Optimization in stellarators

"Computational optimization has revolutionized stellarator design."
[Gates et al, 2017]

What to optimize?

- Coil designs (confinement properties, complexity, cost, stability, robustness, etc)
- Diverter designs
- Turbulent transport optimization
- ...
Optimization in stellarators

"Computational optimization has revolutionized stellarator design.”

[Gates et al, 2017]

What to optimize?

▶ Coil designs (confinement properties, complexity, cost, stability, robustness, etc)
▶ Diverter designs
▶ Turbulent transport optimization
▶ ...

Optimization Challenges:

▶ Involves complex/expensive physics equations
▶ Parameterization of shapes
▶ Multi-objective minimization
▶ Various constraints (engineering constraints, )
▶ Access to derivatives
▶ Local minimia, high-dimensional parameters
▶ ...

...
Optimization problems


Different types of optimization problems:

$$\min_{x \in \mathbb{R}^n} f(x)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Often, one additionally encounters constraints of the form

$$g(x) = 0 \quad \text{(equality constraints)}$$
$$h(x) \geq 0 \quad \text{(inequality constraints)}$$

- Often used: “programming” $\equiv$ optimization
- continuous optimization ($x \in \mathbb{R}^n$) versus discrete optimization (e.g., $x \in \mathbb{Z}^n$)
- nonsmooth (e.g., $f$ is not differentiable) versus smooth optimization (we assume $f \in C^2$)
- convex vs. nonconvex optimization (convexity of $f$)
Outline

Single-objective deterministic optimization

Gradient-based descent algorithms

Computing derivatives

Stochastic optimization

Remarks and discussion
Continuous unconstrained optimization

Assume \( f(\cdot) \in C^2 \), and focus on unconstrained minimization:

\[
\min_{x \in \mathbb{R}^n} f(x).
\]

A point \( x^* \) is a **global solution** if

\[
f(x) \geq f(x^*) \quad \text{for all } x \in \mathbb{R}^n,
\]

(1)

and a **local solution** if (1) for all \( x \) in a neighborhood of \( x^* \).
Continuous unconstrained optimization

Assume $f(\cdot) \in C^2$, and focus on unconstrained minimization:

$$\min_{x \in \mathbb{R}^n} f(x).$$

A point $x^*$ is a **global solution** if

$$f(x) \geq f(x^*) \quad \text{for all } x \in \mathbb{R}^n,$$

and a **local solution** if (1) for all $x$ in a neighborhood of $x^*$.

At a local minimum $x^*$ holds the **first-order necessary condition**

$$\mathbb{R}^n \ni \nabla f(x^*) = 0$$

and the **second-order (necessary) sufficient condition**

$$\mathbb{R}^{n \times n} \ni \nabla^2 f(x^*) \text{ is positive (semi-) definite.}$$
Convex minimization

A function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is convex if for all \( x, y \) holds, for all \( t \in [0, 1] \):

\[
f(t x + (1 - t) y) \leq tf(x) + (1 - t)f(y)
\]

**Theorem:** If \( f \) is convex, then any local minimizer \( x^* \) is also a global minimizer. If \( f \) is differentiable, then any stationary point \( x^* \) is a global minimizer.
Gradient-based versus derivative-free optimization

- **Derivative-free optimization (DFO)** solves optimization problem without access to derivatives. Useful for problems with small/moderate dimension and if it is not possible to compute derivatives. See for instance book by Conn/Scheinberg/Vicente (SIAM, 2009) [https://doi.org/10.1137/1.9780898718768](https://doi.org/10.1137/1.9780898718768) or recent article [https://arxiv.org/abs/1904.11585](https://arxiv.org/abs/1904.11585)

- **Gradient-based descent methods** (the focus here): Derivatives or approximations thereof can be computed.

- **Surrogate-based optimization** builds cheap to evaluate surrogates, e.g., Gaussian processes; gradients useful but not crucially needed.
Single-objective deterministic optimization

Gradient-based descent algorithms

Computing derivatives

Stochastic optimization

Remarks and discussion
Basic descent algorithm:

1. **Initialization**: Choose starting point $x^0$, set $k = 1$.

2. For $k = 0, 1, 2, \ldots$, find a **descent direction** $d^k$.

3. Find a **step length** $\alpha_k > 0$ for the update

   $x^{k+1} := x^k + \alpha_k d^k$

   such that descent is obtained, i.e., $f(x^{k+1}) < f(x^k)$. Set $k := k + 1$ and repeat.
Descent algorithm

Directions, in which the function decreases (locally) are called descent directions.

- **Steepest descent direction:**
  \[
  d^k = -\nabla f(x^k)
  \]

- When \( H_k = H(x^k) = \nabla^2 f(x^k) \) is positive definite, then
  \[
  d^k = -H_k^{-1}\nabla f(x^k)
  \]
  is the **Newton descent** direction. At a local minimum, \( H(x^*) \) is positive (semi)definite.
Idea behind Newton’s method in optimization: Instead of finding minimum of $f$, find minimum of quadratic approximation of $f$ around current point:

$$q_k(d) = f(x^k) + \nabla f(x^k)^T d + \frac{1}{2}d^T \nabla^2 f(x^k) d$$

Minimum is (provided $\nabla^2 f(x^k)$ is spd):

$$d = -\nabla^2 f(x^k)^{-1} \nabla f(x^k).$$

is the Newton search direction. Since this is the minimum of the quadratic approximation, $\alpha_k = 1$ is the “optimal” step length.
Descent algorithm

Directions, in which the function decreases (locally) are called descent directions.

- When $B_k \in \mathbb{R}^{n \times n}$ is positive definite, then
  \[ d^k = -B_k^{-1}\nabla f(x^k) \]
  is the quasi-Newton descent direction.

BFGS methods accumulate gradient information to approximate the Hessian.
Descent algorithm

Directions, in which the function decreases (locally) are called descent directions.

- **Levenberg-Marquardt** method for nonlinear least-squares, where the Hessian is approximately $H_k \approx J_k^T J_k$:

\[
(J_k^T J_k + \lambda I) d^k = -\nabla f(x^k)
\]

$\lambda > 0$ interpolates between steepest descent and Gauss-Newton direction.
Descent algorithm

**Step length:** Need to choose step length $\alpha_k > 0$ in

$$x^{k+1} := x^k + \alpha_k d^k$$

Ideally: Find minimum $\alpha$ of 1-dim. problem

$$\min_{\alpha > 0} f(x^k + \alpha d^k).$$

It is not necessary to find the exact minimum. Usually one finds $\alpha_k$ that satisfies the Armijo condition:

$$f(x^k + \alpha_k d^k) \leq f(x^k) + c_1 \alpha_k \nabla f(x^k)^T d^k,$$  \hspace{1cm} (2)

where $c_1 \in (0, 1)$ (e.g., $c_1 = 10^{-4}$).
Descent algorithm

Find $\alpha_k$ that satisfies the Armijo condition:

$$f(x^k + \alpha_k d^k) \leq f(x^k) + c_1 \alpha_k \nabla f(x^k)^T d^k,$$  \hspace{1cm} (3)

where $c_1 \in (0, 1)$.

Use backtracking linesearch to find a step length that is large enough:

- Start with (large) step length $\alpha_k^0 > 0$.
- If it satisfies (3), accept the step length.
- Else, compute $\alpha_k^{i+1} := \rho \alpha_k^i$ with $\rho < 1$ (usually, $\rho = 0.5$) and go back to previous step.

This also leads to a globally converging method to a stationary point.
Descent algorithm
Convergence to stationary points

Denote the angle between $d^k$ and $-\nabla f(x^k)$ by $\Theta_k$:

$$
\cos(\Theta_k) = \frac{-\nabla f(x^k)^T d^k}{\|\nabla f(x^k)\| \|d^k\|}.
$$

Assumptions on $f : \mathbb{R}^n \to \mathbb{R}$: continuously differentiable, derivative is Lipschitz-continuous, $f$ is bounded from below.

Then:

$$
\sum_{k \geq 0} \cos^2(\Theta_k) \|\nabla f(x^k)\|^2 < \infty.
$$

In particular: If $\cos(\Theta_k) \geq \delta > 0$, then $\lim_{k \to \infty} \|\nabla f(x^k)\| = 0$.

Note that this does not imply that $x^k$ converges.
Let us consider a simple case, where $f$ is quadratic:

$$f(x) := \frac{1}{2} x^T Q x - b^T x,$$

where $Q$ is spd. The gradient is $\nabla f(x) = Qx - b$, and minimizer $x^*$ is solution to $Qx = b$. Using exact line search, the convergence is:

$$\|x^{k+1} - x^*\|_Q^2 \leq \frac{\lambda_{\text{max}} - \lambda_{\text{min}}}{\lambda_{\text{max}} + \lambda_{\text{min}}} \|x^k - x^*\|_Q^2$$

(linear convergence with rate depending on eigenvalues of $Q$)
Descent algorithms

Convergence of steepest descent
Descent algorithms

Convergence of steepest descent
Newton’s method: Assumptions on \( f \): 2\( \times \)differentiable with Lipschitz-continuous Hessian \( \nabla^2 f(x^k) \). Hessian is positive definite in a neighborhood around solution \( x^* \).

Assumptions on starting point: \( x^0 \) sufficient close to \( x^* \).

Then: Quadratic convergence of Newton’s method with \( \alpha_k = 1 \), and \( \| \nabla f(x^k) \| \rightarrow 0 \) quadratically.
Outline

Single-objective deterministic optimization

Gradient-based descent algorithms

Computing derivatives

Stochastic optimization

Remarks and discussion
Computing derivatives

All methods require gradients (and some of them, Hessians).

Computation of derivatives:

▶ **Paper-and-pencil** derivatives for simple functions
▶ **Finite-differences** with small $h > 0$:

$$
\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x)}{h}
$$

requires $n$ (i.e., the number of params) evaluations of $f$.

▶ **Algorithmic/Automatic Differentiation**
▶ **Analytical differentiation using adjoints** for complicated objective functions, i.e., coil shape parameters and $f(x)$ is an objective involving $\iota$ at the magnetic axis (sometimes called PDE-constrained optimization)
Computing derivatives: algorithmic differentiation

Source code to source code method based on chain rule with elementary operators, e.g., consider $f(x_1, x_2) = x_1 x_2 + \sin(x_1)$:

\[
\begin{align*}
\dot{w}_1 &= 1 \quad \text{(seed)} \\
\dot{w}_2 &= 0 \quad \text{(seed)} \\
\dot{w}_3 &= w_2 \cdot \dot{w}_1 + w_1 \cdot \dot{w}_2 \\
\dot{w}_4 &= \cos w_1 \cdot \dot{w}_1 \\
\dot{w}_5 &= \dot{w}_3 + \dot{w}_4
\end{align*}
\]

Operations to compute value

| $w_1 = x_1$ | $w_2 = x_2$ | $w_3 = w_1 \cdot w_2$ | $w_4 = \sin w_1$ | $w_5 = w_3 + w_4$ |

Computational forward graph [Source: English Wikipedia].
Computing derivatives: algorithmic differentiation

Source code to source code method based on chain rule with elementary operators, e.g., consider \( f(x_1, x_2) = x_1 x_2 + \sin(x_1) \):

\[
\begin{align*}
\bar{w}_1 &= x_1 \\
\bar{w}_2 &= x_2 \\
\bar{w}_3 &= \bar{w}_1 \cdot \bar{w}_2 \\
\bar{w}_4 &= \sin \bar{w}_1 \\
\bar{w}_5 &= \bar{w}_3 + \bar{w}_4
\end{align*}
\]

\[
\begin{align*}
\dot{\bar{w}}_1 &= 1 \\
\dot{\bar{w}}_2 &= 0 \\
\dot{\bar{w}}_3 &= \bar{w}_2 \cdot \dot{\bar{w}}_1 + \bar{w}_1 \cdot \dot{\bar{w}}_2 \\
\dot{\bar{w}}_4 &= \cos \bar{w}_1 \cdot \dot{\bar{w}}_1 \\
\dot{\bar{w}}_5 &= \dot{\bar{w}}_3 + \dot{\bar{w}}_4
\end{align*}
\]
Computing derivatives: algorithmic differentiation

- Several tools for algorithmic differentiation exist for C/C++ and FORTRAN (and other) codes.
- Sometimes requires limited human interaction
- Has been applied to very complex codes, e.g., to MITgcm
- Used for neural network training (“backpropagation”)

See for instance:
https://en.wikipedia.org/wiki/Automatic_differentiation
Computing derivatives: optimization problem

Optimization formulations ($g$ is e.g., BVP determining the magnetic axis):

**unconstrained:**

$$\min_p f(x_a(p), p)$$

where $x_a(p)$ solves $g(x_a(p), p) = 0$.

**constrained:**

$$\min_{p,x_a} f(x_a, p)$$

subject to $g(x_a, p) = 0$.

- Formulations are equivalent (provided $g$ has unique solution for given $p$)
- If $g$ is a PDE, this is often called PDE-constrained optimization
Evaluating the gradient using finite differences, i.e.,

\[
\frac{\partial f}{\partial p_k} = \frac{f(p_k + \epsilon) - f(p_k)}{\epsilon} + O(\epsilon) \quad k = 1 \ldots N_p
\]

requires \(N_p + 1\) objective function evaluations.

<table>
<thead>
<tr>
<th>(N_p)</th>
<th>102</th>
<th>192</th>
<th>282</th>
<th>372</th>
<th>462</th>
<th>552</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>84</td>
<td>222</td>
<td>411</td>
<td>664</td>
<td>1057</td>
<td>1473</td>
</tr>
</tbody>
</table>

**Table:** Objective function evaluation timings in terms of \(N_p\) for the NCSX coils on a standard laptop.

- \(~25\) minutes to evaluate the gradient when \(N_p = 552\).
Compute the gradient using adjoint methods

\[
\begin{align*}
\min_{p, x_a} f(x_a, p) \\
\text{subject to } g(x_a, p) &= 0
\end{align*}
\]  

The gradient of the objective function with respect to the design parameters is

\[
\frac{df}{dp} = \frac{\partial f}{\partial x_a} \frac{\partial x_a}{\partial p} + \frac{\partial f}{\partial p}.
\]

\[
\frac{\partial x_a}{\partial p}
\]

is expensive to compute: \(N_p\) solves of a linear system.

Idea: eliminate \(\frac{\partial x_a}{\partial p}\) using the equality constraint.
Compute the gradient by solving an adjoint problem

\[
\min_p f(x_a, p) \tag{6}
\]
subject to \( g(x_a, p) = 0 \),

The gradient of the objective function with respect to the design parameters is

\[
\frac{df}{dp} = \frac{\partial f}{\partial x_a} \frac{\partial x_a}{\partial p} + \frac{\partial f}{\partial p}. \tag{7}
\]

The derivative of the discretized BVP describing the magnetic axis is

\[
\frac{\partial g}{\partial x_a} \frac{\partial x_a}{\partial p} + \frac{\partial g}{\partial p} = 0, \quad \text{which gives}, \quad \frac{\partial x_a}{\partial p} = - \left[ \frac{\partial g}{\partial x_a} \right]^{-1} \frac{\partial g}{\partial p}, \tag{8}
\]

and thus

\[
\frac{df}{dp} = - \underbrace{\frac{\partial f}{\partial x_a} \left[ \frac{\partial g}{\partial x_a} \right]^{-1} \frac{\partial g}{\partial p}}_{\lambda^T} + \frac{\partial f}{\partial p}. \tag{9}
\]
Compute the gradient by solving an adjoint problem

The gradient of the objective function becomes

$$\frac{df}{dp} = -\lambda^T \frac{\partial g}{\partial p} + \frac{\partial f}{\partial p}. \quad (10)$$

where

$$\frac{\partial f}{\partial x_a} = \lambda^T \frac{\partial g}{\partial x_a}, \quad (11)$$

and $\lambda$ is the adjoint variable.

- Determining $\lambda$ requires one matrix solve, independent of the number of design parameters $N_p$.
- Constructing the adjoint system is extremely parallelizable.
- We have explicit formulas for $\frac{\partial g}{\partial p}$, $\frac{\partial g}{\partial x_a}$, and $\frac{\partial f}{\partial p}$, $\frac{\partial f}{\partial x_a}$.
Timings

<table>
<thead>
<tr>
<th>$N_p$</th>
<th>102</th>
<th>192</th>
<th>282</th>
<th>372</th>
<th>462</th>
<th>552</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite differences</td>
<td>84</td>
<td>222</td>
<td>411</td>
<td>664</td>
<td>1057</td>
<td>1473</td>
</tr>
<tr>
<td>Adjoint approach</td>
<td>4</td>
<td>11</td>
<td>26</td>
<td>48</td>
<td>83</td>
<td>116</td>
</tr>
</tbody>
</table>

Table: Timing comparison of finite difference and adjoint gradient evaluation in terms of $N_p$ for the NCSX coils on a standard laptop.

- Adjoint equation requires **one** solve independent of $N_p$. Growth in time only due to increase in discretization accuracy.
- Over an order of magnitude speed-up using the adjoint method over finite differences.
- Majority of time spent constructing the adjoint problem, rather than solving it.
Convergence test

Compare the directional derivative of the objective function in a random direction, \( u \), computed with finite differences,

\[
D_{u}f = \frac{f(p + \epsilon u) - f(p - \epsilon u)}{2\epsilon} + O(\epsilon^2),
\]

with the directional derivative given by the gradient computed with the adjoint approach.

Figure: Convergence of finite difference directional derivative to the one obtained with the adjoint method (black). The blue slope indicates second order convergence.
Benchmark - minimizing ripple in the NCSX coils

- Target physics parameters in $f$ are chosen such that the only nonzero term in the objective function is the ripple.
- $N_F = 4$, i.e., $N_p = 84$.
- Steepest descent optimization algorithm along using the gradient obtained with the adjoint approach.

The ripple on the magnetic axis is successfully reduced without affecting the other monitored physical characteristics of the stellarator.
Comments on derivatives

- Derivatives are crucial for optimization (derivative-free optimization usually limited to small/moderate dimensions)
- Derivatives not only for optimization—also for sensitivity studies (e.g., M. Landreman and S. Hudson’s talk) and uncertainty quantification, stochastic optimization
- When you write a code/design a method, avoid choices that introduce non-smoothness and/or prevent code from being used with automatic differentiation
- When/where does the dimension really matter? It depends on what methods are being used.
Outline

- Single-objective deterministic optimization
- Gradient-based descent algorithms
- Computing derivatives
- **Stochastic optimization**
- Remarks and discussion
Stochastic Optimization

\[
\min_{\mathbf{x}} \mathbb{E}_{\xi} \{ J(\mathbf{x}, \xi) \} + R(\mathbf{x}, \xi)
\]

- One optimization variable for all \( \mathbf{x} \)
- May include risk measure \( R(\cdot) \) (e.g., variance or CVaR)
- Challenges: Integration over high-dimensional \( \xi \)

Sources of uncertainty in \( \xi \):
- Manufacturing uncertainty
- Model uncertainty (i.e., difference between model and real world)
- Experimental uncertainty

Stochastic optimization aims at solutions that are optimal on average (possibly controlling the risk of bad solutions); we assume we know some distribution for \( \xi \).
Stochastic vs. robust optimization

- Robust optimization usually uses a min-max formulation, i.e., minimize the worst case design (usually requires a closed+bounded set of parameters).
- Various different formulations of robust optimization
- Some robust formulations can be considered stochastic optimization problems
Approximations of stochastic optimization problems

Stochastic optimization problem:

$$\min_x E_{\xi} \left\{ J(x, \xi) \right\} + R(x, \xi)$$  \hspace{1cm} (12)

Main two approaches:

▶ **Sample Average Approximation (SAA):** Use Monte Carlo to approximation expectation, then solve a deterministic problem, i.e., (neglecting risk-averse term for now):

$$\min_x \frac{1}{N} \sum_{n=1}^{N} \left\{ J(x, \xi_n) \right\}$$  \hspace{1cm} (13)

▶ **Stochastic Approximation (SA):** Approximate original problem (12) using e.g., partial gradients (leads to stochastic gradient methods–SGD as used in machine learning)
SAA vs. SA

SAA

+ Deterministic optimization methods apply: descent method + linesearch
  - Solves approximate problem
  - Requires many evaluations of $f$ and $f'$ for large $N$
SAA vs. SA

SA

- Algorithm usually has a probabilistic component
- Often finds points only close to (local) minimizer
- Tuning parameters required
- Potential to find better minima (e.g., with larger basin, i.e., more robust)
- Requires many evaluations of $f$ and $f'$ for large $N$
- $f$ not evaluated exactly, thus one cannot check descent (no linesearch!)
Outline

Single-objective deterministic optimization

Gradient-based descent algorithms

Computing derivatives

Stochastic optimization

Remarks and discussion
Finding global minima

Global optimization is difficult. Some heuristics:

- Different initializations in local optimization algorithms.
- Multi-resolution ideas, i.e., start with a low-dimensional space (e.g., few Fourier modes) and increase dimension as you go along.
- Introduce randomness in your algorithms (stochastic gradients). This may help to get out of local minima.
Dimensions of optimization space

- Gradient-based methods do not necessarily suffer from high dimensions
- Sometimes problems can become simpler (and more convex) if considered in higher dimensions (e.g., over-parameterization in neural network training)
Multi-objective optimization

- We usually scalarize the objective (i.e., form a weighted sum of individual objectives).
- Thus the solution depends on how individual objectives are weighted.
- Related but not identical to Pareto optimality (i.e., solutions where it is not possible to improve one objective without making another one worse).
- Might be able to find better solutions if one considers this as multi-objective optimization.
Take-home messages

- Requirements and guarantees for gradient-based descent optimization methods
- Computation of gradients (and role of dimension of optimization space)
- Stochastic gradient methods