Implementing Quasisymmetric Surface Optimization in PyPlasmaOpt

John L. Ball

Aug. 7 2020

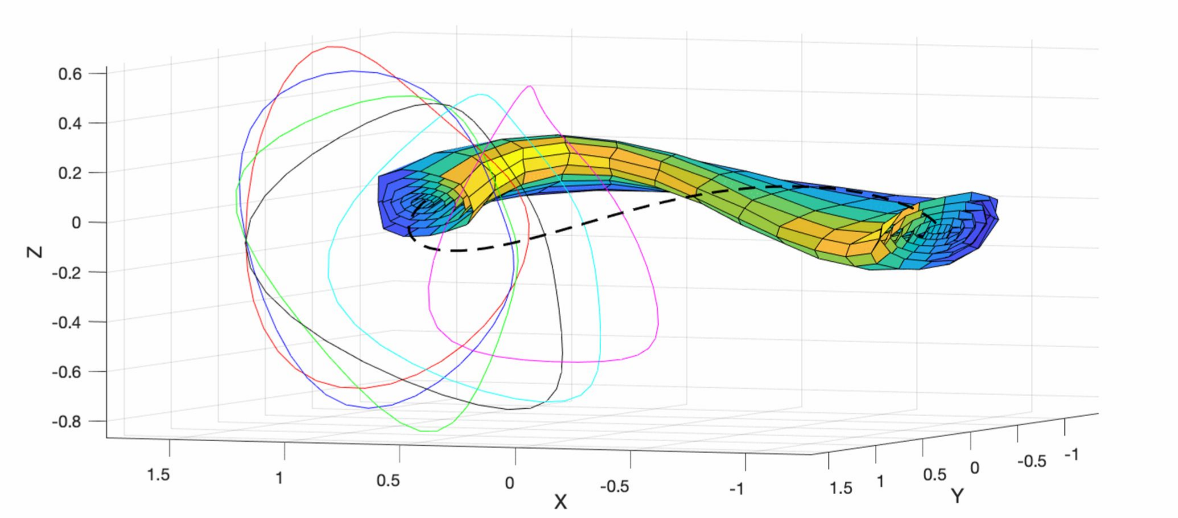


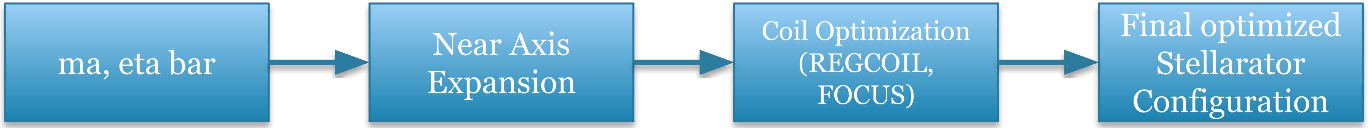
Figure courtesy A. Giuliani

PyPlasmaOpt: Overview

- Code for optimizing Stellarator vacuum field and coils
- Developed as part of Simons Foundation collaboration
- C++ and Python, connected with PyBind11

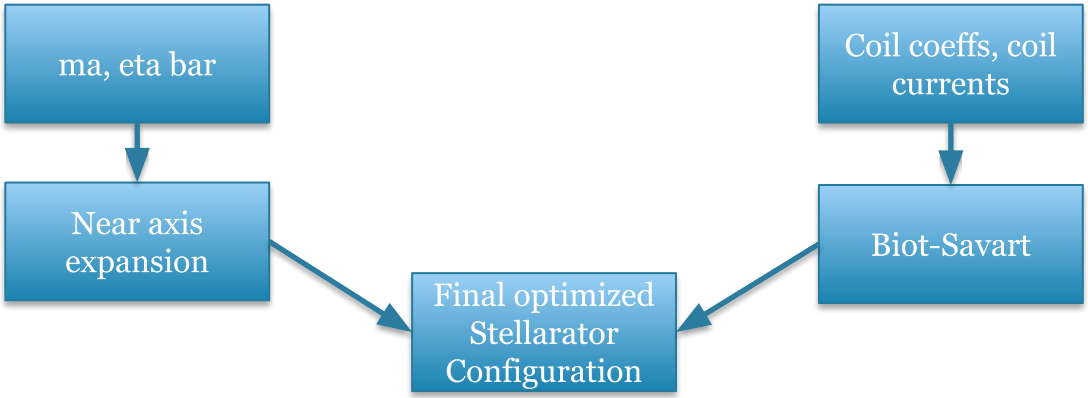
“Traditional” Approach to Stellarator Optimization

- Two distinct steps in optimization:
  - Optimization for physics properties (e.g. quasisymmetry, confinement)
  - Optimization for engineering properties (e.g. coil complexity, intercoil distance)
- Drawbacks – decoupling of physics and engineering could result in weaker overall optimization result.
- Maybe a different QS field has equivalent physics but superior engineering?



PyPlasmaOpt Approach to Stellarator Optimization

- Recent innovations in calculating QS fields analytically based on a near axis expansion (Landreman et al. 2019) allow for rapid calculations of desirable vacuum fields
  - DOFs: Magnetic axis, eta bar
- Field generated by initial simple coils using Biot-Savart
  - DOFs: Coil Fourier coefficients, coil currents, magnetic axis
- Generate an objective function that forces two fields to match as closely as possible, along with other desirable engineering / physics constraints



The PyPlasmaOpt Objective Function:

$$
\begin{aligned}
f(\mathbf{p}, \sigma, \iota_a, \mathbf{x}, \iota_s) = {} & \frac{1}{2}\left(\frac{L_c(\mathbf{p}) - L_{0,c}}{L_{0,c}}\right)^2 + \frac{1}{2}\left(\frac{L_a(\mathbf{p}) - L_{0,a}}{L_{0,a}}\right)^2 \\
& + \int_{\text{axis}} \|\boldsymbol{B}_{\text{coils}}(\mathbf{p}) - \boldsymbol{B}_{QS}(\mathbf{p})\|^2 \, dl + \int_{\text{axis}} \|\nabla \boldsymbol{B}_{\text{coils}}(\mathbf{p}) - \nabla \boldsymbol{B}_{QS}(\mathbf{p}, \sigma, \iota_a)\|^2 \, dl \\
& + \int_{\text{surface}} \left(\|\mathbf{B}_{\text{coils}}(\mathbf{x})\| - B_{\text{QS}}(\mathbf{p})\right)^2 \, da \\
& + \frac{1}{2}\left(\frac{\iota_a - \iota_{0,a}}{\iota_{0,a}}\right)^2 + \frac{1}{2}\left(\frac{\iota_s - \iota_{0,s}}{\iota_{0,s}}\right)^2
\end{aligned}
$$

where $\mathbf{g}_1(\mathbf{p}, \sigma, \iota_a) = \mathbf{0}$ and $\mathbf{g}_2(\mathbf{p}, \mathbf{x}, \iota_s) = \mathbf{0}.$

Courtesy A. Giuliani

Step 1: Boozer Coordinate Flux Surfaces

- Discretize PDEs using a spectral method
- Implement analytic jacobian calculation
- Solve using a newton method
- Ended up using a Levenberg-Marquardt implementation in Scipy
- 4 – 6 times speed up in residual / jacobian computation, only 2x speed increase in surface solve

$$\mathbf{B}_{\text{coils}} = \frac{B^2_{\text{coils}}}{G}\left(\frac{\partial \mathbf{x}}{\partial \theta} + \iota_s \frac{\partial \mathbf{x}}{\partial \varphi}\right)$$

$$\int_0^{2\pi}\int_0^{2\pi}\left\|\frac{\partial \mathbf{x}}{\partial \theta} \times \frac{\partial \mathbf{x}}{\partial \varphi}\right\| \, d\varphi \, d\theta = r$$

Step 2: Gradient of QS Surface Term in Obj Func.

- Use a forward sensitivity approach to calculate the gradient
- Currently implementing computation of sensitivity of surface residual wrt coil input parameters

$$\frac{\partial f}{\partial \mathbf{p}} = -\frac{\partial f}{\partial \mathbf{q}}\left[\frac{\partial \mathbf{g}}{\partial \mathbf{q}}^{-1} \boxed{\frac{\partial \mathbf{g}}{\partial \mathbf{p}}}\right] + \frac{\partial f}{\partial \mathbf{p}}$$

Acknowledgements:

- My SULI mentor, Dr. Stuart Hudson
- Dr. Andrew Giuliani and Dr. Florian Wechsung at NYU CIMS

This work was made possible by funding from the Department of Energy for the Summer Undergraduate Laboratory Internship (SULI) program. This work is supported by the US DOE Contract No. DE-AC02-09CH11466.