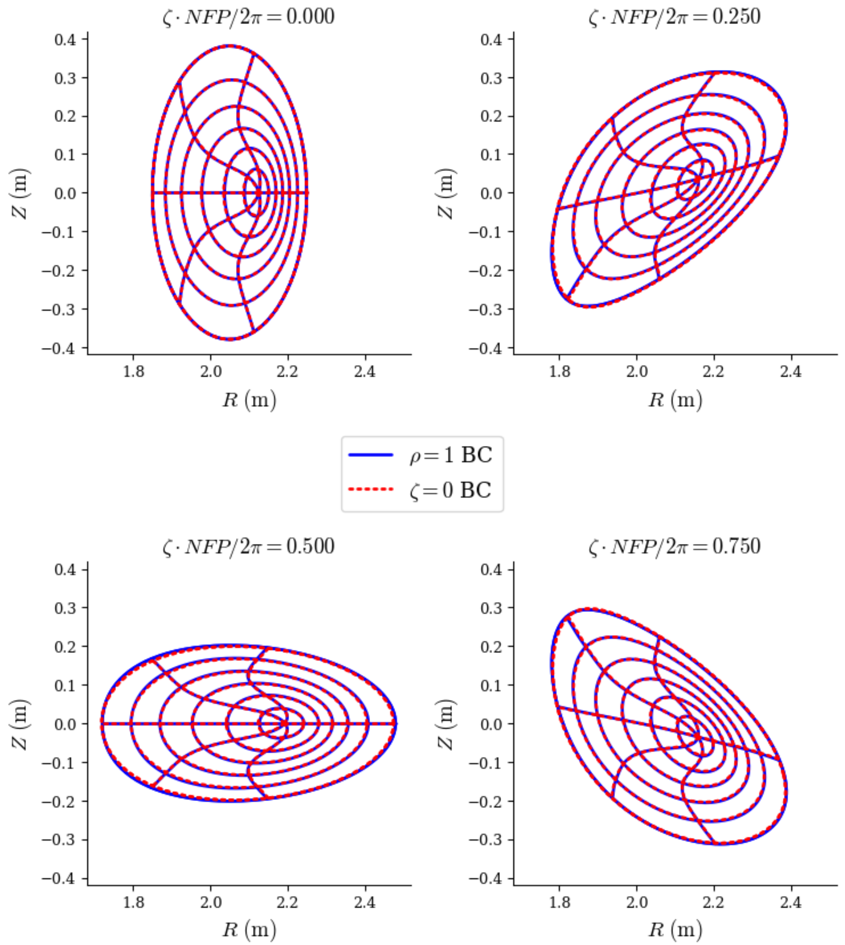
Specify shape of the last closed flux surface at $\rho = 1$

- Do equilibria with nested flux surfaces exist?

<u>Poincarè boundary condition:</u>

Specify surfaces in the cross-section at $\phi = 0$

- Restrict solution space to nested flux surfaces
- Connects tokamaks and stellarators in phase space



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`**

DESC optimization only requires a single equilibrium solve per iteration

1. Newton optimization step with equilibrium constraint

$$\boldsymbol{c}_{n+1} = \boldsymbol{c}_n + \Delta\boldsymbol{c}$$

$$\left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_n}\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_n}\right)^{-1}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{c}_n} - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{c}}\right]\Delta\boldsymbol{c} = \boldsymbol{g}(\boldsymbol{x}_n, \boldsymbol{c}_n)$$

Exact Jacobians known from automatic differentiation!

$\boldsymbol{f}$ = equilibrium constraint
$\boldsymbol{g}$ = optimization objective

$\boldsymbol{x}$ = equilibrium solution
$\boldsymbol{c}$ = optimization variables

2. Perturb equilibrium solution to reflect new parameters

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\boldsymbol{x}$$

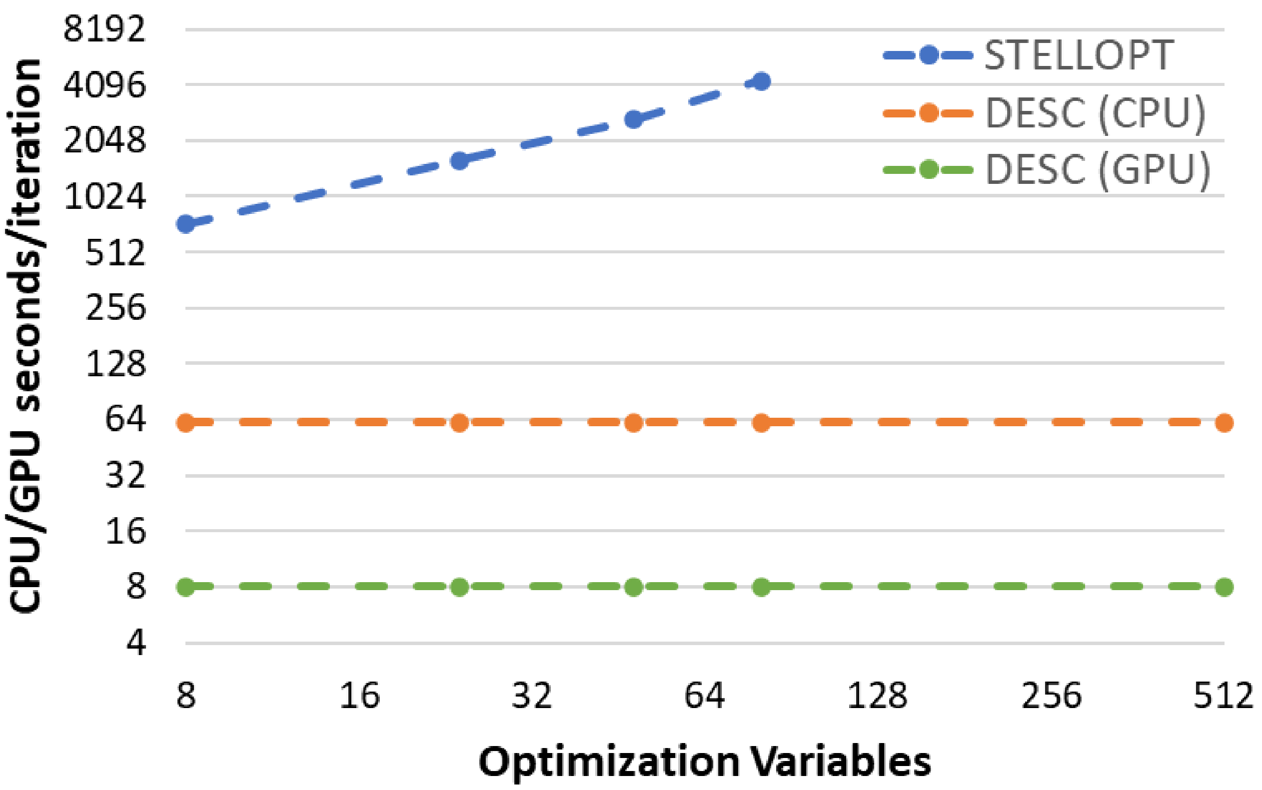
$$\Delta\boldsymbol{x} = -\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_n}\right)^{-1}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{c}_n}\Delta\mathbf{c}$$

3. Re-solve equilibrium from this close initial guess

$$\boldsymbol{x}_{n+1} = \text{argmin}_{\boldsymbol{x}}\left(\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c}_{n+1})\|^2\right)$$

Only 1 “warm-start” equilibrium solve per optimization step!

Fast computations enable exploration of the large stellarator design space



- Finite differences scale unfavorably
- Parallelization can help reduce wall time, but not total resources
- GPU hardware is still improving

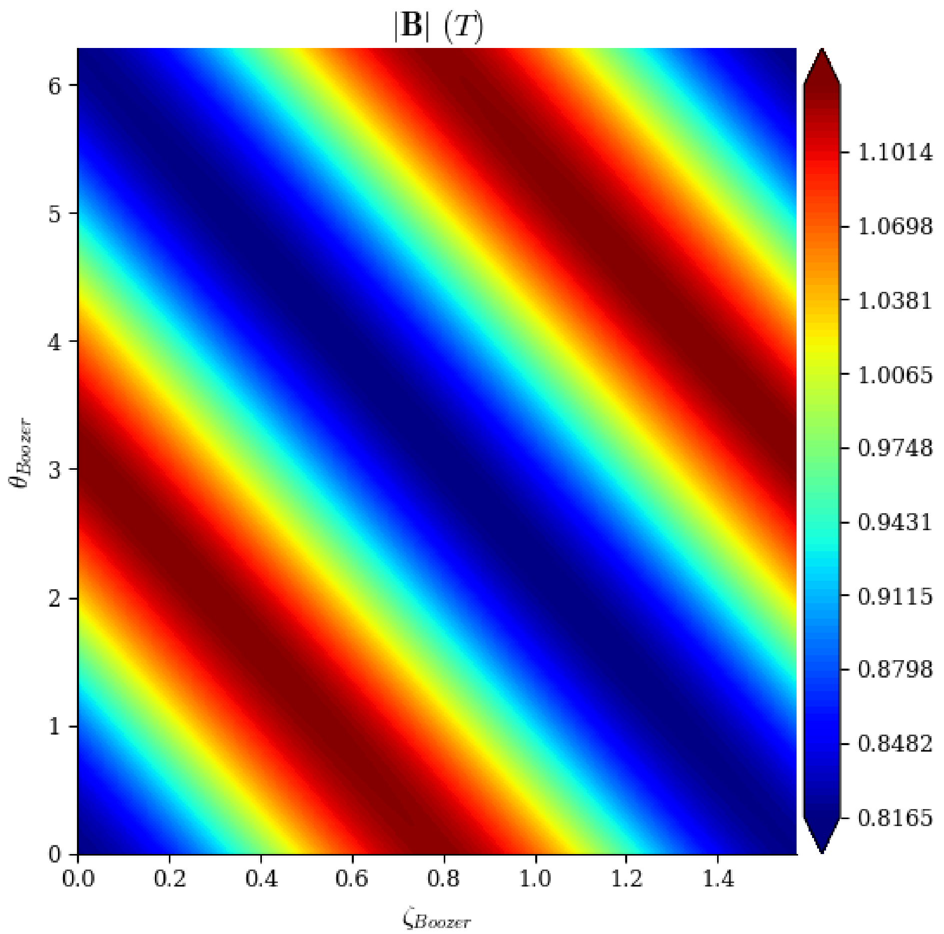
W7-X $\beta = 2\%; L = 24, M = N = 12$

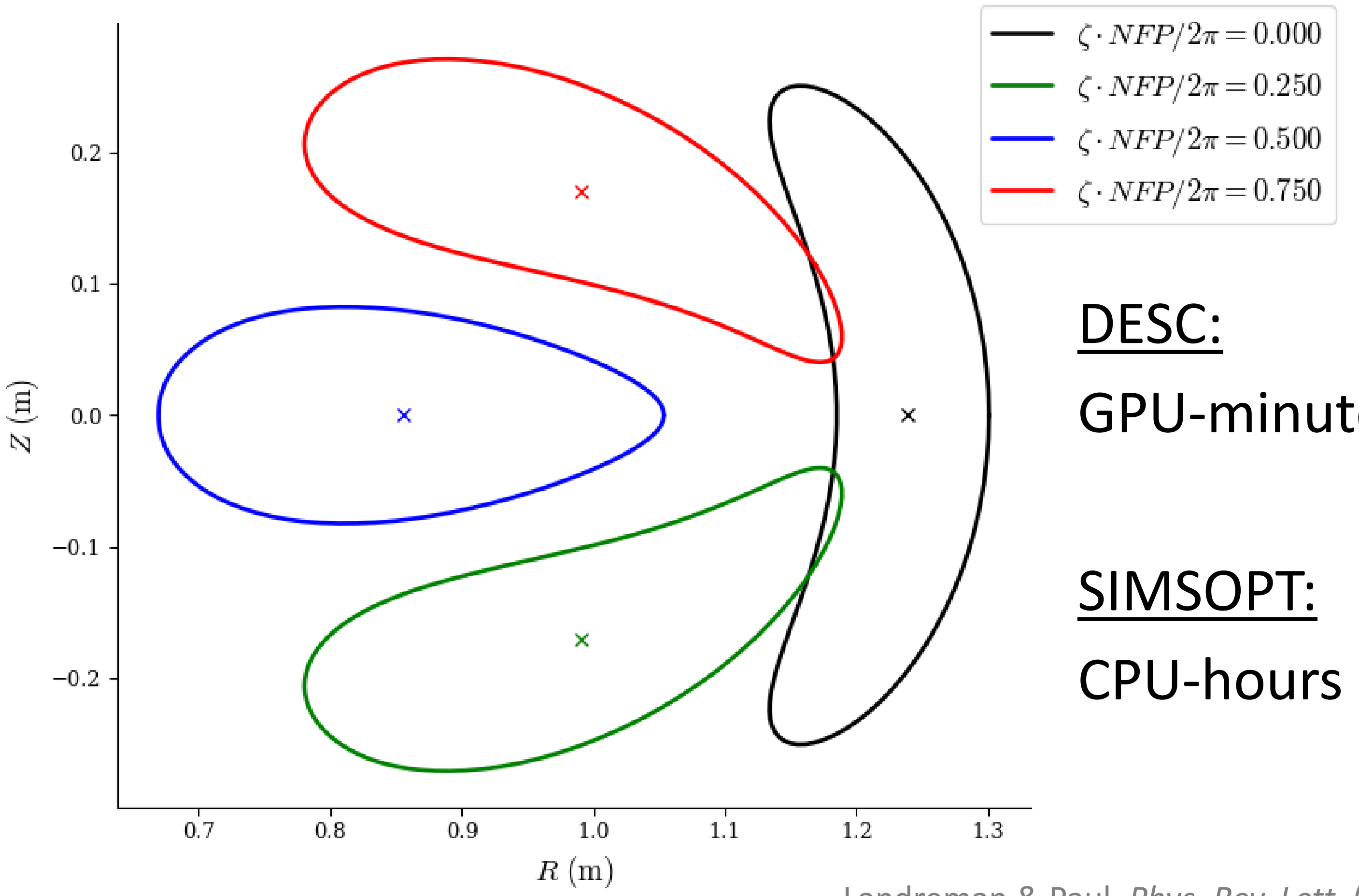
Hardware	Run Time
Intel Cascade Lake CPU	48 min
NVIDIA A100 GPU	20 min

Run optimizations in a few lines of Python code

```python
set_device("gpu")  # run on a GPU
eq = desc.io.load("path/to/initial/equilibrium.h5")
grid = LinearGrid(M=eq.M, N=eq.N, NFP=eq.NFP, rho=np.linspace(0.1, 1, 10))  # computation grid
objective = ObjectiveFunction((AspectRatio(target=8),  # target aspect ratio
    QuasisymmetryTwoTerm(helicity=(1, -eq.NFP), grid=grid, weight=2e-1)))  # optimize for QH
# optimize boundary modes with |m|,|n|<=5 (constrain boundary modes with |m|,|n|>5)
R_modes = np.vstack(([0, 0, 0],  # fix major radius
    eq.surface.R_basis.modes[np.max(np.abs(eq.surface.R_basis.modes), 1) > 5, :]))
Z_modes = eq.surface.Z_basis.modes[np.max(np.abs(eq.surface.Z_basis.modes), 1) > 5, :]
constraints = (ForceBalance(), FixBoundaryR(modes=R_modes), FixBoundaryZ(modes=Z_modes),
    FixPressure(), FixCurrent(), FixPsi())  # fix vacuum profiles
optimizer = Optimizer("lsq-exact")  # least-squares optimization algorithm
eq.optimize(objective, constraints, optimizer)  # run optimization
eq.save("path/to/optimal/solution.h5")
```

Can find “precise quasi-symmetry” & more





<u>DESC:</u>

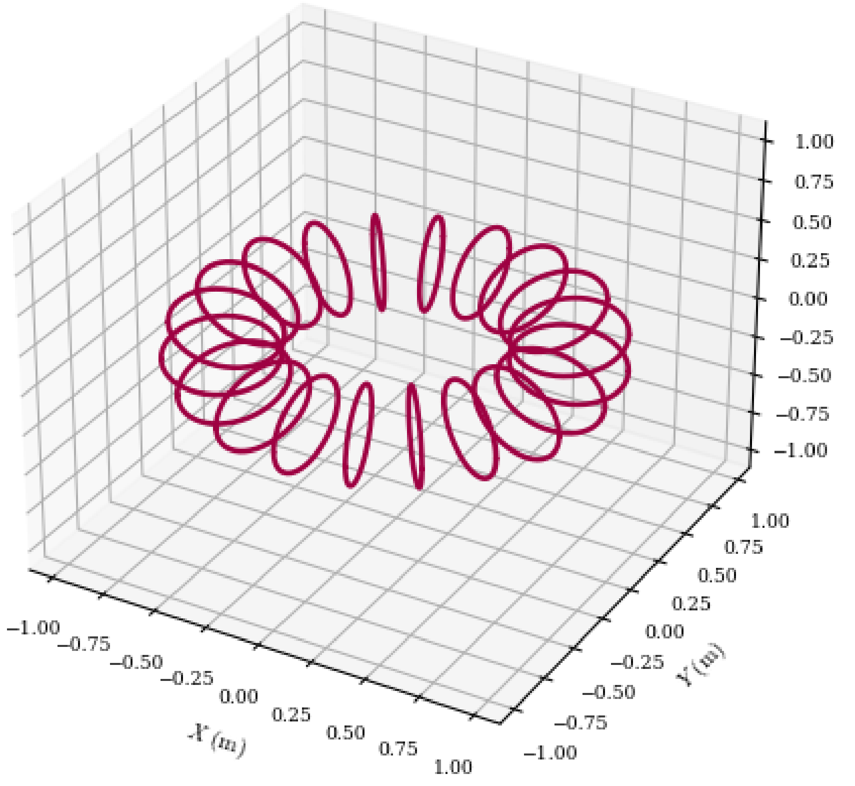
GPU-minutes

<u>SIMSOPT:</u>

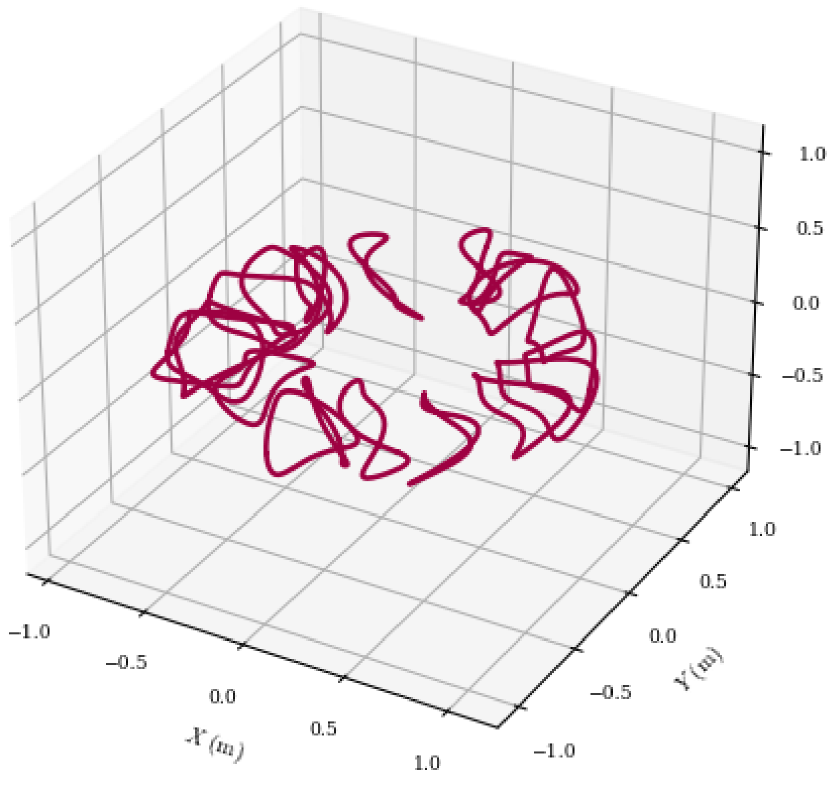
CPU-hours

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

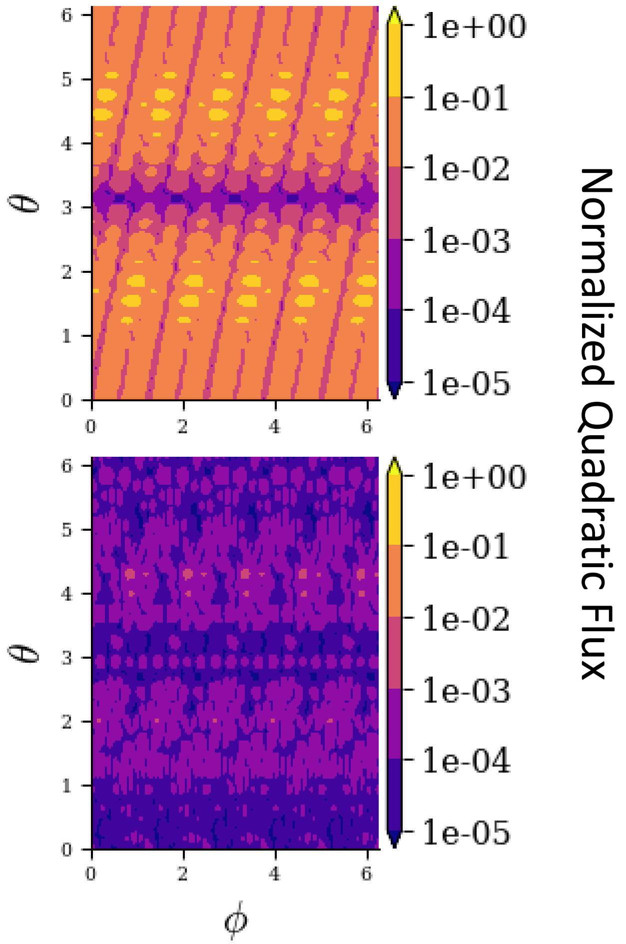
Can perform coil design & optimization



Initial

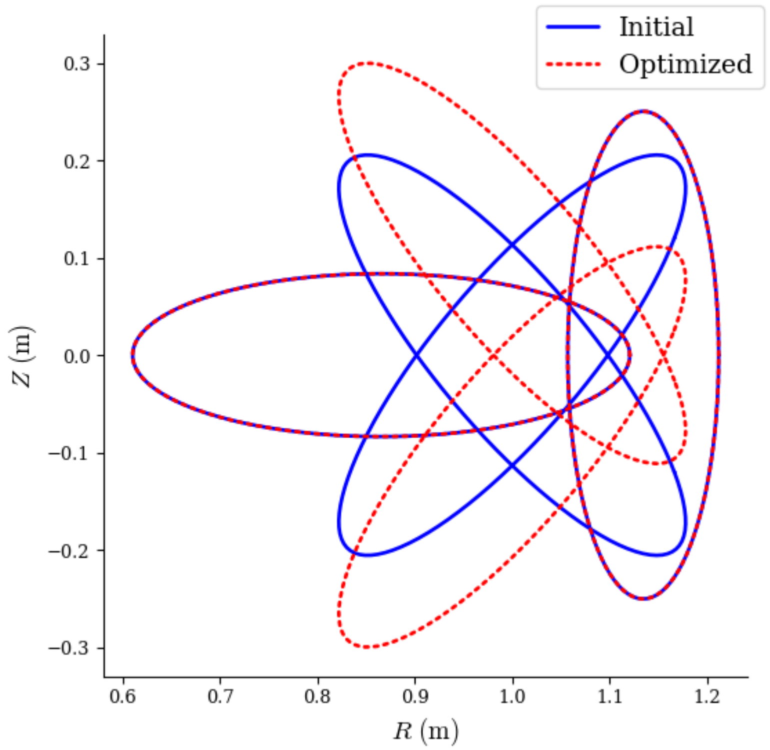


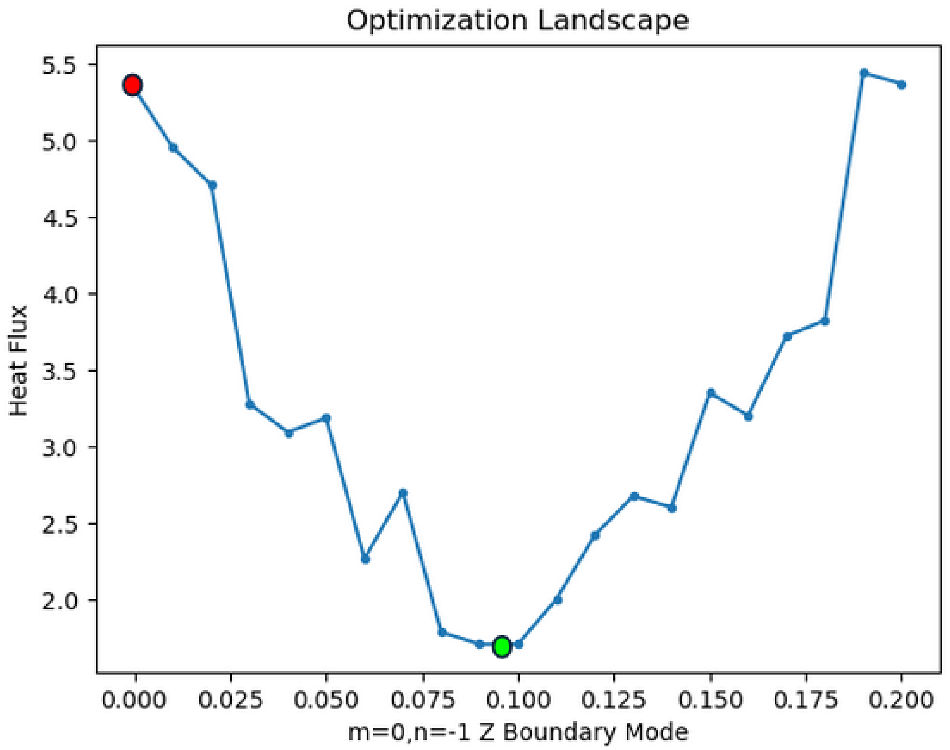
Final



Can wrap other codes with finite differences

- GX is a fast (minutes) pseudo-spectral gyrokinetic code for stellarators



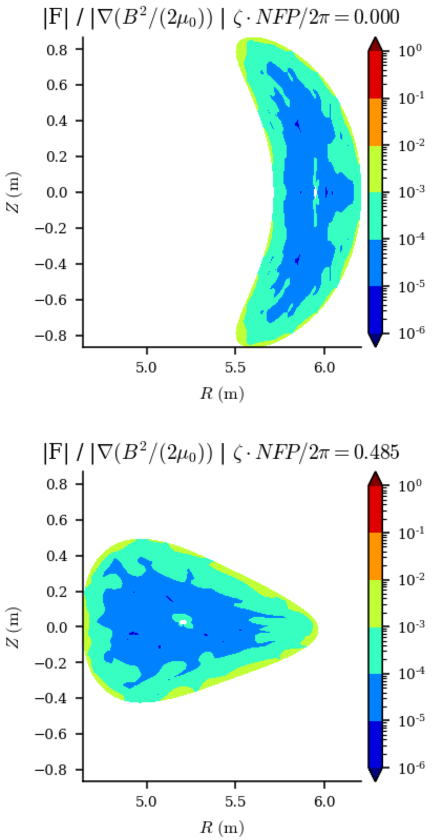


- Also wrapped NEO to optimize for effective ripple ε_{eff}

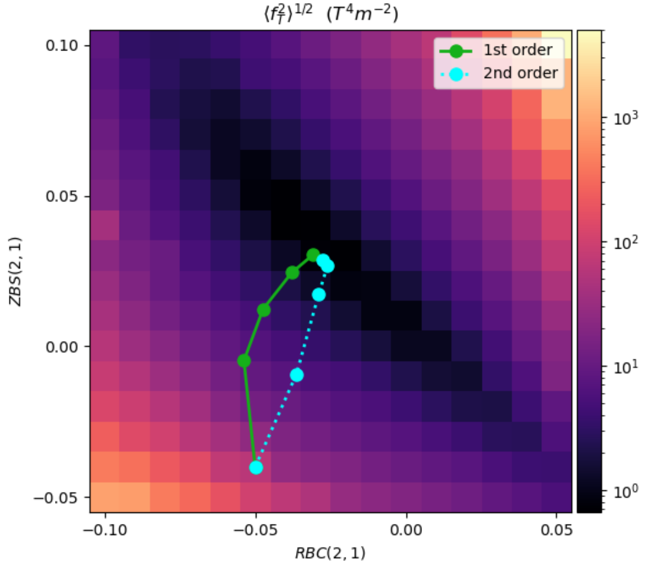
Mandell et al., *J. Plasma Phys.* (2018)
Gonzalez et al., *J. Plasma Phys.* (2022)
Nemov et al., *Phys. Plasmas* (1999)

DESC is a new tool for stellarator optimization

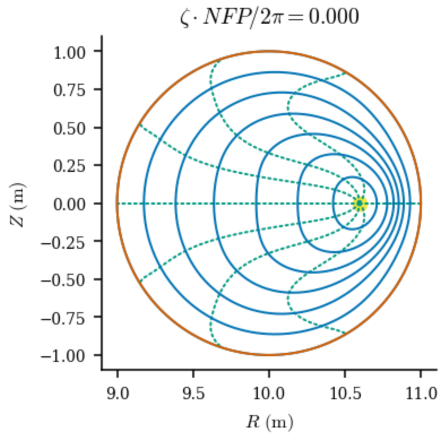
Accurate Equilibria



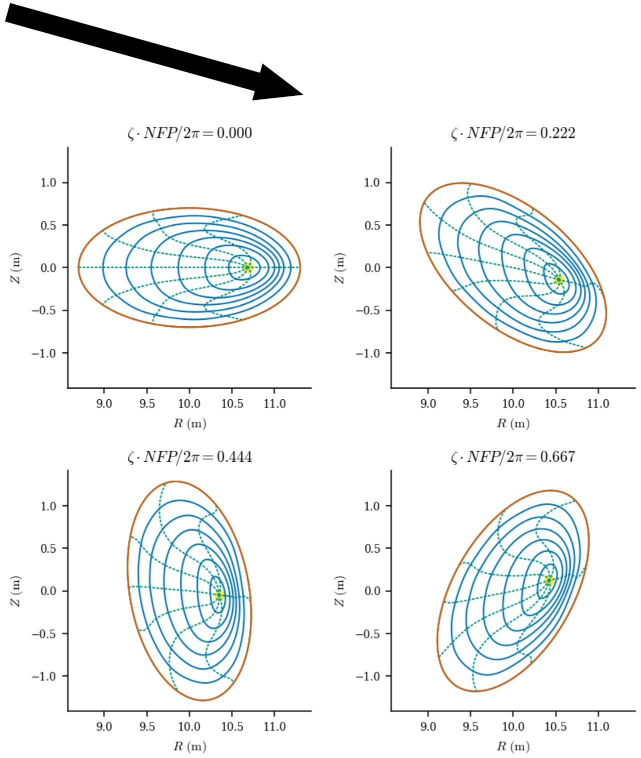
Fast Optimization



Phase-Space Connections



perturb tokamak into stellarator



Better tools = better stellarator reactors!

Additional Resources

Princeton Plasma Control
control.princeton.edu

Software

- Open-source repository: `https://github.com/PlasmaControl/DESC`
- Python package: `pip install desc-opt`

Papers

- The DESC Stellarator Code Suite Part I https://arxiv.org/abs/2203.17173
- The DESC Stellarator Code Suite Part II https://arxiv.org/abs/2203.15927
- The DESC Stellarator Code Suite Part III https://arxiv.org/abs/2204.00078

The Princeton Plasma Control group is recruiting graduate students and post-docs!

Contact Egemen Kolemen: ekolemen@pppl.gov

The MHD equilibrium equations are 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

equilibrium force balance equations evaluated in real space

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c}) = \boldsymbol{0}$$

errors evaluated at a set of collocation points

spectral coefficients

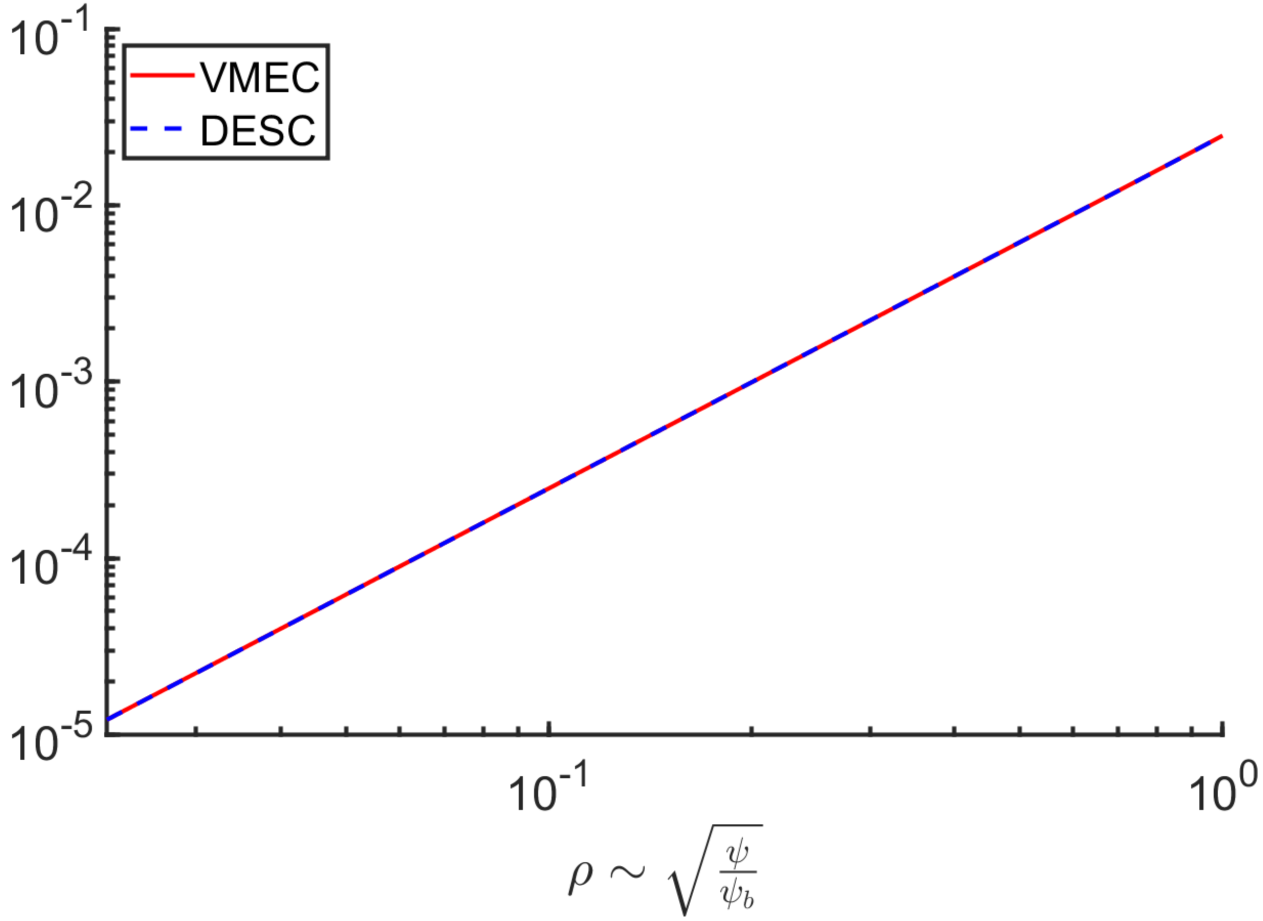
boundary conditions

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

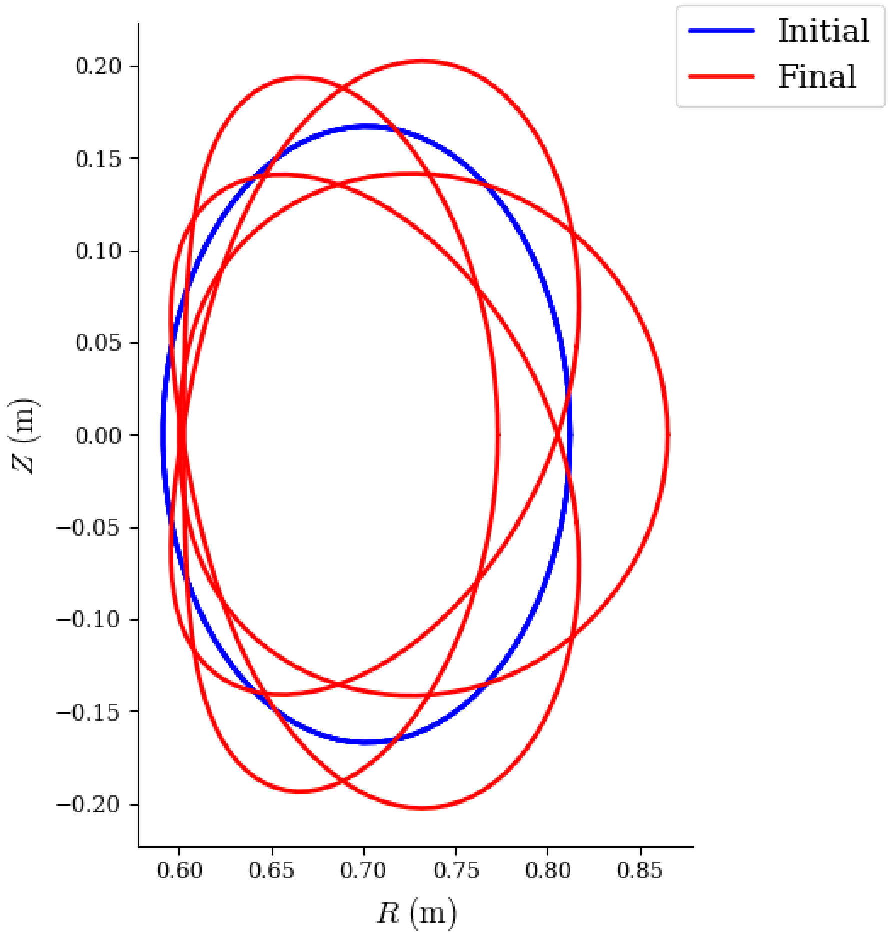
$$\boldsymbol{x}^* = \operatorname{argmin}_{\boldsymbol{x}}\left(\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c})\|^2\right) \qquad \dim(\boldsymbol{f}) \geq \dim(\boldsymbol{x})$$

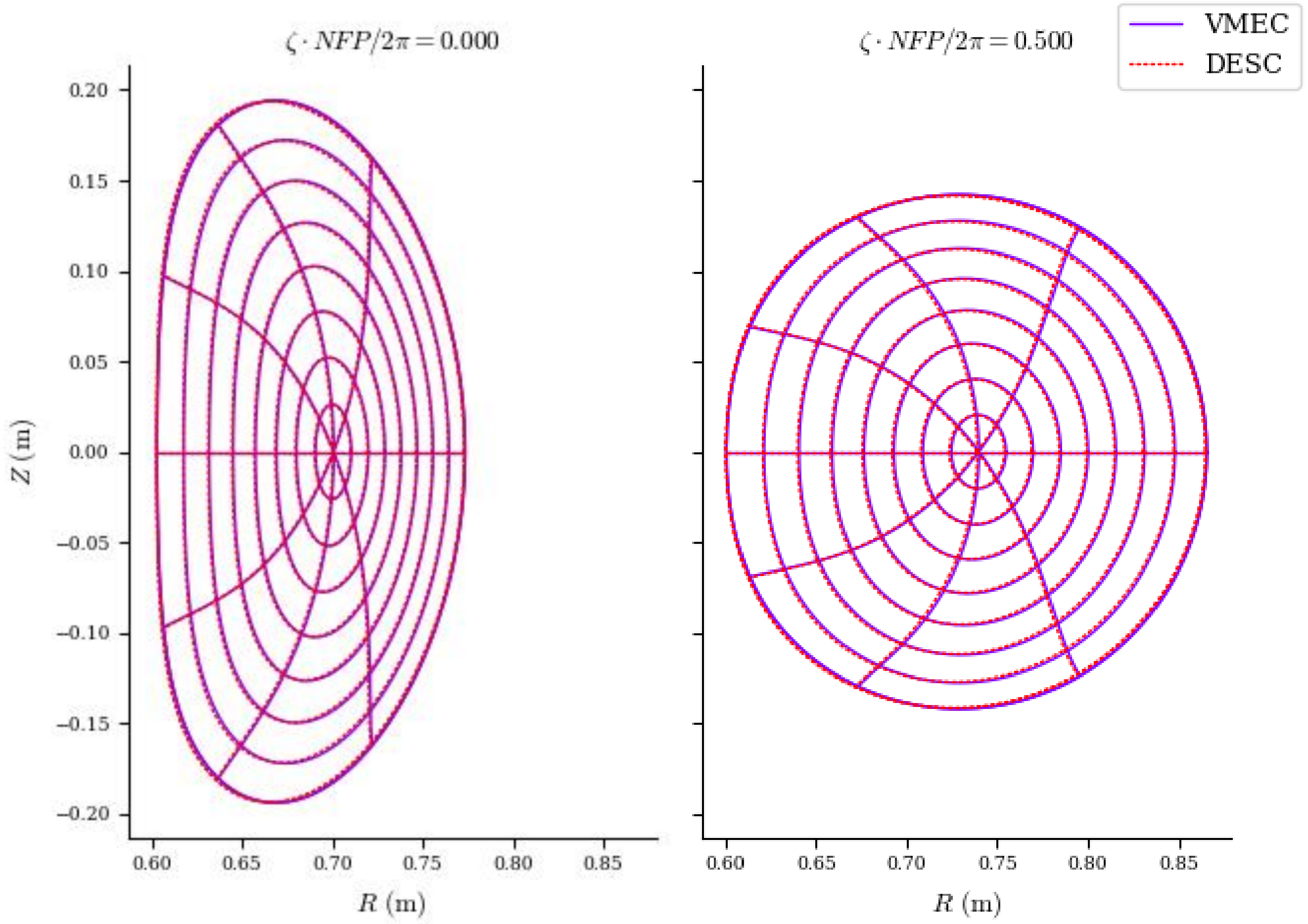
Spectrally condense λ without explicit constraint

$$\frac{\sum_{m,n} m^2\left(R_{mn}^2 + Z_{mn}^2\right)}{\sum_{m,n}\left(R_{mn}^2 + Z_{mn}^2\right)}$$



Free boundary comparison to VMEC

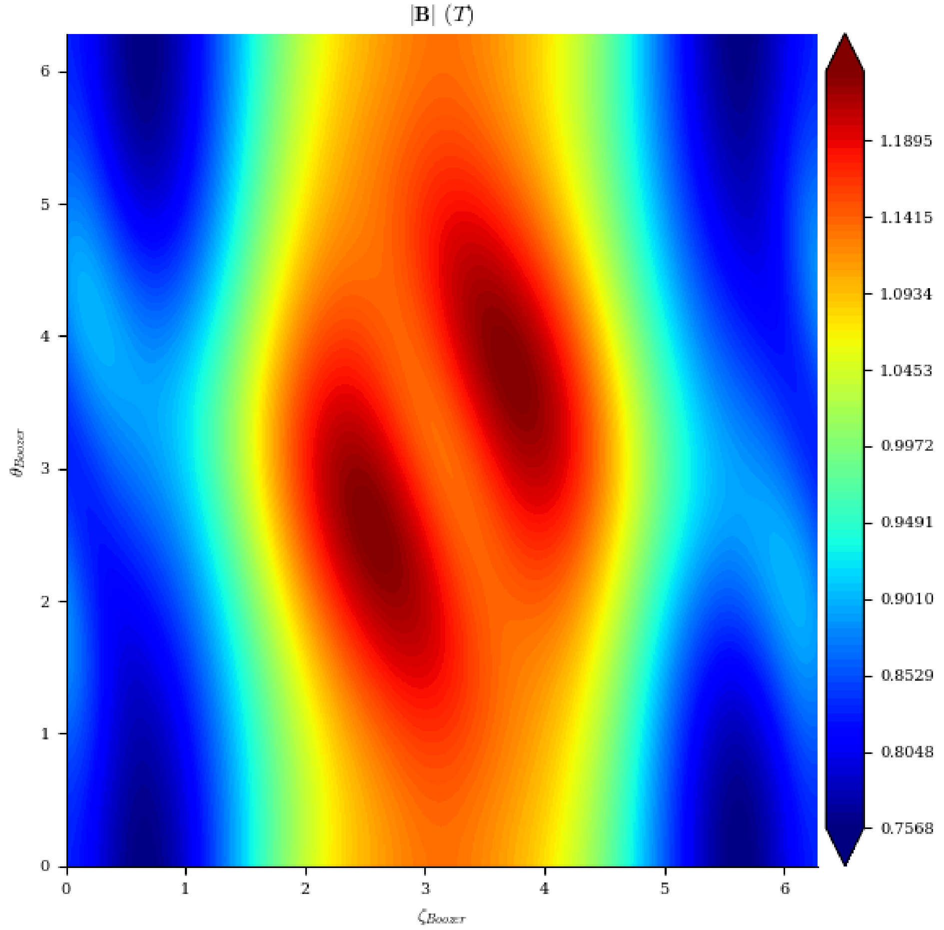




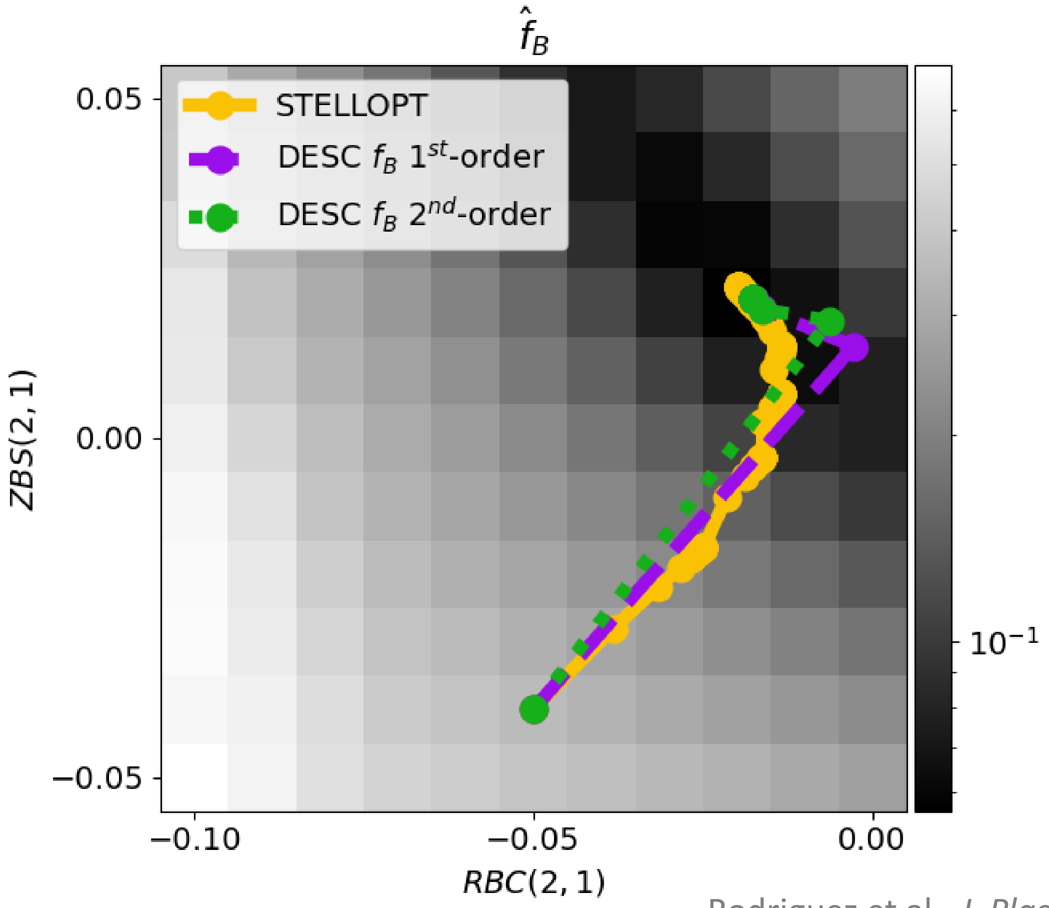
Near-axis expansions can be used as an initial guess for further optimization

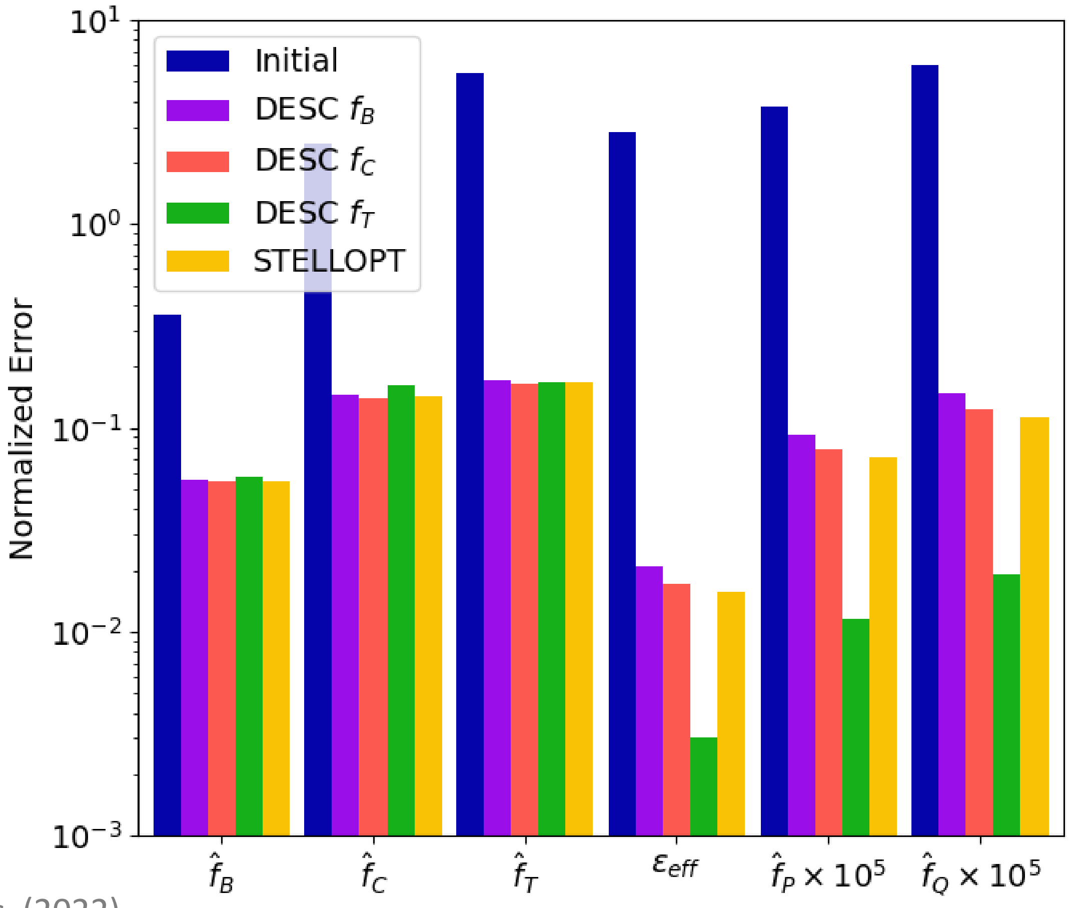
```
# using the pyQSC near-axis expansion code
stel = Qsc.from_paper("r2 section 5.4")
# load QH equilibrium solution
eq = Equilibrium.from_near_axis(stel, r=0.1, M=8, N=8)

# generate QI/QP boundary surface from model
surf = FourierRZToroidalSurface.from_near_axis(
    aspect_ratio=10, elongation=4,
    mirror_ratio=0.3, axis_Z=0.2)
# create vacuum configuration with boundary surface
eq = Equilibrium(M=8, N=8, surface=surf)
eq.solve()  # solve equilibrium force balance
```

Landreman et al., *J. Plasma Phys.* (2019)



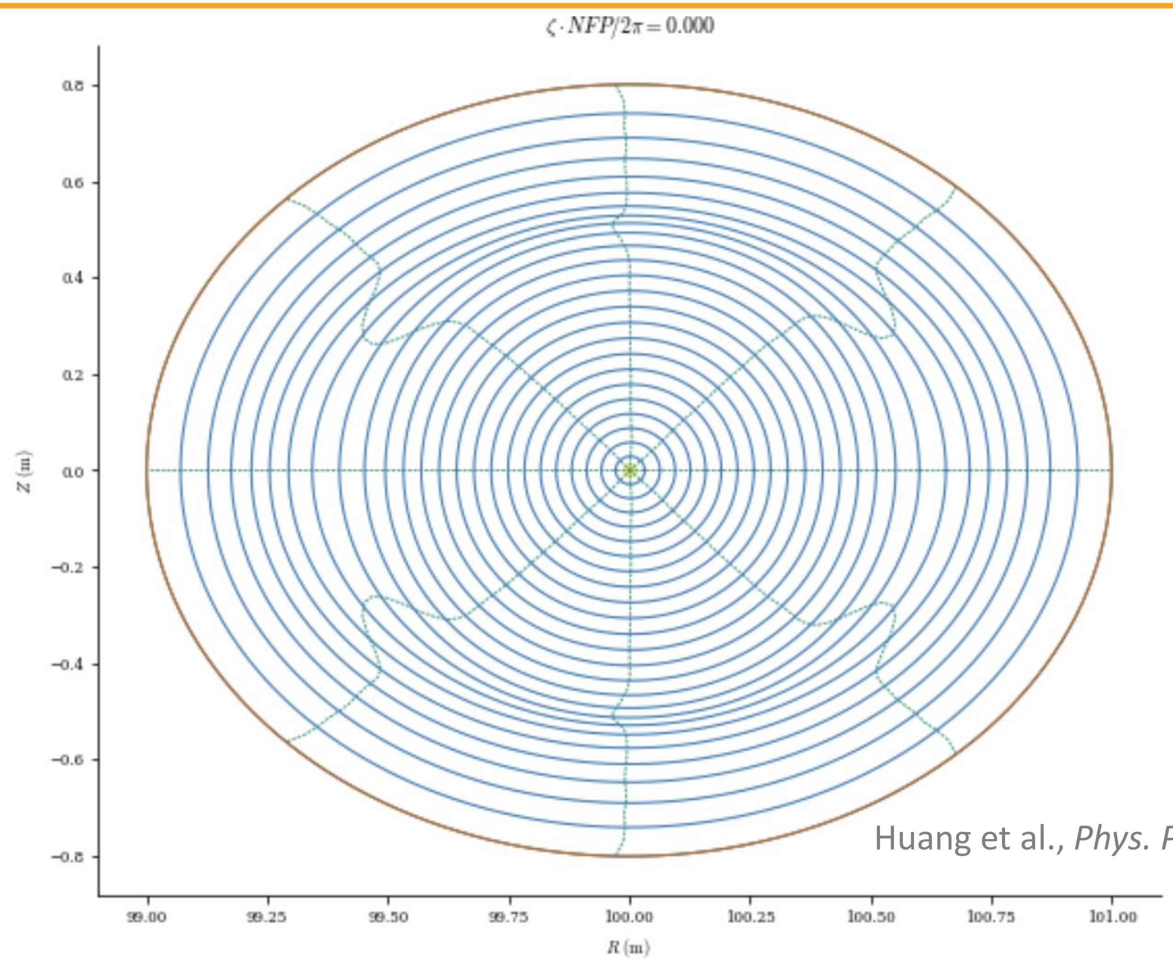
Optimization with various forms of QS





Rodriguez et al., *J. Plasma Phys.* (2022)

Ideal MHD behavior at rational surfaces with Hahm-Kulsrud-Taylor problem



Huang et al., *Phys. Plasmas* (2022)