Redesigning Stellarator Coil Design

Nick McGreivy[1,2] Stuart Hudson[1]

[1]Princeton Plasma Physics Laboratory

[2]PhD Candidate
Program in Plasma Physics
Princeton University

Stellarator Seminar Series
April 16th 2020

Work inspired by Ryan Adams, Ammar Hakim, Caoxiang Zhu, Elizabeth Paul, Matt Landreman, and Jim-Felix Lobsien

Derivatives have been used in stellarator coil design for optimization and sensitivity analysis

For example:

- D.J. Strickler, L.A. Berry, & S.P. Hirshman (2002). Designing coils for compact stellarators *Fusion Sci. Technology*.
- T. Brown, J. Breslau, D. Gates, N. Pomphrey, & A. Zolfaghari (2015). *IEEE 26th Symp. on Fusion Engineering*.
- Caoxiang Zhu, Stuart R. Hudson, Yuntao Song, & Yuanxi Wan (2017). New method to design stellarator coils without the winding surface *Nuclear Fusion*.
- Caoxiang Zhu, Stuart R Hudson, Samuel A Lazerson, Yuntao Song, & Yuanxi Wan (2018). Hessian matrix approach for determining error field sensitivity to coil deviations *Plasma Physics and Controlled Fusion*.
- Caoxiang Zhu, Stuart R Hudson, Yuntao Song, & Yuanxi Wan (2018). Designing stellarator coils by a modified Newton method using FOCUS *Plasma Physics and Controlled Fusion*.
- E.J. Paul, M. Landreman, A. Bader, & W. Dorland (2018). An adjoint method for gradient-based optimization of stellarator coil shapes *Nuclear Fusion*.
- Matt Landreman & Elizabeth Paul (2018). Computing local sensitivity and tolerances for stellarator physics properties using shape gradients *Nuclear Fusion*.
- Hudson, S., Zhu, C., Pfefferle, D., & Gunderson, L. (2018). Differentiating the shape of stellarator coils with respect to the plasma boundary *Physics Letters. A*.
- Caoxiang Zhu, David A. Gates, Stuart R. Hudson, Haifeng Liu, Yuhong Xu, Akihiro Shimizu, & Shoichi Okamura (2019). Identification of important error fields in stellarators using the Hessian matrix method *Nuclear Fusion*.

We can compute derivatives three ways

1. Finite-difference derivatives (numerical derivatives)

- For an N-dimensional function, finite-difference requires $N + 1$ function evaluations to get the N-dimensional gradient
- Inexact due to truncation and round-off errors
- Simple

We can compute derivatives three ways

1. Finite-difference derivatives (numerical derivatives)

- For an N-dimensional function, finite-difference requires $N + 1$ function evaluations to get the N-dimensional gradient
- Inexact due to truncation and round-off errors
- Simple

2. Analytic derivatives

- Either computed by hand or with symbolic differentiation
- Needs to be programmed by a human
- Efficient

3. Automatic Differentiation (AD)

Also known as algorithmic differentiation or computational differentiation

- Automatic Differentiation (AD) is a technology for automatically computing the exact numerical derivatives of any differentiable function, including arbitrarily complex simulations, represented by a computer program.
- AD has two modes, forward mode AD and reverse mode AD.

Important facts about AD

- Suppose $\boldsymbol{y} = f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$. Then first-order AD computes the numerical value of the Jacobian $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ at a particular value of $\boldsymbol{x}$.
- Suppose f takes time T to compute.
- Forward mode AD computes the Jacobian of f in time $\mathcal{O}(nT)$.
- Reverse mode AD computes the Jacobian of f in time $\mathcal{O}(mT)$.
- To compute automatic derivatives, you need to program f using an AD software tool. As we will see, this is remarkably easy.

Why is AD useful?

1. Simplicity

- Finding analytic derivatives is time-consuming and often hard.
- Programming those derivatives is time-consuming.
- AD removes these steps.

Why is AD useful?

1. Simplicity

- Finding analytic derivatives is time-consuming and often hard.
- Programming those derivatives is time-consuming.
- AD removes these steps.

2. Ideal for gradient-based optimization

- For a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ which has time-complexity $\mathcal{O}(T)$, reverse mode AD computes the gradient in time $\mathcal{O}(T)$. This is exactly as efficient as the best analytic methods.

Why is AD useful?

1. Simplicity

- Finding analytic derivatives is time-consuming and often hard.
- Programming those derivatives is time-consuming.
- AD removes these steps.

2. Ideal for gradient-based optimization

- For a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ which has time-complexity $\mathcal{O}(T)$, reverse mode AD computes the gradient in time $\mathcal{O}(T)$. This is exactly as efficient as the best analytic methods.

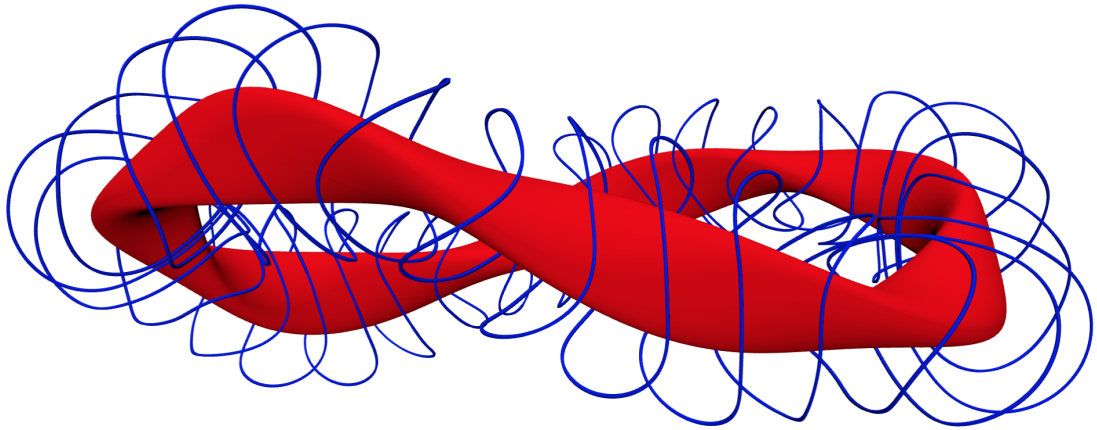
3. Effortless gradients

- Easy to rapidly prototype new ideas and objectives.

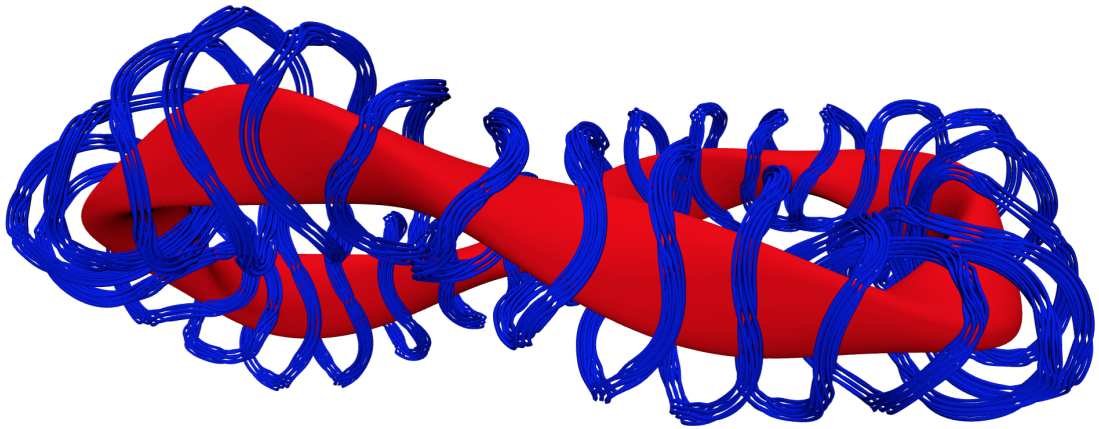
"AD cuts through complex calculations like butter" -Tony Qian

Finding the analytic derivatives of finite-build coils is a very difficult task. AD makes it nearly trivial, and has led to the first finite-build stellarator coil design code.

FOCUS (C. Zhu, S. R. Hudson, Y. Song, Y. Wan 2017)



FOCUSADD (N. McGreivy, S. R. Hudson 2020)



Stellarator coil optimization should be a 2-step process instead of a 1-step process

I will return to this later in the talk

- Step 1: Optimize the stellarator position and shape with respect to physics and engineering objectives
- Step 2: Keep the coil shape fixed, allow the coil to move and rotate freely in space, and optimize only the physics objectives subject to those degrees of freedom.

To my knowledge, so far we have only done step 1

Freeze coil shapes, for each coil define $\boldsymbol{r}_{COM}$ and Euler angles (θ, ϕ, ψ), and optimize those 6 degrees of freedom. AD makes step 2 easy and efficient.

Why would you not compute derivatives with AD?

- You have a massive legacy code which you can't rewrite.
- You need derivatives with respect to a variable which isn't continuous.
- You've already written your code and it efficiently computes analytic derivatives, so why change it? (Next time though, use AD...)

This is a talk in two parts

Organization of talk (35 more minutes without questions, please ask questions)

- Part I: How does automatic differentiation work? (20 minutes)
- Part II: Cool new things I've done with automatic differentiation (15 minutes)

This is a talk in two parts

Organization of talk (35 more minutes without questions, please ask questions)

- Part I: How does automatic differentiation work? (20 minutes)
- Part II: Cool new things I've done with automatic differentiation (15 minutes)

What I'm not covering in this talk but could cover in a future AD-only talk

- How AD tools are implemented
- The mathematical foundations of and various ways of formalizing AD
- Second-order derivatives and derivatives to arbitrary order
- Automatic differentiation of non-linear discretized PDEs and fixed point iterations
- Checkpointing methods for reducing memory consumption of reverse mode
- The relationship between AD and the adjoint method
- Where else in plasma physics could AD be useful?

Part I: What is Automatic Differentiation?

Let's take a derivative

Suppose we have the following function f:

$$f(x, y) = \sin(xy) + x^2/y$$

An abstract view

f takes two variables, x and y, and combines them using elementary operations (multiply, divide, sin, square, add) to compute the output of the function.

Let's take a derivative

Suppose we have the following function f:

$$f(x, y) = \sin(xy) + x^2/y$$

What if we want $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$? Chain rule!

$$\frac{\partial f}{\partial x} = \cos(xy)y + 2x/y$$

$$\frac{\partial f}{\partial y} = \cos(xy)x - x^2/y^2$$

Let's take a derivative

Suppose we have the following function f:

$$f(x, y) = \sin(xy) + x^2/y$$

What if we want $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$? Chain rule!

$$\frac{\partial f}{\partial x} = \frac{\partial (xy)}{\partial x}\frac{\partial \sin(xy)}{\partial (xy)} + \frac{\partial (x^2)}{\partial x}\frac{\partial (x^2/y)}{\partial (x^2)}$$

$$\frac{\partial f}{\partial y} = \frac{\partial (xy)}{\partial y}\frac{\partial \sin(xy)}{\partial (xy)} + \frac{\partial (1/y)}{\partial y}\frac{\partial (x^2/y)}{\partial (1/y)}$$

The chain rule

Why does the chain rule work?

The chain rule works because we know the partial derivatives of each elementary operation in our function.

For example, we know that the derivative of the elementary operation sine with respect to it's input is cosine.

$$\frac{\partial f}{\partial x} = \frac{\partial (xy)}{\partial x} \frac{\partial \sin(xy)}{\partial (xy)} + \frac{\partial (x^2)}{\partial x} \frac{\partial (x^2/y)}{\partial (x^2)}$$

The chain rule

Why does the chain rule work?

The chain rule works because we know the partial derivatives of each elementary operation in our function.

For example, we know that the derivative of the elementary operation sine with respect to it's input is cosine.

$$\frac{\partial f}{\partial x} = \frac{\partial (xy)}{\partial x} \frac{\partial \sin(xy)}{\partial (xy)} + \frac{\partial (x^2)}{\partial x} \frac{\partial (x^2/y)}{\partial (x^2)}$$

Automatic Differentiation (AD)

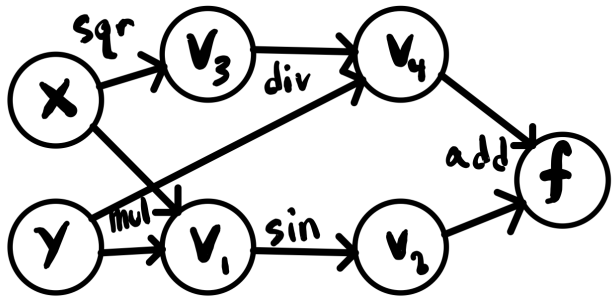
As we'll see, AD works on this exact principle.

How does AD work?

Step 1: Compute function

The AD tool computes the function f, one elementary operation at a time. A representation of the 'computational graph' is built.

$$f(x, y) = \sin(xy) + x^2/y$$



Step 2: Compute derivatives in the reverse order

Compute the derivative of the output of f with respect to each variable v_i by traversing the graph in reverse order. This is exactly the chain rule, applied in a clever way.

$$\frac{\partial f}{\partial v_i} = \sum_{\substack{j \in \text{children} \\ \text{of } i}} \frac{\partial f}{\partial v_j} \frac{\partial v_j}{\partial v_i}$$

The details

Step 1:

$x = 2.0$
$y = 3.0$
$v_1 = x * y = 6.0$
$v_2 = \sin(v_1) = -0.297$
$v_3 = x^2 = 4.0$
$v_4 = v_3/y = 1.333$
$f = v_2 + v_4 = 1.054$

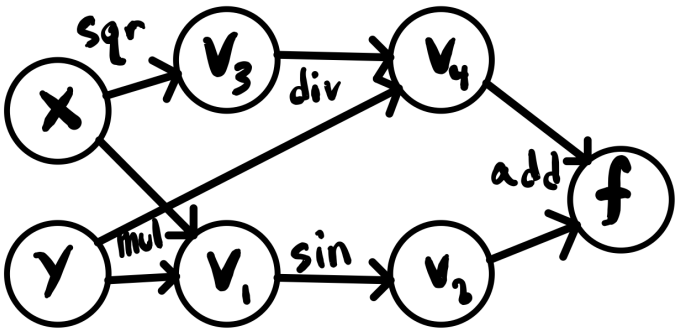
Step 2:

$\frac{\partial f}{\partial f} = 1.0$
$\frac{\partial f}{\partial v_4} = 1.0$
$\frac{\partial f}{\partial v_3} = \frac{\partial f}{\partial v_4}\frac{\partial v_4}{\partial v_3} = 0.333$
$\frac{\partial f}{\partial v_2} = 1.0$
$\frac{\partial f}{\partial v_1} = \frac{\partial f}{\partial v_2}\frac{\partial v_2}{\partial v_1} = 0.960$

Step 2, continued:

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v_1}\frac{\partial v_1}{\partial y} + \frac{\partial f}{\partial v_4}\frac{\partial v_4}{\partial y}$
$\frac{\partial f}{\partial y} = 1.476$
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial v_3}\frac{\partial v_3}{\partial x} + \frac{\partial f}{\partial v_1}\frac{\partial v_1}{\partial x}$
$\frac{\partial f}{\partial x} = 4.214$

$$f(x, y) = \sin(xy) + x^2/y$$



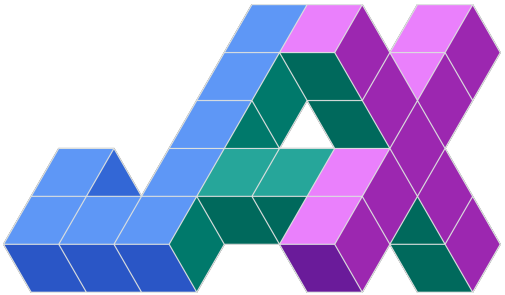
$$\frac{\partial f}{\partial v_i} = \sum_{\substack{j \in \text{children} \\ \text{of } i}} \frac{\partial f}{\partial v_j}\frac{\partial v_j}{\partial v_i}$$

Next up: AD tools and how to use them

Any questions on how AD works?

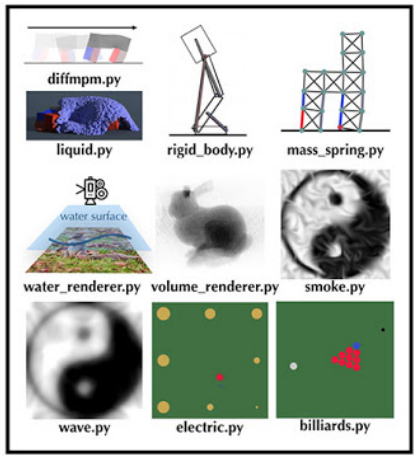
AD Tools

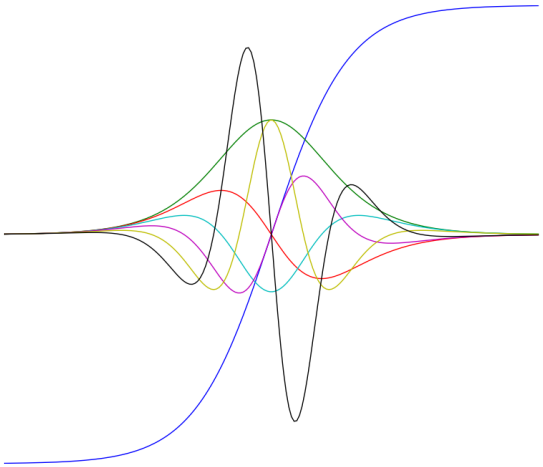
Effortless gradients



julia

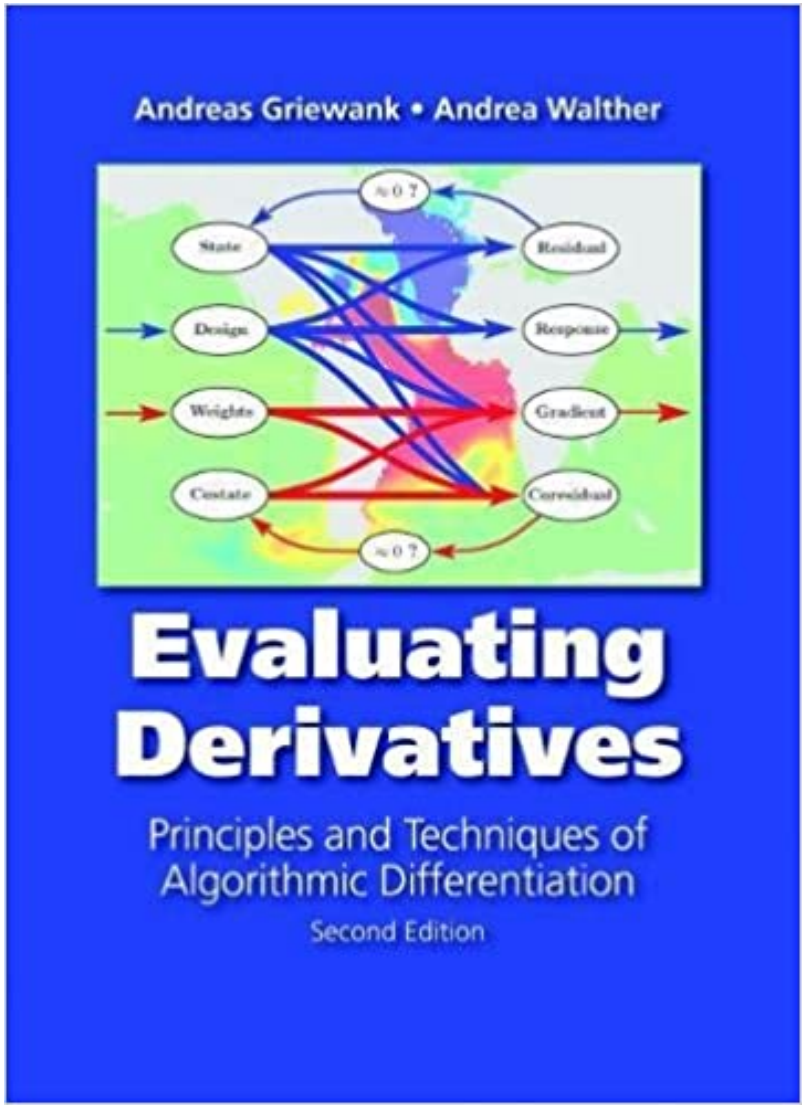
$$\frac{\partial}{\partial x}$$

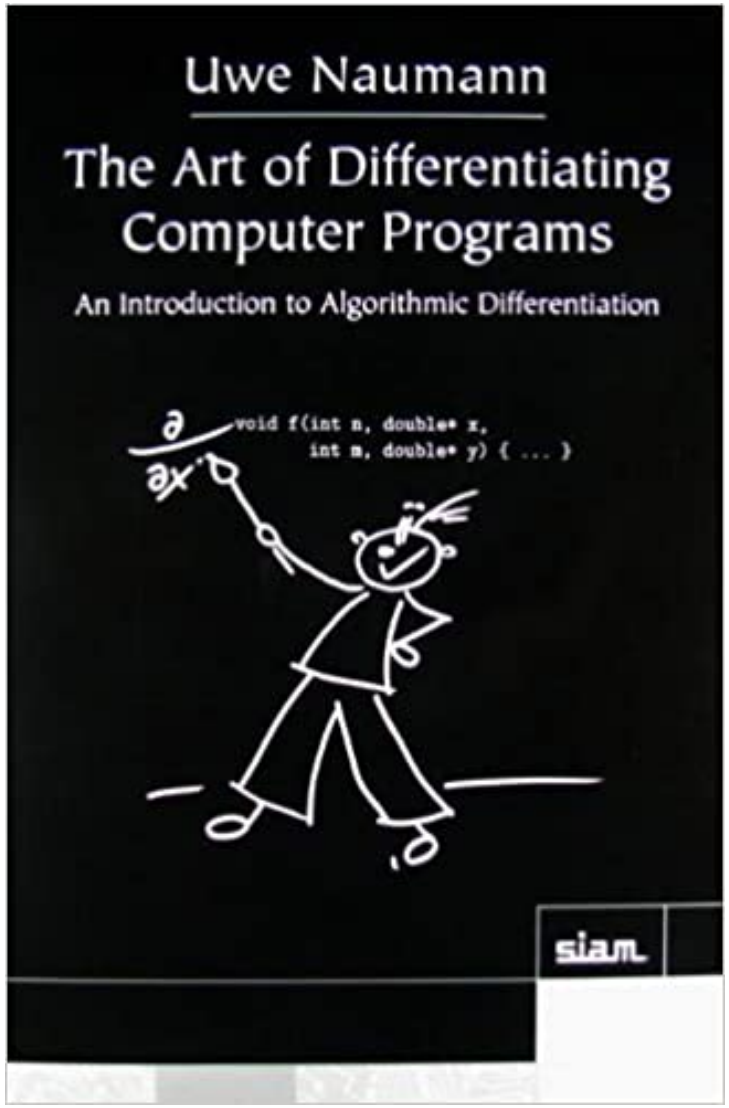




dolfin-adjoint

AD is a well-studied technique





Theory developed throughout 80s, 90s, 00s. Mostly by Germans.

A brief modern history of AD

- 2009: Nobody in ML is using AD
- 2013-14: Everybody in ML is using AD
- 2015: Tensorflow (Google)
- 2015: Autograd (Dougal Maclaurin, Ryan Adams, Matt Johnson, David Duvenaud, Harvard)
- 2016: PyTorch (Facebook)
- 2018: AD formalized as program transformation of functional programs (Conal Elliot)
- 2018: JAX (Google, Dougal Maclaurin, Matt Johnson)

Dougal Maclaurin

Ryan Adams

My timeline

- Fall 2017: Start PhD
- Fall 2018: Ryan Adams joins Princeton Faculty
- Fall 2019: COS597 Advanced Topics in Automatic Differentiation (Ryan Adams, 4 CS grad students, and me)

Ryan Adams

AD in machine learning (ML):

A 2009 blog post made a convincing argument that ML researchers should use AD.

Criticisms of blog post:

- Computing derivatives distracts from what the field actually wants to accomplish.
- Valuable researcher time is wasted.
- Leads to preference for functions they are capable of manually deriving gradients for.

AD was eventually fully adopted by the ML community.

Should AD be partially adopted by the stellarator optimization community? It's worth considering.

This is an enormously complex question. I'd love to discuss this in depth after the talk or offline.

Should AD be partially adopted by the stellarator optimization community? It's worth considering.

This is an enormously complex question. I'd love to discuss this in depth after the talk or offline.

Pros

- Gradient-based optimization of high-dimensional non-convex objective functions has been successful in many domains.
- AD and the adjoint method work together particularly well.
- If $N = 50$, does a factor of 10-20 increase in computational speed matter?
- Exact derivative needed?
- Much easier to rewrite an existing code we understand than write a new code.

Should AD be partially adopted by the stellarator optimization community? It's worth considering.

This is an enormously complex question. I'd love to discuss this in depth after the talk or offline.

Pros

- Gradient-based optimization of high-dimensional non-convex objective functions has been successful in many domains.
- AD and the adjoint method work together particularly well.
- If $N = 50$, does a factor of 10-20 increase in computational speed matter?
- Exact derivative needed?
- Much easier to rewrite an existing code we understand than write a new code.

Cons

- Is there sufficient demand for rewriting STELLOPT with an AD tool? Does our team have the right expertise?
- The stellarator community seems to like FORTRAN. Bad for AD.
- Does the right tool exist?
- Is gradient-free Bayesian optimization better? Bayesian optimization with gradients? Do we have resources to try all the above?
- Memory manageable?

Next up: Cool new things I've done with AD

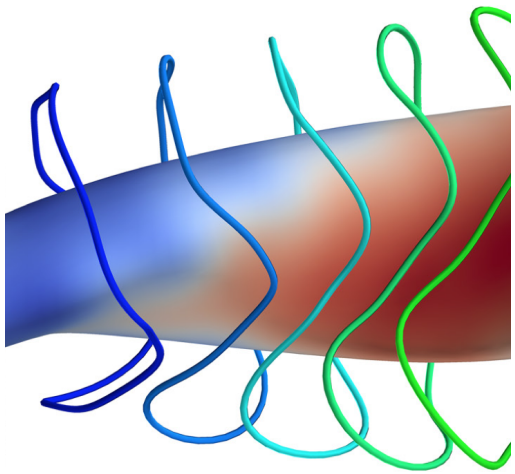
What are your questions?

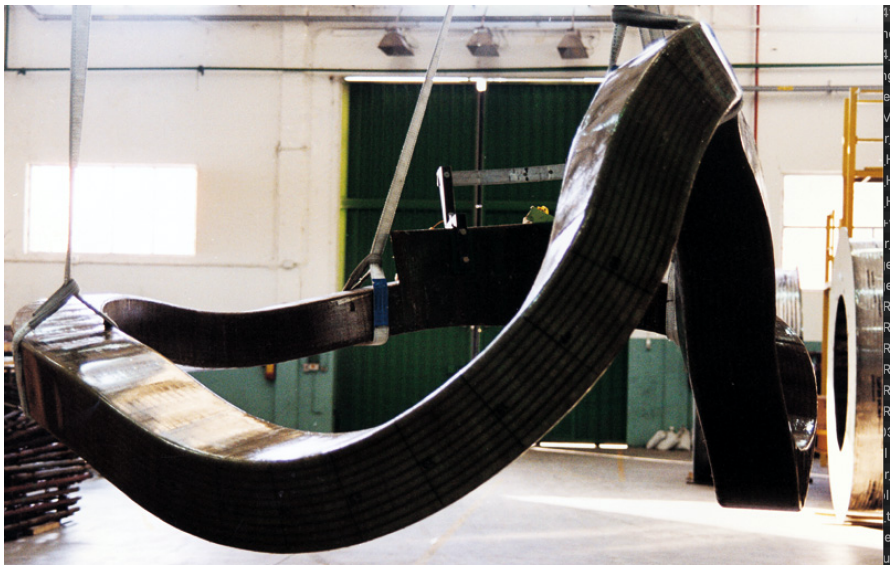
Part II:
Cool new things I've done with AD

Finite-build stellarator coils

Motivation:

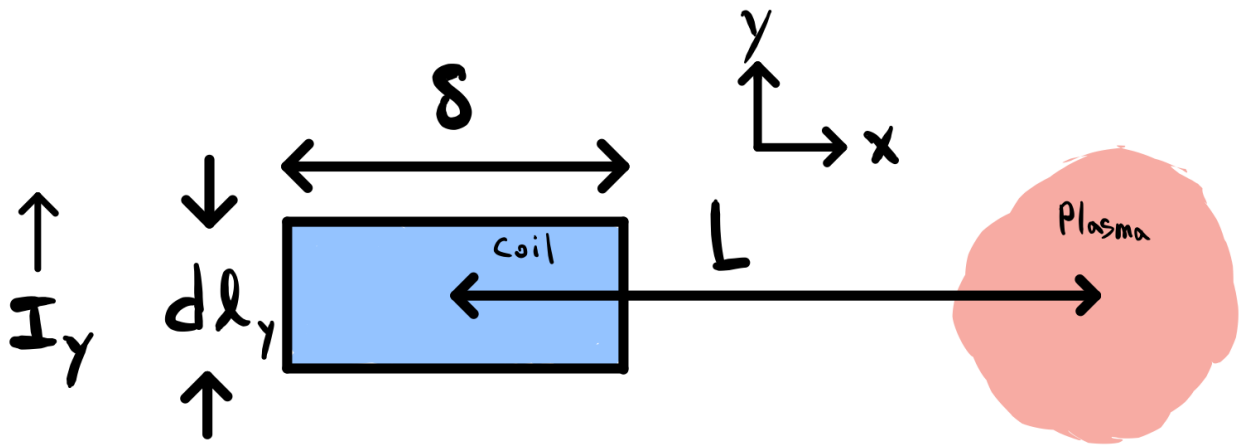
- How much does including finite coil thickness change optimized coils?





Deviations in the magnetic field are second-order in the coil thickness divided by the coil-plasma distance

Suppose we have a coil carrying current in the y-direction with thickness δ a distance L away from our plasma.



$$\text{Biot-Savart: } \frac{dB_z}{d\ell_y} = -\frac{\mu_0 I_y}{4\pi}\int_{-\delta/2}^{\delta/2}\frac{dx}{(L+x)^2}$$

$$dB_z \approx -\frac{\mu_0 I_y d\ell_y}{4\pi L^2}\int_{-\delta/2}^{\delta/2}(1-\cancelto{0}{\frac{2x}{L}}+\frac{3x^2}{L^2})dx \approx dB_{filament}(1+\frac{\delta^2}{4L^2})$$

Analytic magnetic fields for circular coils

From Static and Dynamic Electricity (1950) by W.R. Smythe, p. 270-271, we have

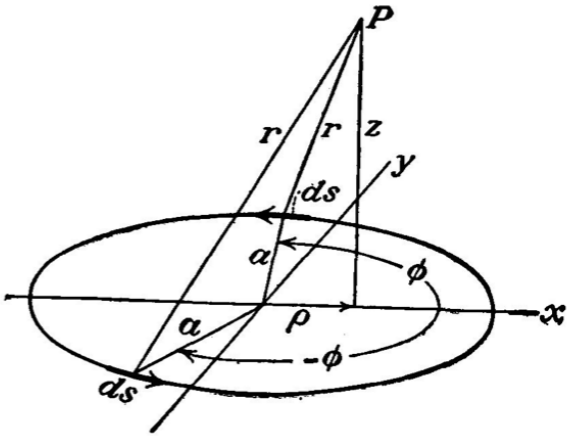


FIG. 7.10.

$$B_\rho = \frac{\mu I}{2\pi} \frac{z}{\rho[(a+\rho)^2+z^2]^{\frac{1}{2}}}\left[-K + \frac{a^2+\rho^2+z^2}{(a-\rho)^2+z^2}E\right]$$

$$B_z = \frac{\mu I}{2\pi} \frac{1}{[(a+\rho)^2+z^2]^{\frac{1}{2}}}\left[K + \frac{a^2-\rho^2-z^2}{(a-\rho)^2+z^2}E\right]$$

where $K(k)$ and $E(k)$ are complete elliptic integrals of the first and second kind and $k^2 \equiv 4a\rho/[(a+\rho)^2+z^2]$. In the plane of the coil, we have $z = 0$ and $B_\rho = 0$, giving

$$B_z = \frac{\mu_0 I}{2\pi(a+\rho)}\left[K(k) + \frac{a+\rho}{a-\rho}E(k)\right]$$

Taylor expand in the $z = 0$ plane

$$B_z = \frac{\mu_0 I}{2\pi(a+\rho)}\left[K(k) + \frac{a+\rho}{a-\rho}E(k)\right]$$

We can rewrite this in terms of $\epsilon \equiv \rho/a$ and use ϵ as an expansion parameter.

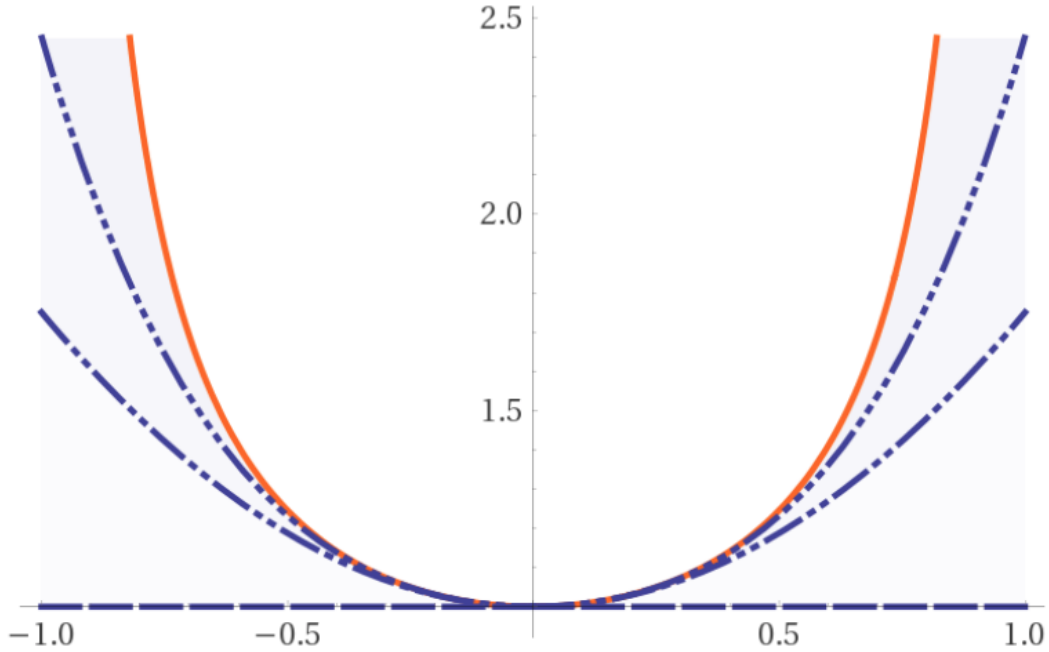
$$B_z = \frac{\mu_0 I}{2\pi a}\left[\frac{1}{1+\epsilon}K(k) + \frac{1}{1-\epsilon}E(k)\right] \quad k^2 = \frac{4\epsilon}{(1+\epsilon)^2}$$

Since k^2 is small, we can use $K(k) = \frac{\pi}{2}(1 + \frac{1}{4}k^2 + \frac{9}{64}k^4 + \dots)$ and $E(k) = \frac{\pi}{2}(1 - \frac{1}{4}k^2 - \frac{3}{64}k^4 - \dots)$. Working out the expansion gives us

$$B_z \approx \frac{\mu_0 I}{2\pi a}\left[1 + \frac{3\epsilon^2}{4} + \frac{45\epsilon^4}{64}\right]$$

How good of an approximation is this?

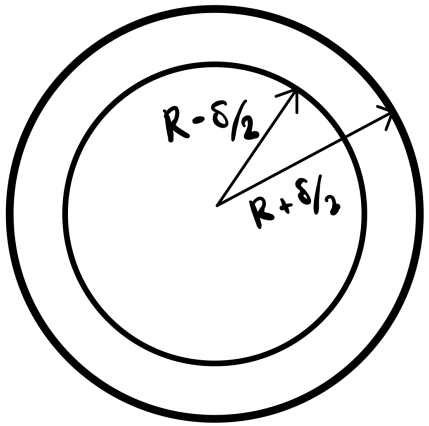
$$B_z \approx \frac{\mu_0 I}{2a}\left[1 + \frac{3\epsilon^2}{4} + \frac{45\epsilon^4}{64}\right]$$



This approximation is robust for $\epsilon = \rho/a \lessapprox 0.6$.

From filamentary coils to finite-build coils

Suppose our coil is an annulus of radius R and thickness δ carrying total current I with constant volumetric current density. Then the magnetic field at $z = 0$ and radius ρ is an integral over dB_z from $r = R - \delta/2$ to $r = R + \delta/2$.



$$B_z \approx \frac{\mu_0 I}{2R\delta} \int_{r=R-\delta/2}^{r=R+\delta/2} \left(1 + \frac{3x^2}{4} + \frac{45x^4}{64}\right) dr$$

Expanding this integral in δ/R, to lowest order this is

$$B_z \approx \frac{\mu_0 I}{2R}\left[1 + \frac{3\rho^2}{4R^2}\left(1 + \frac{\delta^2}{4R^2}\right) + \frac{45\rho^4}{64R^4}\left(1 + \frac{5\delta^2}{6R^2}\right)\right]$$

Is a second-order effect a problem?

$L \equiv$ minimum coil-plasma distance

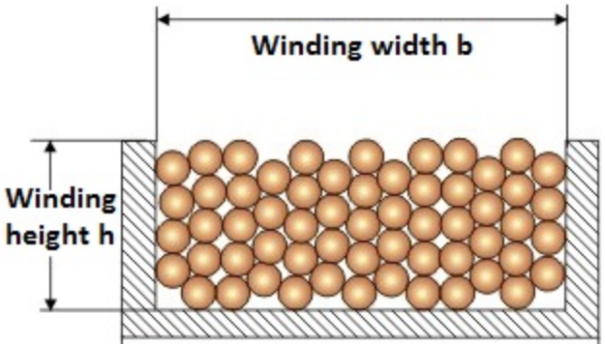
$\delta \equiv$ Coil thickness

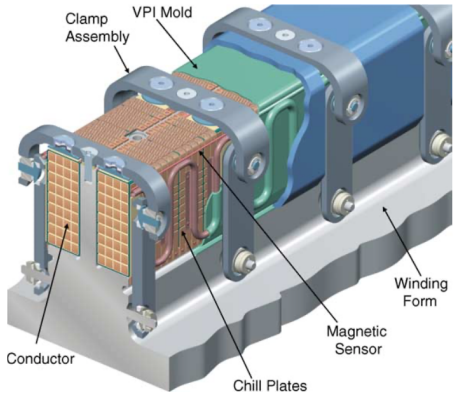
Stellarator	Minor Radius	L	δ	$\delta^2/4L^2$
W7-X	50cm	37cm	20cm	0.073
NCSX	35cm	20cm	12cm	0.09

While 7-9% would be a significant effect, this is a worst-case estimate.
The true correction is much smaller.

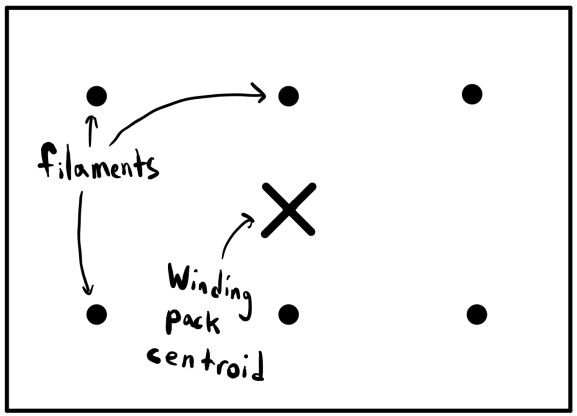
Multi-filament approximation to coil winding pack

Real coils are wound in a so-called 'winding pack'.





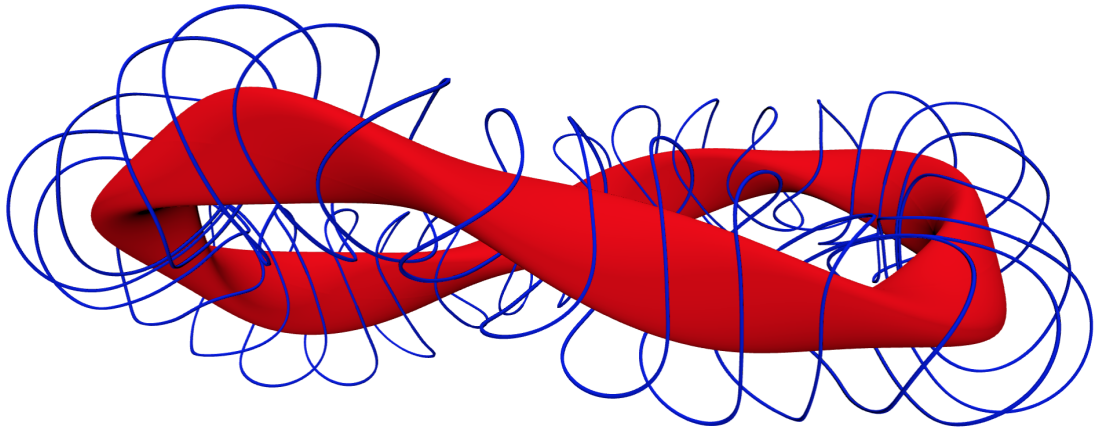
One option: 'multi-filament' approximation to the coil winding pack. The filaments are placed on a grid defined by the Frenet frame surrounding the coil centroid. Mathematical details in the 'additional slides'.



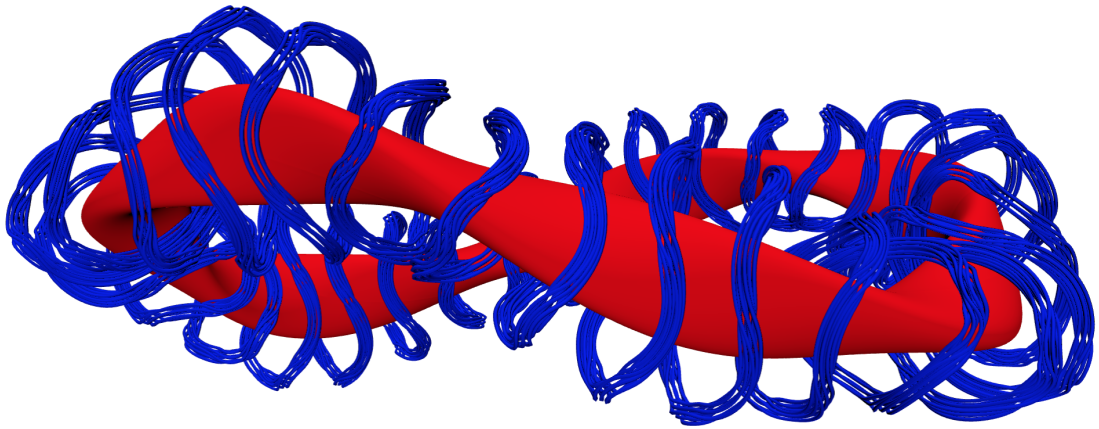
A group at Wisconsin has been exploring finite-width coils: New method for the optimization of finite-width stellarator coils Luquant Singh, T. Kruger, A. Bader, C. Zhu, S.R. Hudson, and D.T. Anderson. *(Preprint)*

FOCUSADD: **F**lexible **O**ptimized **C**oils **U**sing **S**pace curves with **A**utomatic **DD**ifferentiation

FOCUS (C. Zhu, S. R. Hudson, Y. Song, Y. Wan 2017)



FOCUSADD (N. McGreivy, S. R. Hudson 2020)



FOCUSADD: Magnetic field given by Biot-Savart law

Magnetic field is sum of magnetic field generated by external currents and magnetic field generated by currents in the plasma. FOCUSADD currently ignores the plasma field, for simplicity.

$$\boldsymbol{B}(\boldsymbol{r}) = \boldsymbol{B}_{vacuum}(\boldsymbol{r}) + \cancelto{0}{\boldsymbol{B}_{plasma}(\boldsymbol{r})}$$

The magnetic field generated by external currents is given by the Biot-Savart law.

$$\boldsymbol{B}_{vacuum}(\boldsymbol{r}) = \sum_{i=1}^{N_c} \sum_{n=1}^{N_1} \sum_{b=1}^{N_2} \mu_0 I_{n,b}^i \oint \frac{d\boldsymbol{l}_{n,b}^i \times (\boldsymbol{r} - \boldsymbol{r}_{n,b}^i)}{|\boldsymbol{r} - \boldsymbol{r}_{n,b}^i|^3}$$

FOCUSADD: Objective function

The optimization is performed using gradient descent on an objective function f_{total}, given by a sum of physics objectives and engineering objectives.

$$f_{total}(\boldsymbol{p}) = f_{phys}(\boldsymbol{p}) + \lambda_{eng} f_{eng}(\boldsymbol{p})$$

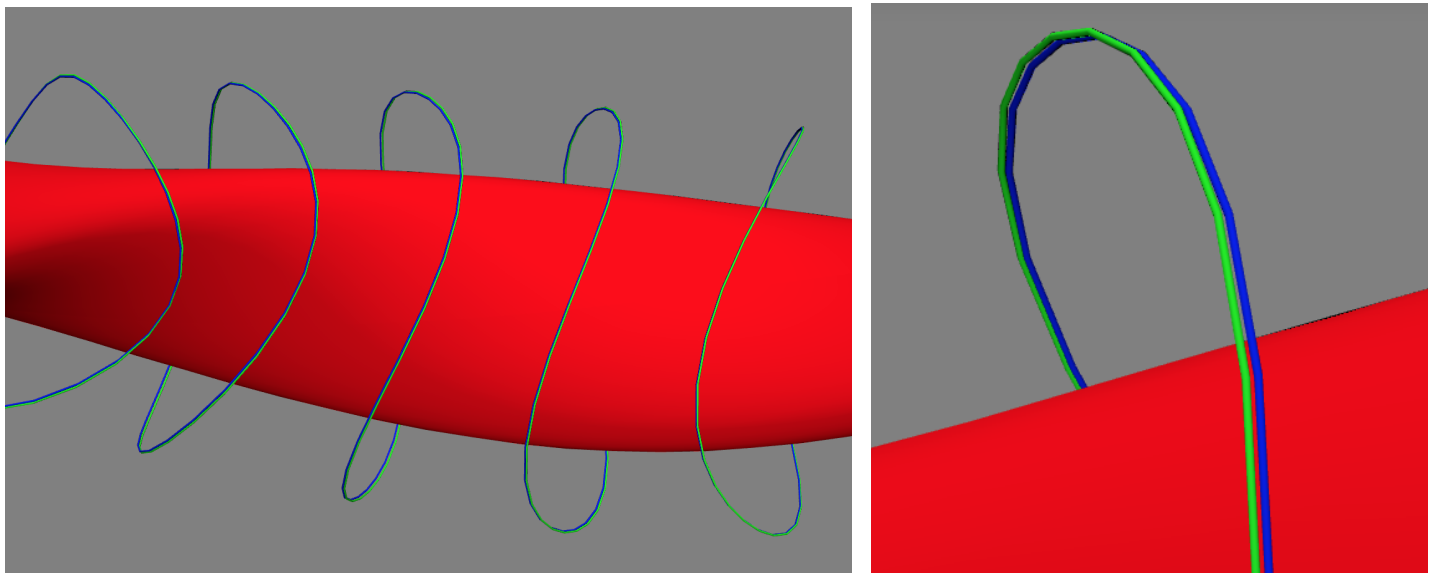
$$f_{phys} \equiv \frac{1}{2} \int_S (\boldsymbol{B} \cdot \boldsymbol{n})^2 dA$$

$$f_{eng} \equiv \sum_{i=1}^{N} L_i$$

Example #1: Including finite thickness changes final coil positions slightly

Here I found two sets of coils for a 5-period, aspect ratio 10, elliptical cross-section stellarator with minor radius 50cm:

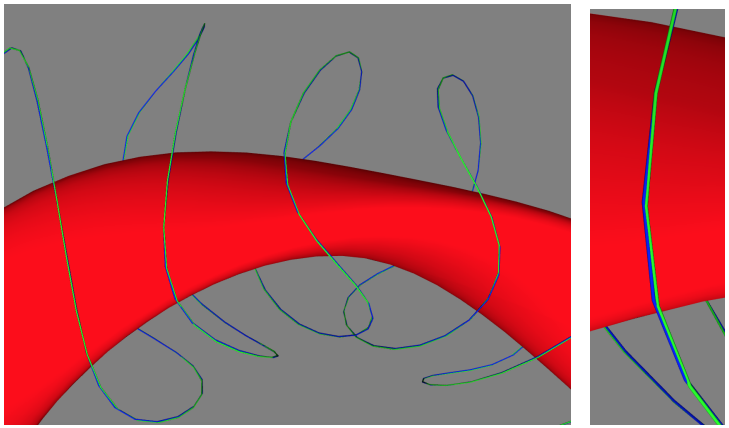
- Filamentary coils (green), like FOCUS
- Finite-build coils (blue, coil centroid is plotted) with coil thickness 15cm, $\delta^2/4L^2 = 0.043$



The radius of the green and blue tubes is 0.5cm, so the optimal coil positions are shifted by no more than 1.5cm for this optimization.

Example #2: Smaller length penalty

- Filamentary coils (green), Finite-build coils (blue, coil centroid is plotted) with coil thickness 15cm
- Coils much further from plasma, $\delta^2/4L^2 = 0.015$



The optimal coil positions are shifted by no more than 0.5cm compared to filamentary coils for this optimization. The finite-build coils decrease the quadratic flux compared to the optimized filamentary coils by 0.5%.

These are initial results

I do not have a W7-X or NCSX comparison yet. Those will be interesting results.

Tolerances: determined by physics objectives

Step 1 of optimization: optimize f_{total} with respect to coil parameters

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

$$f_{total}(\boldsymbol{p}) = f_{phys}(\boldsymbol{p}) + f_{eng}(\boldsymbol{p})$$

Step 2 of optimization: optimize f_{phys} while holding coil shapes fixed.

$$\boldsymbol{r}^*_{COM}, \theta^*, \phi^*, \psi^* = \arg\min_{\boldsymbol{r}_{COM},\theta,\phi,\psi} f_{phys}(\boldsymbol{r}_{COM}, \theta, \phi, \psi)$$

Tolerances: determined by physics objectives

Step 1 of optimization: optimize f_{total} with respect to coil parameters

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

$$f_{total}(\boldsymbol{p}) = f_{phys}(\boldsymbol{p}) + f_{eng}(\boldsymbol{p})$$

Step 2 of optimization: optimize f_{phys} while holding coil shapes fixed.

$$\boldsymbol{r}^*_{COM}, \theta^*, \phi^*, \psi^* = \arg\min_{\boldsymbol{r}_{COM},\theta,\phi,\psi} f_{phys}(\boldsymbol{r}_{COM}, \theta, \phi, \psi)$$

Tolerances: shape gradient and shape Hessian of f_{phys}

$$\delta f_{phys}(\boldsymbol{r}_0, \boldsymbol{\Delta r}) \equiv f_{phys}(\boldsymbol{r}_0 + \boldsymbol{\Delta r}) - f_{phys}(\boldsymbol{r}_0) = \frac{\partial f_{phys}}{\partial \boldsymbol{r}} \boldsymbol{\Delta r} + \frac{1}{2} \boldsymbol{\Delta r}^T \frac{\partial f^2_{phys}}{\partial \boldsymbol{r} \partial \boldsymbol{r}} \boldsymbol{\Delta r} + \ldots$$

Coil tolerance $\sim 1/\delta f_{phys}$

Sensitivity analysis: shape gradients

What is a shape gradient?

Suppose I have some scalar function f. The shape gradient $\frac{\partial f}{\partial \boldsymbol{r}}$ tells us how f changes with respect to real-space displacements of the coils.

Matt Landreman & Elizabeth Paul (2018). Computing local sensitivity and tolerances for stellarator physics properties using shape gradients *Nuclear Fusion*.

Antonsen, T., Paul, E., & Landreman, M. (2019). Adjoint approach to calculating shape gradients for three-dimensional magnetic confinement equilibria *Journal of Plasma Physics*.

Sensitivity analysis: shape gradients

What is a shape gradient?

Suppose I have some scalar function f. The shape gradient $\frac{\partial f}{\partial \boldsymbol{r}}$ tells us how f changes with respect to real-space displacements of the coils.

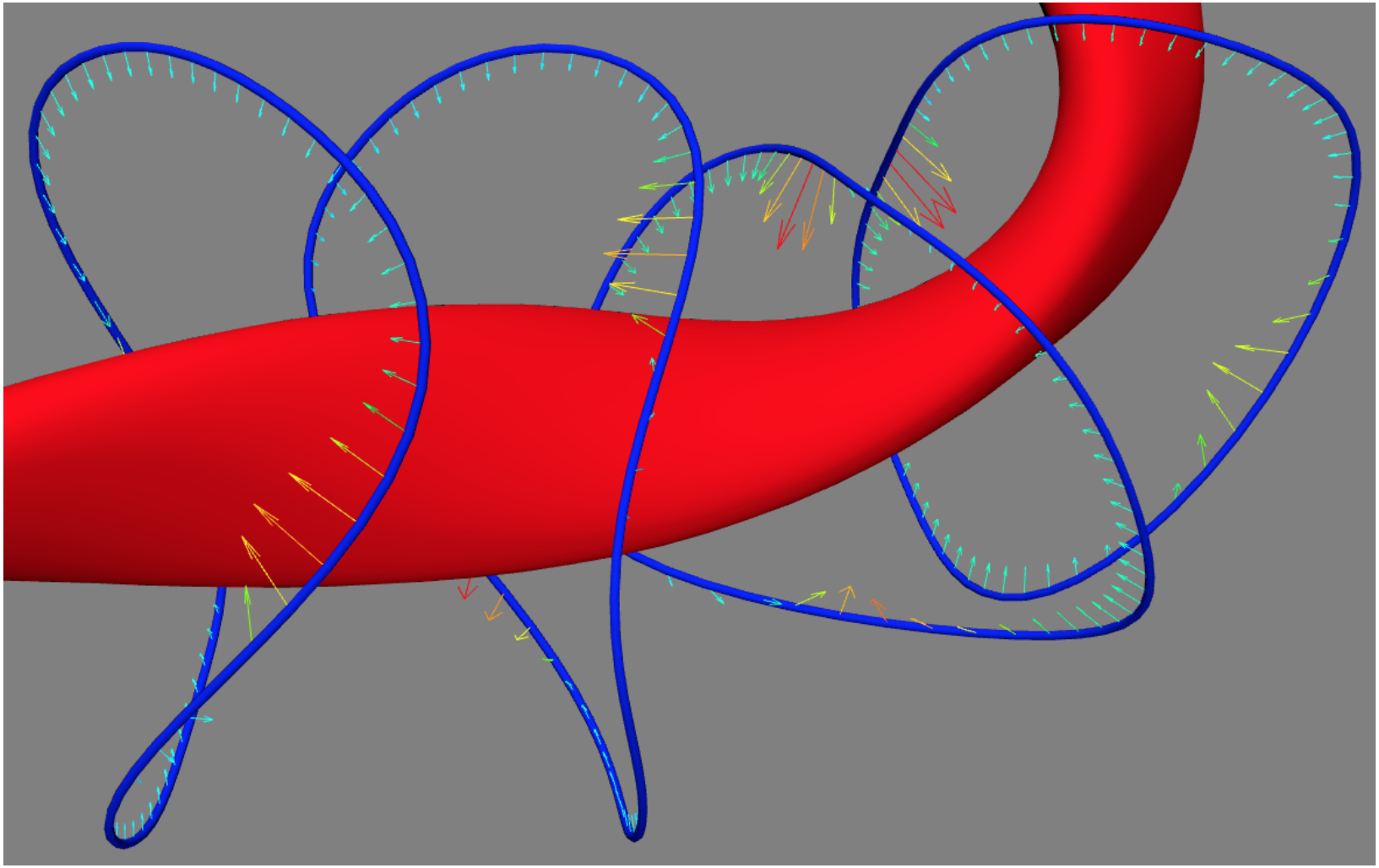
Matt Landreman & Elizabeth Paul (2018). Computing local sensitivity and tolerances for stellarator physics properties using shape gradients *Nuclear Fusion*.

AD makes this easy

Computing the shape gradient and shape Hessian of the coil objectives are each a single additional line of code.

Antonsen, T., Paul, E., & Landreman, M. (2019). Adjoint approach to calculating shape gradients for three-dimensional magnetic confinement equilibria *Journal of Plasma Physics*.

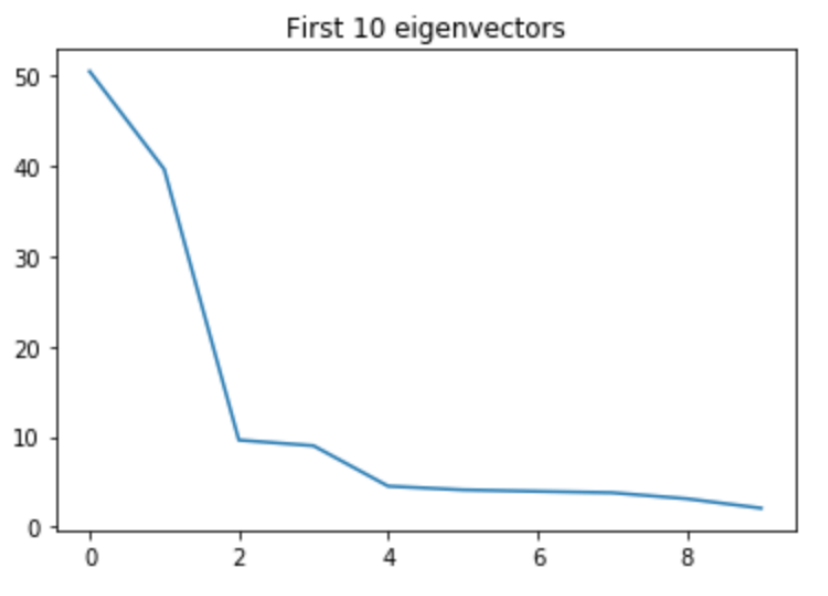
Shape gradient of the coils with respect to quadratic flux



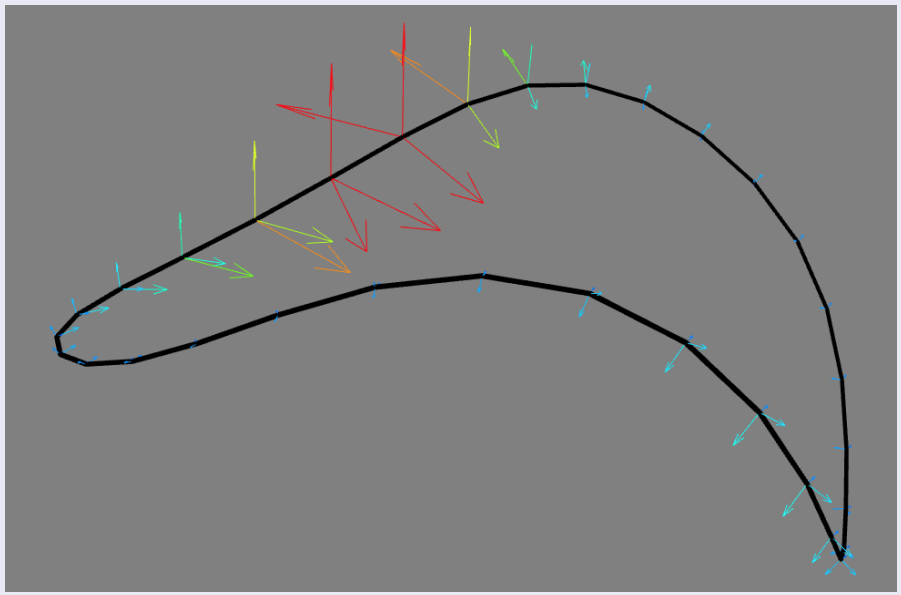
Shape Hessian of the coils with respect to quadratic flux

Eigenvalues and Eigenvectors tell us about coil sensitivities

Eigenspectrum decays quickly



First 3 principal eigenvectors of the shape hessian for a single coil:



New idea: Robust optimization with AD to design higher-tolerance coils

Suppose we add a new term to the objective function which is small if the tolerances are large.

$$f_{total} = f_{phys} + \lambda_{eng} f_{eng} + \lambda_{tol} f_{tol}$$

$$f_{tol} = \sum_{i=1}^{N_{coils}} \mathrm{Tr}(\boldsymbol{H}^i_{phys})$$

where $\boldsymbol{H}^i_{phys}$ is a Hessian matrix of f_{phys} with respect to some properties of the ith coil. This will encourage finding a more robust optimum.

Thank you!

Additional Slides

What does the word 'adjoint' mean?

It's confusing because we use it to mean two different but related things.

Adjoints:

Suppose we have a scalar function f where

$$\boldsymbol{x}_2 = \boldsymbol{f}_1(\boldsymbol{x}_1) \quad \boldsymbol{x}_3 = \boldsymbol{f}_2(\boldsymbol{x}_2) \quad f = f_3(\boldsymbol{x}_3)$$

$$\frac{df}{d\boldsymbol{x}_1} = \frac{d\boldsymbol{x}_2}{d\boldsymbol{x}_1}\frac{d\boldsymbol{x}_3}{d\boldsymbol{x}_2}\frac{df}{d\boldsymbol{x}_3}$$

- An adjoint is a derivative which is computed from right to left rather than left to right.
- Reverse mode AD computes adjoints.

The Adjoint Method:

Suppose we want to minimize a scalar function $f(\boldsymbol{u}(\boldsymbol{p}), \boldsymbol{p})$ where $\boldsymbol{u}$ is the solution to the constraint equation $\boldsymbol{g}(\boldsymbol{u}, \boldsymbol{p}) = 0$. Often $\boldsymbol{g}$ is a discretized PDE, so the adjoint method is related to PDE-constrained optimization.

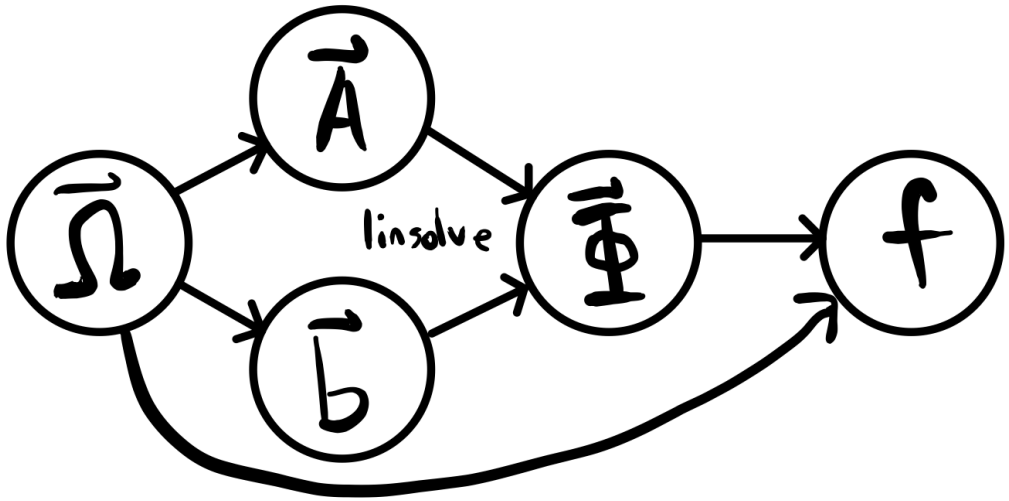
- The adjoint method computes $\frac{df}{d\boldsymbol{p}}$ for this set of equations, using adjoints.

AD performs the adjoint method for linear equations 'effortlessly'

Goal: $\boldsymbol{\Omega}^* = \arg\min_{\boldsymbol{\Omega}} f(\boldsymbol{\Phi}(\boldsymbol{\Omega}), \boldsymbol{\Omega})$

Use GD: $\boldsymbol{\Omega}^{n+1} = \boldsymbol{\Omega}^n - \eta \frac{\partial f}{\partial \boldsymbol{\Omega}}$

s.t. $\boldsymbol{A}(\Omega)\boldsymbol{\Phi} = \boldsymbol{b}(\Omega)$



What about non-linear equations?

It depends on whether your non-linear solver is implemented as a primitive operation by your AD tool. JAX has implemented linear solves but not yet implemented non-linear solvers. dolfin-adjoint is designed to compute the adjoint of a forward model written in the Python interface to FEniCS and Firedrake.

The Adjoint Method: Details

$$\boldsymbol{p}^* = \arg\min_{\boldsymbol{p}} f(\boldsymbol{u}(\boldsymbol{p}), \boldsymbol{p} \text{ s.t. } \boldsymbol{g}(\boldsymbol{u}, \boldsymbol{p}) = 0$$

$$\frac{df}{d\boldsymbol{p}} = \frac{\partial f}{\partial \boldsymbol{p}} + \frac{\partial f}{\partial \boldsymbol{u}} \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{p}}$$

$$\frac{d\boldsymbol{g}}{d\boldsymbol{p}} = 0 = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} \frac{\partial \boldsymbol{u}}{\partial p} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{p}} \Rightarrow \frac{\partial \boldsymbol{u}}{\partial p} = -\left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right)^{-1} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{p}}$$

$$\frac{df}{d\boldsymbol{p}} = \frac{\partial f}{\partial \boldsymbol{p}} - \frac{\partial f}{\partial \boldsymbol{u}} \left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right)^{-1} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{p}}$$

$$\frac{\partial f}{\partial \boldsymbol{u}} \left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right)^{-1} = \boldsymbol{\lambda}^T$$

$$\left(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right)^T \boldsymbol{\lambda} = \frac{\partial f}{\partial \boldsymbol{u}}$$

Structure of FOCUSADD (1 of 6)

Coil centroid is parametrized in free space with a Fourier series, as in FOCUS. Here $\boldsymbol{r}_i(\theta)$ is the position of the i coil centroid.

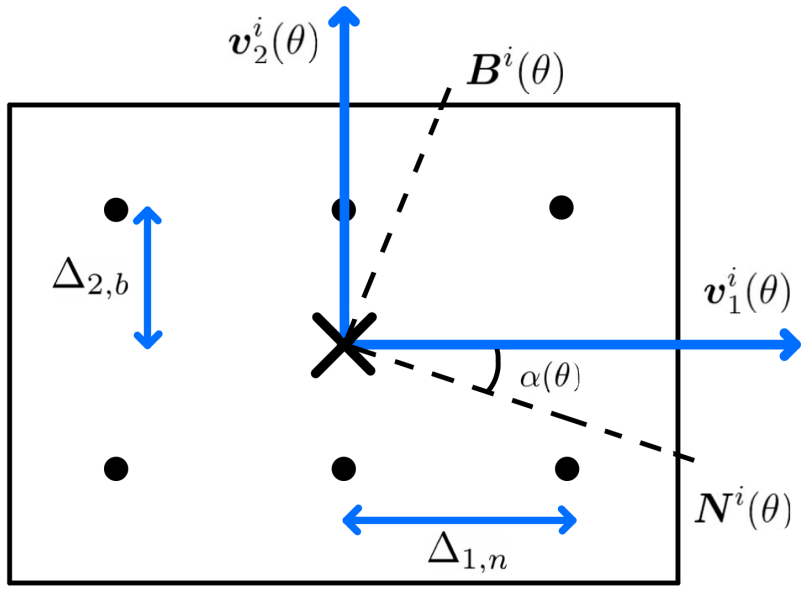
$$x^i(\theta) = \sum_{m=0}^{N_F-1} X^i_{cm}\cos(m\theta) + X^i_{sm}\sin(m\theta)$$

$$y^i(\theta) = \sum_{m=0}^{N_F-1} Y^i_{cm}\cos(m\theta) + Y^i_{sm}\sin(m\theta)$$

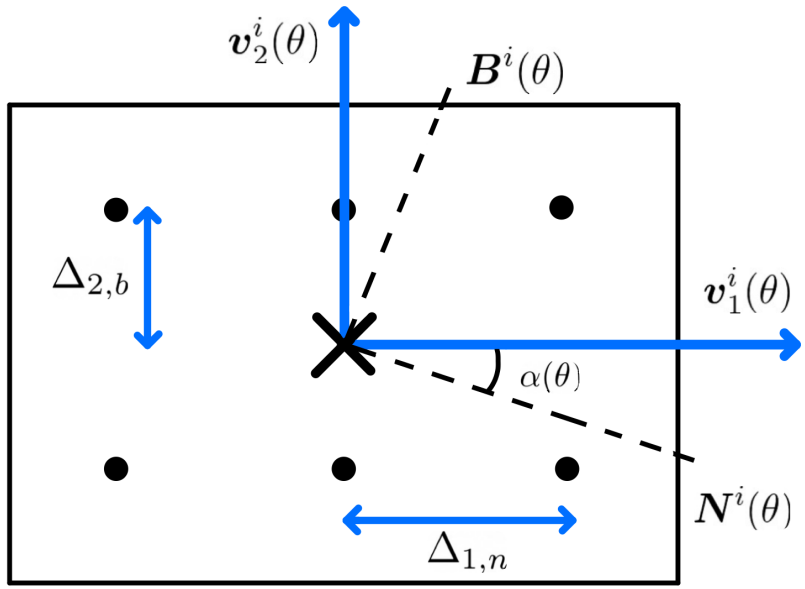
$$z^i(\theta) = \sum_{m=0}^{N_F-1} Z^i_{cm}\cos(m\theta) + Z^i_{sm}\sin(m\theta)$$

Structure of FOCUSADD (2 of 6)

The multi-filament winding pack surrounds the coil centroid. For the ith coil, the axes of the winding pack $\boldsymbol{v}_1^i$ and $\boldsymbol{v}_2^i$ are rotated by an angle α^i relative to the Normal $\boldsymbol{N}^i$ and Binormal $\boldsymbol{B}^i$ vectors of the coil centroid's Frenet-Serret frame.



Structure of FOCUSADD (3 of 6)



α is parametrized by another Fourier series, giving the coil the freedom to twist in space.

$$\alpha^i(\theta) = \frac{N_R\theta}{2} + \sum_{m=0}^{N_{FR}-1} A^i_{cm}\cos(m\theta) + A^i_{sm}\sin(m\theta)$$

Structure of FOCUSADD (4 of 6)

Once we have $\boldsymbol{v}_1^i$ and $\boldsymbol{v}_2^i$, we can compute the position of the $N_n \times N_b$ filaments for each coil. We do this using the following formula for the nth and bth filaments, where n runs from 0 to $N_n - 1$ and b runs from 0 to $N_b - 1$. Here, l_n is the spacing between the filaments in the $\boldsymbol{v}_1$ direction, and l_b is the spacing between the filaments in the $\boldsymbol{v}_2$ direction.

$$\boldsymbol{r}_{n,b}^i(\theta) = \boldsymbol{r}_{central}^i + \left[n - \frac{N_n - 1}{2}\right] l_n \boldsymbol{v}_1^i(\theta) + \left[b - \frac{N_b - 1}{2}\right] l_b \boldsymbol{v}^i(\theta)$$

So far we've only computed vacuum fields, using the Biot-Savart law.

$$\boldsymbol{B}(\boldsymbol{r}) = \sum_{i=1}^{N_c} \sum_{n=1}^{N_1} \sum_{b=1}^{N_2} \mu_0 I_{n,b}^i \oint \frac{d\boldsymbol{l}_{n,b}^i \times (\boldsymbol{r} - \boldsymbol{r}_{n,b}^i)}{|\boldsymbol{r} - \boldsymbol{r}_{n,b}^i|^3}$$

Structure of FOCUSADD (5 of 6)

The optimization is performed using gradient descent on an objective function f_{total}, given by a sum of physics objectives and engineering objectives.

$$f_{total}(\boldsymbol{p}) = f_{Phys}(\boldsymbol{p}) + \lambda_{Eng} f_{Eng}(\boldsymbol{p})$$

The simplest possible physics objective was chosen, the squared surface-normal magnetic field integrated over the surface.

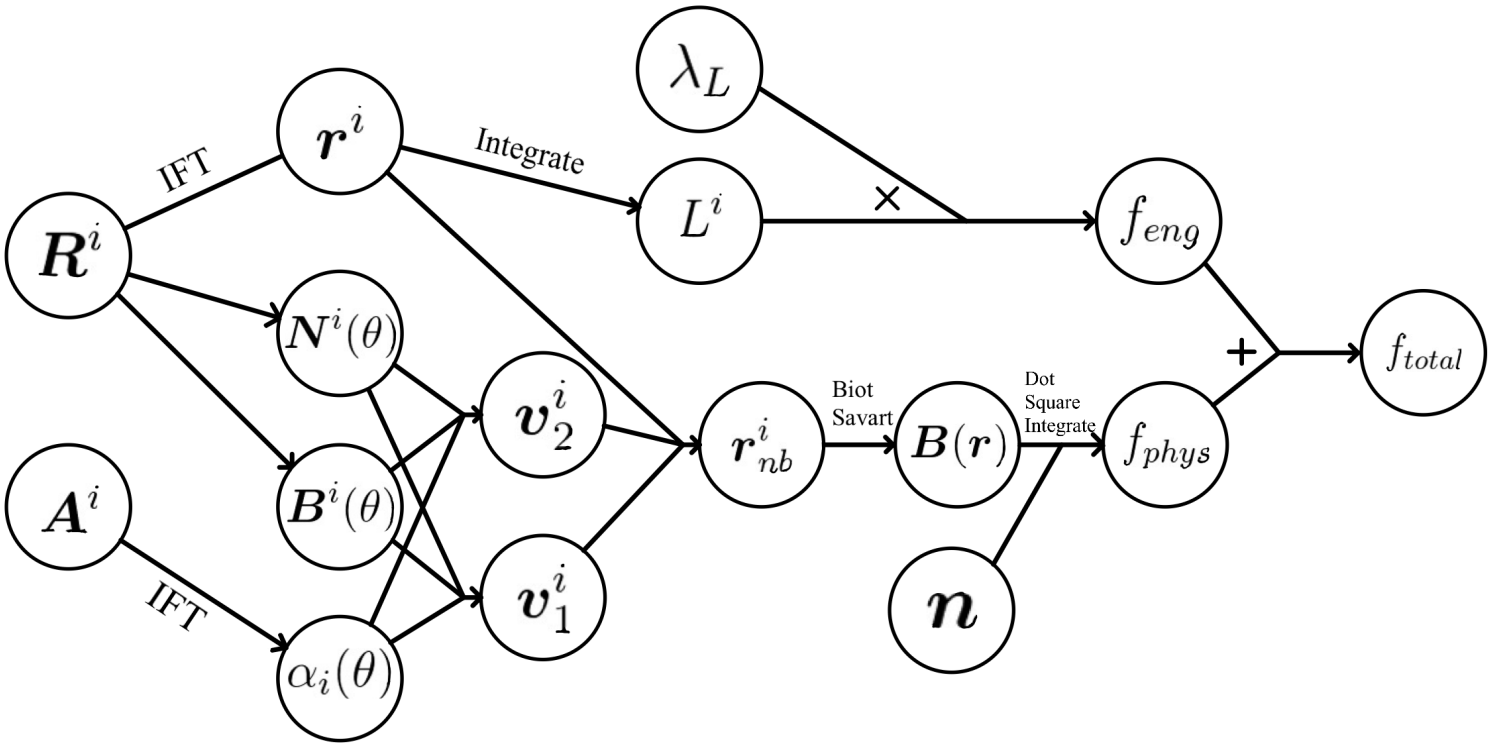
$$f_{Phys} \equiv \int_S (\boldsymbol{B} \cdot \boldsymbol{n})^2 dA$$

A simple engineering objective was chosen, namely the total length of the coils.

$$f_{Eng} \equiv \sum_{i=1}^{N} L_i$$

Structure of FOCUSADD (6 of 6)

The following computational graph describes the structure of the computation performed by FOCUSADD. In my AD tool (JAX), I simply compute f_{total}, then type "grad" to get the gradient.



Using different values of the length penalty, I get different values of $\delta^2/4L^2$

Here I found both filamentary and finite-build coils for a 5-period, aspect ratio 10, elliptical cross-section stellarator with minor radius 50cm, for two examples with different length penalties. I encouraged the coils in example #1 to have a similar $\delta^2/4L^2$ to NCSX and W7-X. In example #2, I allowed the coils to be further from the plasma.

Stellarator	Minor Radius	L	δ	$\delta^2/4L^2$
W7-X	50cm	37cm	20cm	0.073
NCSX	35cm	20cm	12cm	0.09
FOCUSADD Example #1	50cm	36cm	15cm	0.043
FOCUSADD Example #2	50cm	60cm	15cm	0.015

Elliptical cross-section has no concave sections

Therefore, the f_{phys} is lower for coils which are further from the plasma, while f_{Eng} encourages coils to be closer to the plasma. The minimum of the objective finds a balance between these two objectives.

Does a shift of 1cm matter?

Example # 1 doesn't answer that question.

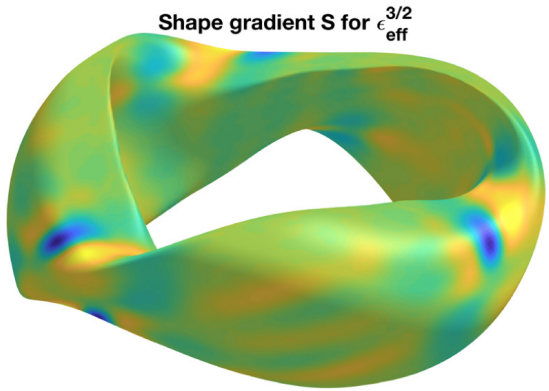
- The true quadratic flux of the optimized filamentary coils is 4% *lower* than the quadratic flux of the optimized finite-build coils.
- One might think that the true quadratic flux of the optimized filamentary coils should be be *higher*, not *lower*, because it ignores the finite-build correction.
- In fact, because the penalty on the length is very large in example #1, and because the finite-build correction to the magnetic field is *positive*, the finite-build coils can be slightly closer to the plasma than the filamentary coils, decreasing f_{Eng} by a large amount but increasing f_{phys} slightly.
- This explanation relies on the fact that the quadratic flux for elliptical cross-section stellarators is lower when the coils are further from the plasma.

Sensitivity analysis: shape gradients

What is a shape gradient?

Suppose I have some scalar function f. The shape gradient tells us how f changes with respect to real-space displacements of the plasma surface or the coils.

This technique was introduced to the stellarator community by Matt Landreman, Elizabeth Paul, and Thomas Antonsen and applied to figures of merit from MHD with respect to the plasma and with respect to the coils. Here, I use AD to calculate shape gradients and shape Hessians of the coils with respect to the coil objectives.



Matt Landreman & Elizabeth Paul (2018). Computing local sensitivity and tolerances for stellarator physics properties using shape gradients *Nuclear Fusion*.

Antonsen, T., Paul, E., & Landreman, M. (2019). Adjoint approach to calculating shape gradients for three-dimensional magnetic confinement equilibria *Journal of Plasma Physics*.