Improvements to the DESC code for finding and optimizing stellarator equilibria

Rory Conlin (PU/PPPL)
Daniel Dudt (PU)
Dario Panici (PU)
Egemen Kolemen (PU/PPPL)

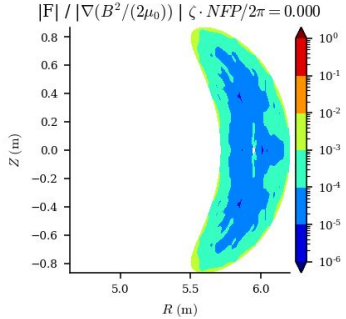
APS DPP 2021

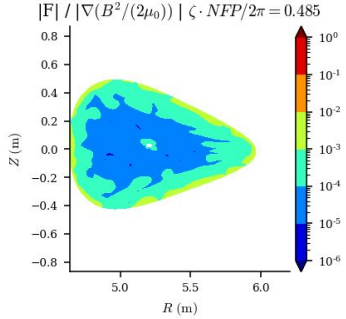
PRINCETON UNIVERSITY

What can DESC do?

High resolution, low error equilibria

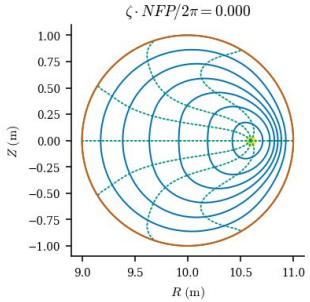
```
# solving an equilibrium
eq = Equilibrium(inputs)
eq.solve()
```



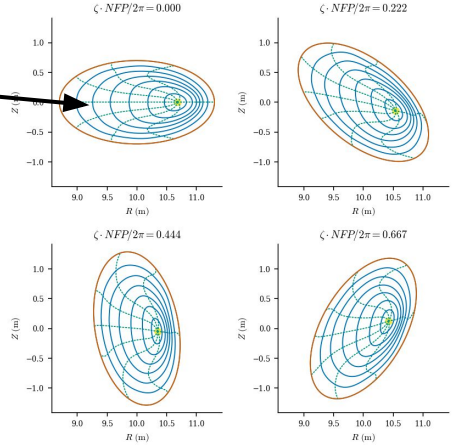


Transform tokamaks into stellarators

```
eq.perturb(dRb=delta_R_bdry,
           dZb=delta_Z_bdry,
           order=2)
```

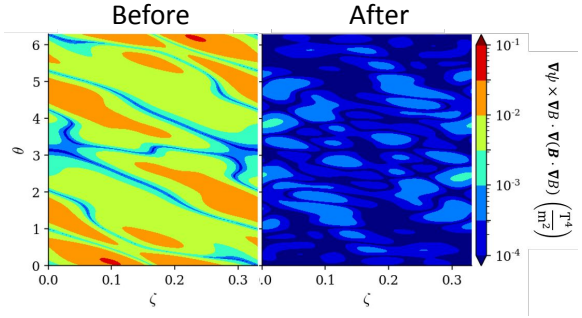


1 step



Optimize Quasi-symmetry

```
eq.perturb(objective=QS_fun,
           dRb=R_bdry_modes,
           dZb=Z_bdry_modes)
```



And more...

Equilibrium force balance is solved as a system of nonlinear equations

$$\mathbf{f} \equiv \mathbf{J} \times \mathbf{B} - \nabla p = 0$$

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

$$\boldsymbol{f} = \begin{bmatrix} f_\rho \\ f_\beta \\ BC \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} R_{lmn} \\ Z_{lmn} \\ \lambda_{lmn} \end{bmatrix} \quad \boldsymbol{c} = \begin{bmatrix} R^b_{mn} \\ Z^b_{mn} \\ p_l \\ \iota_l \\ \psi_a \end{bmatrix}$$

$$\boldsymbol{x}^* = \mathrm{argmin}_{\boldsymbol{x}}(\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{c})\|^2)$$

- Pseudo-spectral approach
  - Solution defined by global basis functions
  - Minimize residuals at collocation nodes
  - Exponential convergence if smooth
- Flexible choice of solver
  - Modified Newton methods, least-squares minimization, custom routines, etc.
  - Quadratic convergence near solution

Modern object-oriented code design

- Written in Python3
  - Object-oriented structure is easy to use, extend, and interface with other codes
  - Modular design allows flexibility for different solvers, objectives, etc.
- Uses JAX
  - Developed by Google, same backend as TensorFlow
  - Automatic differentiation for exact derivatives of arbitrary order
  - JIT compilation to CPU & GPU via XLA

Perturbations provide approximations to neighboring solutions

$\boldsymbol{f}$ = force balance error
$\boldsymbol{x}$ = state vector
$\boldsymbol{c}$ = input parameters

- First-order Taylor expansion:

$$\boldsymbol{f}(\boldsymbol{x}+\Delta\boldsymbol{x},\boldsymbol{c}+\Delta\boldsymbol{c}) = \cancelto{\sim 0}{\boldsymbol{f}(\boldsymbol{x},\boldsymbol{c})} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{c}}\Delta\boldsymbol{c} = \boldsymbol{0}$$

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

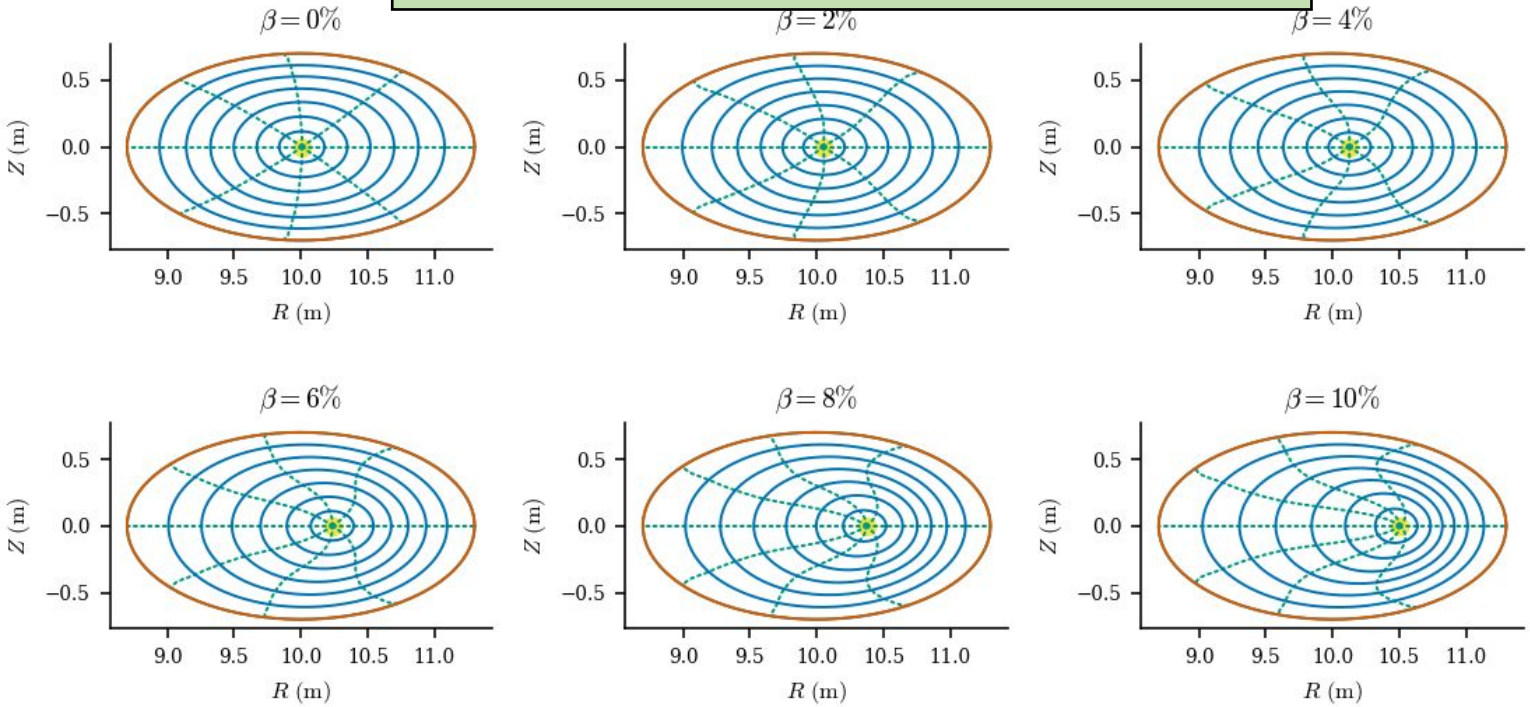
- Derivatives are computed exactly and efficiently with automatic differentiation
- Jacobian matrix $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is already computed in the final Newton iteration when solving the equilibrium
- Easily extended to higher order : $\Delta x = \varepsilon x_1 + \varepsilon^2 x_2 + \ldots$

Autodiff allows efficient computation of high order expansions

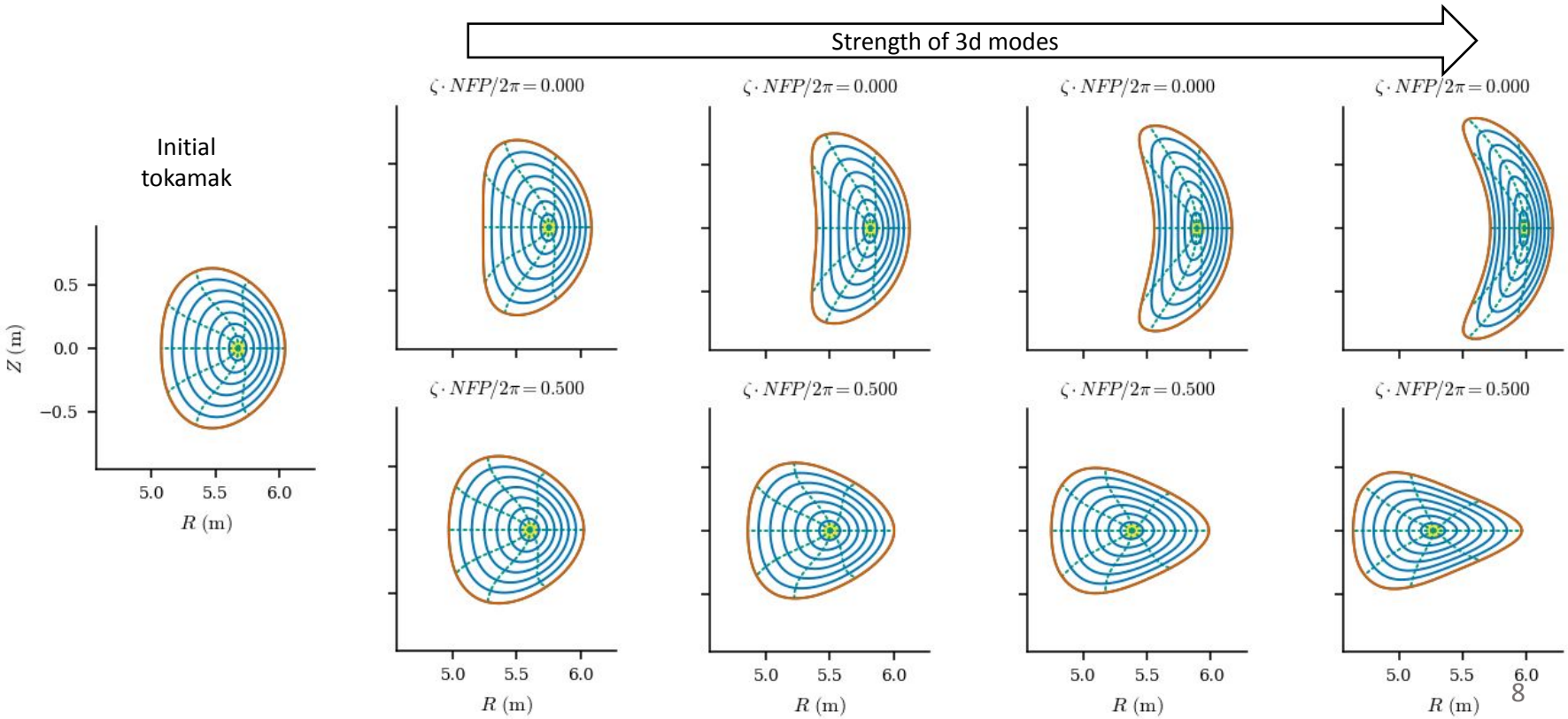
- Computing high order perturbations requires high order derivatives such as $\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{x}^2}, \frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{x} \partial \boldsymbol{c}}$ etc
- Materializing full 2nd derivative tensor $\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{x}^2}$ would take 100s-1000s GB of ram
- "Halley's method": only need directional derivatives: $\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{x}^2} \Delta \boldsymbol{x}_1 \Delta \boldsymbol{x}_1$
  - aka "Jacobian vector products"
- Can be computed using automatic differentiation with significantly reduced memory
- Implemented up to 3rd order expansion in the code

Example: perturbations compute parameter scans "for free"

```
delta_p = 1.8e3*np.array([1, -2, 1])
eq_new = eq.perturb(dp=delta_p,
order=1)
```



Example: continuously deforming tokamak into stellarator



Optimization objectives can be incorporated into the perturbations

- Define an optimization cost function $\boldsymbol{g} \equiv \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{c})$
  - Quasi-symmetry, coil complexity, etc.
- First-order Taylor expansion:

$$\boldsymbol{g}(\boldsymbol{x} + \Delta\boldsymbol{x}, \boldsymbol{c} + \Delta\boldsymbol{c}) = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{c}) + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \Delta\boldsymbol{x} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{c}} \Delta\boldsymbol{c} = \boldsymbol{0}$$

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

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

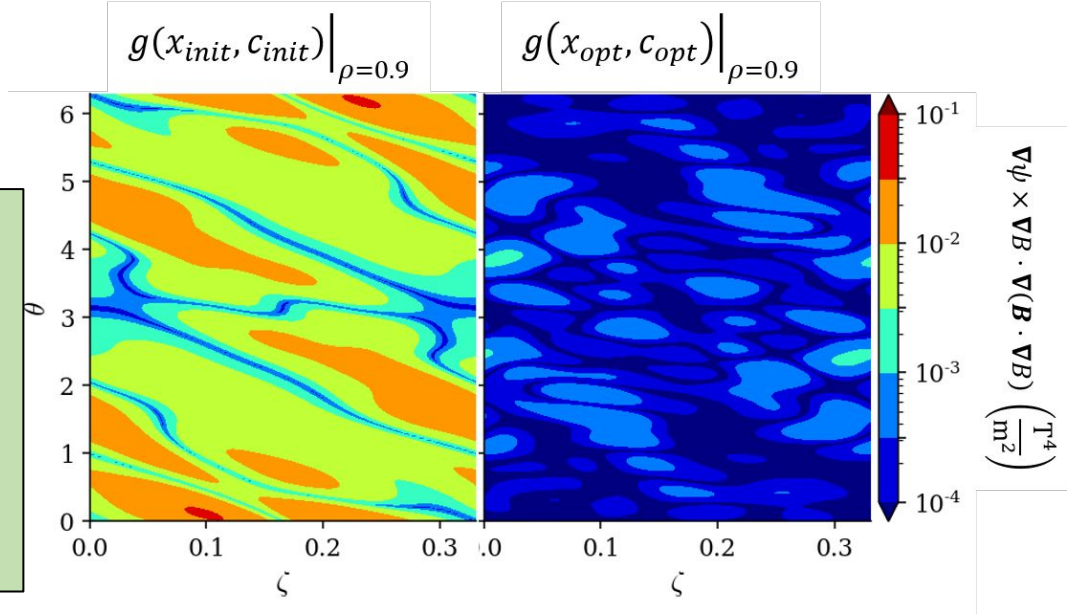
- This yields the perturbation $\Delta\boldsymbol{c}$ that most improves the objective while maintaining force balance
- No additional equilibrium solves required!

Example optimization objective: quasi-symmetry

- Optimized the boundary shape for quasi-symmetry at the ρ=0.9 flux surface

$$\boldsymbol{g}(\boldsymbol{x},\boldsymbol{c}) \equiv \nabla\psi \times \nabla B \cdot \nabla(\boldsymbol{B}\cdot\nabla B)^{1}$$

```
# optimization loop
for step in iterations:
    eq.perturb(
        objective=QS_fun,
        dRb=R_bdry_modes,
        dZb=Z_bdry_modes,
    )
# a few Newton iterations to
# make sure it's still converged
eq.solve(ftol=1e-2)
```



Thank You!

Publication: Dudt et. al. *Phys. Plasmas*. 2020.
Repository: https://github.com/PlasmaControl/DESC
Python Package: `pip install desc-opt`

Other DESC talks:

- BO08.00010 : Stellarator Optimization with DESC
- PP11.00086 : A Comparison of VMEC and DESC 3D Equilibrium Codes

Current & Future work

- Free boundary equilibria
- Equilibrium reconstruction
- Constraining toroidal current profile vs iota
- Alternate boundary conditions
- Couple with FOCUSADD for combined coil + plasma optimization
- Magnetic islands & stochastic regions

DESC was thoughtfully designed to be the ideal stellarator optimization code

- **Force balance** (instead of energy) minimization achieves lower error solutions
- **Newton methods** for solving systems of equations allows quadratic convergence
- **Pseudo-spectral** method allows exponential convergence
- **Zernike polynomial** basis functions properly resolve the magnetic axis
- **Python** code provides a modular framework that is easy to use
- **Automatic differentiation** supplies exact derivatives for solving and optimization
- **GPUs** are the future workhorses of high-performance computing

f(x,c) = 0 == equilibrium
x = magnetic fields
c = pressure, boundary, etc

$$
\begin{aligned}
f(x+\Delta x, c+\Delta c) &= f(x,c) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial c}\Delta c + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}\Delta x \Delta x^T \\
&+ \frac{1}{2}\frac{\partial^2 f}{\partial c^2}\Delta c \Delta c^T + \frac{\partial^2 f}{\partial x \partial c}\Delta x \Delta c^T + O(\Delta x^3)
\end{aligned}
$$

$$
\begin{aligned}
\Delta x &= \epsilon x_1 + \epsilon^2 x_2 \\
\Delta c &= \epsilon c_1
\end{aligned}
$$

Assume perturbation series for x

$$
\epsilon x_1 = -\left(\frac{\partial f}{\partial x}\right)^{-1}\left(f(x,c) + \frac{\partial f}{\partial c}\epsilon c_1\right)
$$

2nd order step computed using directional derivatives in direction given by 1st order terms

$$
\epsilon^2 x_2 = -\left(\frac{\partial f}{\partial x}\right)^{-1}\left(\frac{1}{2}\frac{\partial^2 f}{\partial x^2}\epsilon^2 x_1 x_1^T + \frac{1}{2}\frac{\partial^2 f}{\partial c^2}\epsilon^2 c_1 c_1^T + \frac{\partial^2 f}{\partial x \partial c}\epsilon^2 x_1 c_1^T\right)
$$

Least squares optimization

$$y = \frac{1}{2}\mathbf{f}^T\mathbf{f}$$

$$J = \frac{d\mathbf{f}}{d\mathbf{x}}$$

$$\mathbf{g} = \nabla y = \mathbf{f}^T\frac{d\mathbf{f}}{d\mathbf{x}} = \mathbf{f}^T J$$

$$H = \nabla^2 y = J^T J + \mathbf{f}^T\frac{d^2\mathbf{f}}{d\mathbf{x}^2}$$

- "small residual approximation"
  - $H \sim J^T J$
- even for large $|\mathbf{f}|$, $H \sim J^T J$ is often still useful
- Approximates convex hull of true solution landscape
- Helps to avoid stalling & local minima

Trust region method

$$\min_{\mathbf{p}} ||J\mathbf{p} - \mathbf{f}|| \quad s.t. \quad ||\mathbf{p}|| \leq r$$

Can be shown to be equivalent to:

$$(J^T J + \alpha I)\mathbf{p} = J^T \mathbf{f}$$

$$\alpha(||\mathbf{p}|| - r) = 0$$

So either full step lies in trust region, or reduces problem to 1d newton's method to find correct value of α