Optimization II: Unconstrained Multivariable


Doug James (and Justin Solomon)
Announcements

- Today’s class:
  - Unconstrained optimization:
    - Newton’s method (uses Hessians)
    - BFGS method (no Hessians)
  - Automatic differentiation
- Homework 5 (due Thursday); polynomial differentiation
- Midterm (update)
Unconstrained Multivariable Problems

minimize

\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]
Recall

\[ \nabla f(\vec{x}) \]

“Direction of steepest ascent”
Recall

\[-\nabla f(\vec{x})\]

“Direction of steepest descent”
Observation

If $\nabla f(\vec{x}) \neq \vec{0}$, for sufficiently small $\alpha > 0$,

$$f(\vec{x} - \alpha \nabla f(\vec{x})) \leq f(\vec{x})$$
Gradient Descent Algorithm

Iterate until convergence:

1. \( g_k(t) \equiv f(\vec{x}_k - t \nabla f(\vec{x}_k)) \)

2. Find \( t^* \geq 0 \) minimizing (or decreasing) \( g_k \)

3. \( \vec{x}_{k+1} \equiv \vec{x}_k - t^* \nabla f(\vec{x}_k) \)
Stopping Condition

\[ \nabla f(\vec{x}_k) \approx \vec{0} \]

Don’t forget: Check optimality!
Line Search

\[ g_k(t) \equiv f(\vec{x}_k - t \nabla f(\vec{x}_k)) \]

- One-dimensional optimization
- Don’t have to minimize completely: Wolfe conditions
- Constant \( t \): “Learning rate”
Line Search

\[ g_k(t) \equiv f(\vec{x}_k - t \nabla f(\vec{x}_k)) \]

- One-dimensional optimization
- Don’t have to minimize completely: Wolfe conditions
- Constant \( t \): “Learning rate”

Worth reading about:
**Nesterov’s Accelerated Gradient Descent**
function GRADIENT-DESCENT(f(\vec{x}), \vec{x}_0)

\ vec{x} \leftarrow \vec{x}_0

for \( k \leftarrow 1, 2, 3, \ldots \)

\textbf{Define-Function}(g(t) \equiv f(\vec{x} - t\nabla f(\vec{x})))

\( t^* \leftarrow \textbf{Line-Search}(g(t), t \geq 0) \)

\vec{x} \leftarrow \vec{x} - t^*\nabla f(\vec{x}) \quad \triangleright \text{Update estimate of minimum}

\textbf{if} \|\nabla f(\vec{x})\|_2 < \varepsilon \textbf{ then}

\textbf{return} \ x^* = \vec{x}
Newton’s Method (again!)

\[
f(\vec{x}) \approx f(\vec{x}_k) + \nabla f(\vec{x}_k)^\top (\vec{x} - \vec{x}_k) \\
+ \frac{1}{2} (\vec{x} - \vec{x}_k)^\top H_f(\vec{x}_k)(\vec{x} - \vec{x}_k)
\]
Newton’s Method (again!)

\[
f(\vec{x}) \approx f(\vec{x}_k) + \nabla f(\vec{x}_k)^\top (\vec{x} - \vec{x}_k) \\
+ \frac{1}{2}(\vec{x} - \vec{x}_k)^\top H_f(\vec{x}_k)(\vec{x} - \vec{x}_k)
\]

\[
\implies \vec{x}_{k+1} = \vec{x}_k - [H_f(\vec{x}_k)]^{-1}\nabla f(\vec{x}_k)
\]
Newton's Method (again!)

\[ f(\vec{x}) \approx f(\vec{x}_k) + \nabla f(\vec{x}_k)^\top (\vec{x} - \vec{x}_k) + \frac{1}{2}(\vec{x} - \vec{x}_k)^\top H_f(\vec{x}_k)(\vec{x} - \vec{x}_k) \]

\[ \Rightarrow \quad \vec{x}_{k+1} = \vec{x}_k - [H_f(\vec{x}_k)]^{-1}\nabla f(\vec{x}_k) \]

Consideration:
What if \( H_f \) is not positive (semi-)definite?
Motivation

- $\nabla f$ might be hard to compute but $H_f$ is harder

- $H_f$ might be dense: $n^2$
Approximate derivatives to avoid expensive calculations

e.g. secant, Broyden, …
Common Optimization Assumption

- $\nabla f$ known
- $H_f$ unknown or hard to compute
Quasi-Newton Optimization

\[ \vec{x}_{k+1} = \vec{x}_k - \alpha_k B_k^{-1} \nabla f(\vec{x}_k) \]

\[ B_k \approx H_f(\vec{x}_k) \]
Warning

<advanced material>

See Nocedal & Wright
Broyden-Style Update

\[
B_{k+1}(\vec{x}_{k+1} - \vec{x}_k) = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)
\]
Additional Considerations

- $B_k$ should be symmetric
- $B_k$ should be positive (semi-)definite
Davidon-Fletcher-Powell (DFP)

\[
\begin{align*}
\min_{B_{k+1}} & \quad \|B_{k+1} - B_k\| \\
\text{s.t.} & \quad B_{k+1}^\top = B_{k+1} \\
& \quad B_{k+1}(\vec{x}_{k+1} - \vec{x}_k) = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)
\end{align*}
\]
Observation

\[ \| B_{k+1} - B_k \| \text{ small does not mean } \| B_{k+1}^{-1} - B_k^{-1} \| \text{ is small} \]
Observation

\[ \| B_{k+1} - B_k \| \quad \text{small does not mean} \quad \| B_{k+1}^{-1} - B_k^{-1} \| \quad \text{is small} \]

Idea: Try to approximate

\[ B_k^{-1} \quad \text{directly} \]
BFGS Update

\[
\begin{align*}
\min_{HB_{k+1}} & \quad \|H_{k+1} - H_k\| \\
\text{s.t.} & \quad H_{k+1}^\top = H_{k+1} \\
& \quad \overrightarrow{x}_{k+1} - \overrightarrow{x}_k = H_{k+1}(\nabla f(\overrightarrow{x}_{k+1}) - \nabla f(\overrightarrow{x}_k))
\end{align*}
\]

State of the art!
The BFGS algorithm for finding a local minimum of differentiable $f(\vec{x})$ without its Hessian. The function COMPUTE-ALPHA finds large $\alpha > 0$ satisfying $\vec{y} \cdot \vec{s} > 0$, where $\vec{y} = \nabla f(\vec{x} + \vec{s}) - \nabla f(\vec{x})$ and $\vec{s} = \alpha \vec{p}$.
Lots of Missing Details

- Choice of $|| \cdot ||$
- Limited-memory alternative
Automatic Differentiation

- Techniques to numerically evaluate the derivative of a function specified by a computer program.


- Different from *finite differences* (approximation) and *symbolic differentiation*.

- In Julia: http://www.juliadiff.org
  - Example: ForwardDiff. (uses dual numbers) https://github.com/JuliaDiff/ForwardDiff.jl