Automatic Differentiation & Unifying Stellarator Coil Design

Nick McGreivy

PhD Candidate
Program in Plasma Physics
Princeton University

March 1st 2021

I won't be talking about my paper

Optimized finite-build stellarator coils using automatic differentiation

N. McGreivy[1,2,*], **S.R. Hudson**[1] **and C. Zhu**[1]

[1] Princeton Plasma Physics Laboratory, Princeton University, PO Box 451, Princeton, NJ 08543-0451, United States of America
[2] Department of Astrophysical Sciences, Princeton University, Princeton, NJ, United States of America

E-mail: mcgreivy@princeton.edu

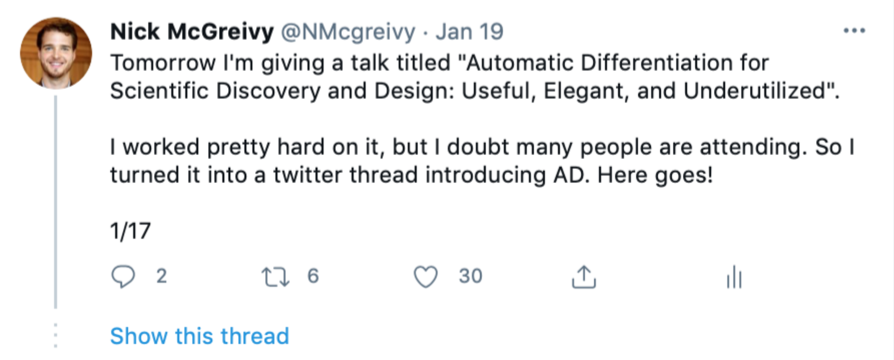
Received 29 August 2020, revised 13 November 2020
Accepted for publication 24 November 2020
Published 13 January 2021

CrossMark

Abstract
A new stellarator coil design code is introduced that optimizes the position and winding pack orientation of finite-build coils. The new code, called flexible optimized curves in space using automatic differentiation (AD) and finite build (FOCUSADD), performs gradient-based optimization in a high-dimensional, non-convex space. The derivatives with respect to parameters of finite-build coils are easily and efficiently computed using AD. FOCUSADD parametrizes coil positions in free space using a Fourier series and uses a multi-filament approximation to the coil winding pack. The orientation of the winding pack is parametrized with a Fourier series and can be optimized as well. Optimized finite-build coils for a Wendelstein 7-X (W7-X)-like stellarator are found, and compared with filamentary coil results. The final positions of optimized finite-build W7-X-like coils are shifted, on average, by approximately 2.5 mm relative to optimized filamentary coils. These results suggest that finite-build effects should be accounted for in the optimization of stellarators with low coil tolerances.

Recap of my last talk

Automatic Differentiation for Scientific Discovery and Design: Useful, Elegant, and Underutilized



How is AD elegant? (Recap)

Automatic differentiation with `grad`

JAX has roughly the same API as Autograd. The most popular function is `grad` for reverse-mode gradients:

```
from jax import grad
import jax.numpy as jnp

def tanh(x):  # Define a function
  y = jnp.exp(-2.0 * x)
  return (1.0 - y) / (1.0 + y)

grad_tanh = grad(tanh)  # Obtain its gradient function
print(grad_tanh(1.0))   # Evaluate it at x = 1.0
# prints 0.4199743
```

Why is AD useful? (Recap)

1. Simplicity

- Finding and programming analytic derivatives is time-consuming and error-prone; meanwhile finite-difference is inefficient.

2. Ideal for gradient-based optimization

- Reverse mode AD computes the gradient of a scalar function in time $\mathcal{O}(1)$. This is as efficient as the best analytic methods.

3. Effortless gradients

- Easy to rapidly prototype new ideas and objectives.

New Paradigm in Coil Design

Instead of thinking about *how* to compute gradients of an objective function, we think of *what* objective function to optimize.

How I conclude my last talk

Nick McGreivy @NMcgreivy · Jan 19

A new paradigm in stellarator coil design is a good start, but I think we can do better. Since the beginning of time, we've optimized the stellarator plasma geometry first, then the coils second. The next big kahuna, however, is optimizing both at the same time.

15/17

Combined coil-plasma optimization requires passing gradients between plasma solvers and coil codes:

$$\frac{df}{d\text{coil}} = \frac{\partial f}{\partial \text{coil}} + \frac{\partial f}{\partial \text{plasma}} \frac{\partial \text{plasma}}{\partial \text{coil}}$$

1 2

Nick McGreivy @NMcgreivy · Jan 19

We need to pass gradients between nonlinear plasma solvers, which afaik require analytic adjoint methods, and coil codes, which are amenable to AD.

It's not my job to design or program the state-of-the-art optimization code people have been talking about. But if I were...

16/17

1 2

Nick McGreivy @NMcgreivy · Jan 19

...I'd design it as a collection of modular primitive operations, which can be composed arbitrarily and chained together using AD. Under this framework, adjoint methods are implemented as primitives. Each primitive then forms building blocks of the unified optimization.

17/17

1 2

Unifying Stellarator Coil Design

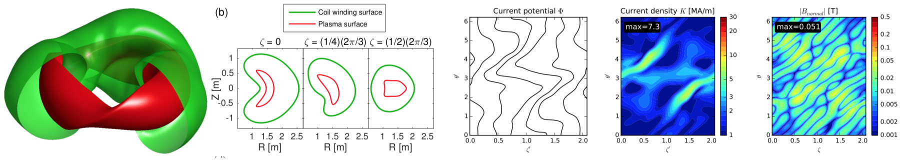
The state of the field

We have created many coil design codes, each of which has its own strengths and weaknesses. But none of them are clearly better than the others to design a stellarator. Therefore, when we are ready to design an actual stellarator, we should unify these approaches to coil design by developing a new approach which incorporates the best elements of each existing approach.

What's the point?

The point, besides introducing an interesting idea, is to show another example of what AD makes possible.

REGCOIL (w/ Winding Surface Optimization)



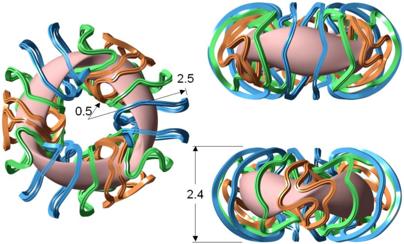
Pros

- Convex, fast
- Having a winding surface facilitates the engineering process
- Principled approach to choosing winding surface

Cons

- Doesn't account for discreteness of coils
- Doesn't account for finite build of coils

COILOPT



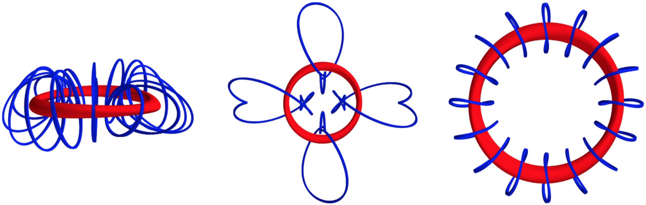
Pros

- Directly optimizes for discreteness of coils
- Having a winding surface facilitates the engineering process

Cons

- Finite-difference derivatives, inefficient
- Doesn't account for finite build of coils
- Non-convex

FOCUS



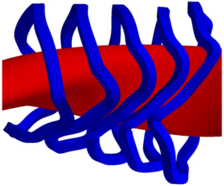
Pros

- Directly optimizes for discreteness of coils
- Efficient derivatives and gradient-based optimization
- Adds additional freedom to coil shapes by eliminating winding surface

Cons

- No winding surface, which may be undesirable from an engineering perspective
- Does not account for finite build of coils
- Non-convex

FOCUSADD



Pros

- Directly optimizes for finite-build of coils
- Efficient, easy derivative computation

Cons

- No winding surface, which may be undesirable from an engineering perspective
- Defining winding pack frame with good engineering properties for free curve is challenging
- Non-convex

Code	WS	D	FB	C	EG	AR
REGCOIL	✓	✗	✗	✓	✓	✗
COILOPT	✓	✓	✗	✗	✗	✓
FOCUS	✗	✓	✗	✗	✓	✗
FOCUSADD	✗	✓	✓	✗	✓	✓
unified	✓	✓	✓	✗	✓	✓

- WS = has winding surface
- D = discrete coils are directly optimized
- FB = finite-build coils are directly optimized
- C = convex
- EG = efficient gradients (if nonconvex) or efficient optimization (if convex)
- AR = an arbitrary regularization penalty can be added, either to the winding surface or to the coils

Comments

- Whether having a winding surface is a ‘pro’ or a ‘con’ can be debated; I take the view that the winding surface makes engineering design simpler.
- FOCUSADD is meant for illustration and extensibility, while the other three codes are more polished, flexible, and production-ready. Don’t let the green checks deceive you.
- Defining a finite-build frame in FOCUSADD with uniformly good engineering properties has been challenging; the unified approach solves these issues.

Unified Approach

First, define a winding surface:

$$\boldsymbol{\Omega} \equiv \text{Winding Surface Parameters}$$

Then, place coil filaments on the winding surface using a Fourier series:

$$\boldsymbol{\Theta} \equiv \text{Coil Fourier Parameters}$$

Use the filaments to define a tangent vector and the winding surface to define a coil normal vector; this defines a finite build frame. From the finite-build frame, use the Biot-Savart law to compute an objective function f. Add some regularization penalty to f. Then use automatic differentiation and gradient-based optimization to minimize the objective function with respect to the coil parameters $\boldsymbol{p} \equiv \{\boldsymbol{\Omega}, \boldsymbol{\Theta}\}$:

$$\boldsymbol{p}^* = \arg\min_{\boldsymbol{p}} f(\boldsymbol{p})$$

Bottom Line

Multi-filament coils on a winding surface, optimized using AD.

Thanks for listening!

Questions?