Machine Learning and Photonics

Week 3

Ergun Simsek

University of Maryland Baltimore County
simsek@umbc.edu

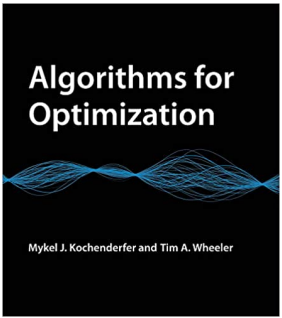
Numerical Optimization

February 14, 2023

Numerical Optimization: An Introduction

Goal: Develop basic understanding of how numerical optimization works

- Local Optimization
  - Derivatives and Gradients
  - First/second-Order optimization
  - Gradient free methods
  - Stochastic methods
- Global Optimization
- Surrogate Optimization



https://algorithmsbook.com/optimization/

Derivative

The goal of optimization is to find the point that minimizes an objective function. Knowing how the value of a function changes (derivative) is useful.

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x$$

$$f'(x) = \frac{\Delta f(x)}{\Delta x}$$

Derivatives in multiple dimensions

Jacobian (partial derivative)
$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \cdots \frac{\partial f(x)}{\partial x_n}\right]$$

Hessian (second derivatives)
$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ & \vdots & \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

Numerical Differentiation

For practical applications, we rely on numerical methods to evaluate the derivatives.

- Finite Difference Methods

$$f'(x) \approx \begin{cases} \frac{f(x+h)-f(x)}{h} & \text{forward} \\ \frac{f(x+h/2)-f(x-h/2)}{h} & \text{central} \\ \frac{f(x)-f(x-h)}{h} & \text{backward} \end{cases}$$

- Complex Step Method

$$f'(x) = \text{imag}(f(x+ih)/h)$$

https://doi.org/10.1145/838250.838251

Comparison: Forward vs. Central vs Complex Step Method

Imagine you have dataset that includes 2 different patterns: one is slowly changing and one changing so rapidly. Your dataset is sampled according to the fast one.

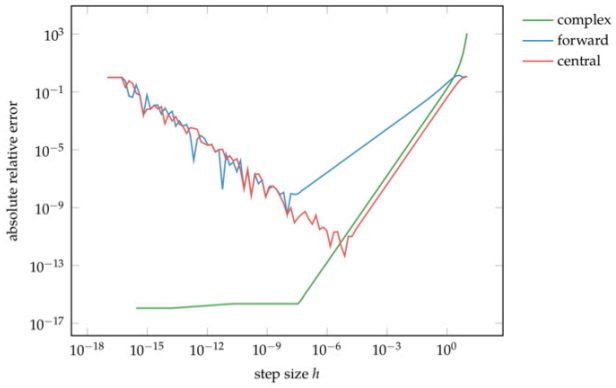


Figure 2.4. A comparison of the error in derivative estimate for the function $\sin(x)$ at $x = 1/2$ as the step size is varied. The linear error of the forward difference method and the quadratic error of the central difference and complex methods can be seen by the constant slopes on the right hand side. The complex step method avoids the subtractive cancellation error that occurs when differencing two function evaluations that are close together.

Complex Step

According to Taylor expansion,

$$f(x+ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2!} - ih^3\frac{f'''(x)}{3!} + \cdots$$

If we take only the imaginary part,

$$\mathrm{Im}(f(x+ih)) = hf'(x) - h^3\frac{f'''(x)}{3!} + \cdots$$

$$f'(x) = \frac{\mathrm{Im}(f(x+ih))}{h} + h^2\frac{f'''(x)}{3!} - \cdots = \frac{\mathrm{Im}(f(x+ih))}{h} + O(h^2)$$

While the real part is

$$\mathrm{Re}(f(x+ih)) = f(x) - h^3\frac{f''(x)}{2!} + \cdots$$

$$f(x) = \mathrm{Re}(f(x+ih)) + O(h^2)$$

The complex step method is advantageous since

- Both $f(x)$ and $f'(x)$ can be evaluated in a single run

Why is complex step better than central difference?

$$f(x+ih) = u(x,h) + iv(x,h)$$
$$\mathrm{Im}(f(x+ih)) = h\frac{\partial v(x,y)}{\partial y}|_{y=0} + O(h^2)$$

If $v(x,0) = 0$, $f(x) = u(x,0)$. Dividing by h, we have f'(x)

$$\frac{\partial v(x,y)}{\partial y}|_{y=0} = \frac{\partial u(x,0)}{\partial x}$$

The left side is what we used to compute the complex step. The right side is due to Cauchy-Riemann equation, which is used by the finite difference. Note that two method use two different functions (u and v).

Why is complex step better than central difference?

Consider a function $f(z) = z^2$,

$$f(z) = z^2 = x^2 - y^2 + i2xy$$

The finite difference does the function x^2, while the complex step gives $2x$, in the case for any $h = y > 0$.

Try another function:

$$\cos(x + iy) = \cos(x)\cosh(y) - i\sin(x)\sinh(y).$$

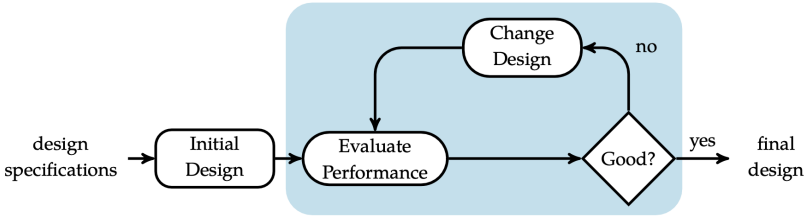
The imaginary part is $-\sin(x)\sinh(y)$. For a small y, it gives $-\sin(x)$.

Where are we going?

- Taylor Series –> Numerical Derivation
- Numerical Derivation –> Root finding
- Root finding <–> Numerical Optimization
  - Solving $f(x) = y$ is actually a root finding problem, i.e. $f(x) - y = 0$.
  - If you can define your cost function with a single parameter x, then actually this is a single variable numerical optimization problem.

What is optimization?

Optimization in engineering is the process of finding the best system design subject to a set of constraints.



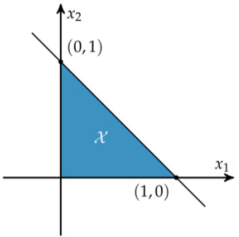
What is optimization? Why do we care?

A typical optimization problem is to

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && x \in X \end{aligned}$$

A design point (x) can be represented as a vector of values corresponding to different design variables.

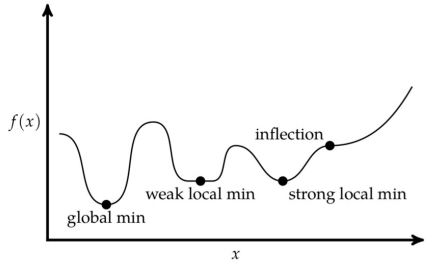
$$\begin{aligned} \underset{x_1,x_2}{\text{minimize}} \quad & f(x_1, x_2) \\ \text{subject to} \quad & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_1 + x_2 \leq 1 \end{aligned}$$



Why is $f'(x)$ important?

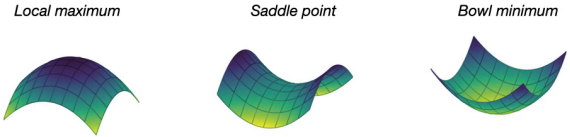
$f'(x) = 0$ is not a sufficient condition

Univariate:



- $f'(x^*) = 0$ and $f''(x^*) > 0$, then strong local minimum
- $f'(x^*) = 0$ and $f''(x^*) < 0$, then strong local maximum

Multivariate:



LOCAL DESCENT: A general approach to optimization

Incrementally improve a design point x by taking a step that minimizes the objective value based on a local model.
The local model may be obtained, for example, from a first- or second-order Taylor approximation.

- Check whether $\mathbf{x}_k$ satisfies the termination conditions. If it does, terminate; otherwise proceed to the next step.
- Determine the descent direction $\mathbf{d}_k$ using local information such as the gradient (or Hessian for multivariate optimization).
- Determine the step size or learning rate α_k. Note that if $\alpha_k == h$, then this is same as Taylor. Here the idea is choose something smaller or larger to control the learning speed!
- Compute the next design point according to: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$
- Note that this approach is valid for both single-variable and multi-variable optimization problems.

Line Search

- Assume that we have a descent direction **d**.
- We need to choose the step factor (a.k.a. learning rate) α to obtain our next design point.
- In line search method, we **select the step factor that minimizes the 1D function**:

$$\underset{\alpha}{\text{minimize}} : f(\mathbf{x} + \alpha \mathbf{d})$$

We need to be cautious in choosing the α.

- Large steps will result in faster convergence but risk overshooting the minimum.
- Smaller steps is more stable but very slow.

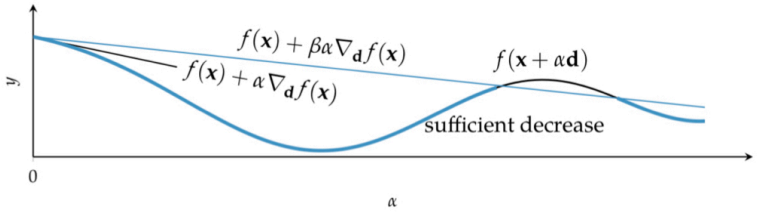
Approximate line search

- Doing an exact line search at each iteration and finding α could be expensive (computationally).
- A cheaper option: find an OK α, update your **x** and **d**, and repeat until satisfied.
- So how can we find that OK α?
- Answer: The Wolfe conditions

  - Sufficient decrease (the step size must cause a sufficient decrease in the objective function value)
  $$f(x^{k+1}) \leq f(x^k) + \beta\alpha\nabla_{d^k} f(x^k)$$
  - Curvature condition (the directional derivative at the next iteration must be shallower)
  $$\nabla_{d^k} f(x^{k+1}) \geq \sigma\nabla_{d^k} f(x^k)$$

Sufficient decrease: $f(x^{k+1}) \leq f(x^k) + \beta\alpha\nabla_{d^k} f(x^k), \quad \beta \in [0, 1]$

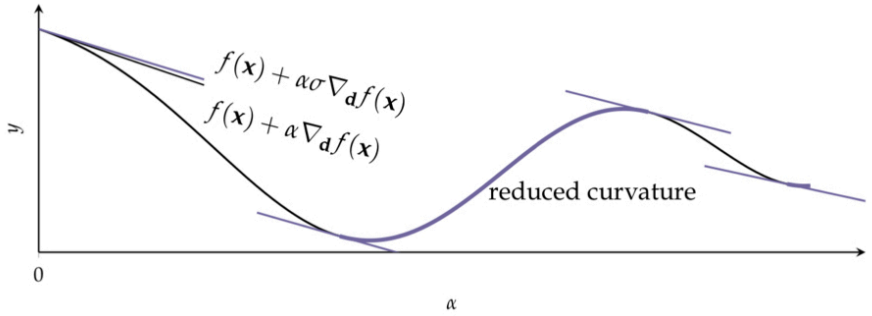
If $\beta = 0$, then any decrease is acceptable.
If $\beta = 1$, then the decrease has to be at least as much as what would be predicted by a first-order approximation.
A common choice is $\beta = 1e - 4$.



Question: what will happen if you adjust β?

Curvature condition: $\nabla_{d^k} f(x^{k+1}) \geq \sigma \nabla_{d^k} f(x^k)$

σ controls how shallow the next directional derivative must be.
We want to stop at a place where the gradient is less negative than the current or even a positive one!
Typically, $\beta < \sigma < 1$. In conjugate gradient method, $\sigma = 0.1$. In Newton's method, $\sigma = 0.9$.



Backtracking line search

- If **d** is a valid descent direction, then there must exist a sufficiently small step size that satisfies the sufficient decrease condition.
- We can thus start with a large step size and decrease it by a constant reduction factor until the sufficient decrease condition is satisfied.
- This algorithm is known as backtracking line search because of how it backtracks along the descent direction.

Example: Backtracking line search

Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$, and a second Wolfe condition parameter $\sigma = 0.9$.

Example: Backtracking line search

Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$, and a second Wolfe condition parameter $\sigma = 0.9$.

Step-1 Trial-1

We check whether the maximum step size satisfies the first Wolfe condition, where the gradient at $\mathbf{x}$ is $\mathbf{g} = [4, 5]$:

$$f(\mathbf{x} + \alpha \mathbf{d}) \leq f(\mathbf{x}) + \beta \alpha (\mathbf{g}^\top \mathbf{d})$$

$$f([1, 2] + 10 \cdot [-1, -1]) \leq 7 + 1 \times 10^{-4} \cdot 10 \cdot [4, 5]^\top [-1, -1]$$

$$217 \leq 6.991$$

It is not satisfied.

Example: Backtracking line search

Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$, and a second Wolfe condition parameter $\sigma = 0.9$.

Step-1 Trial-2

The step size is multiplied by 0.5 to obtain 5, and the first Wolfe condition is checked again:

$$f([1,2] + 5 \cdot [-1,-1]) \leq 7 + 1 \times 10^{-4} \cdot 5 \cdot [4,5]^\top [-1,-1]$$
$$37 \leq 6.996$$

It is not satisfied.

Example: Backtracking line search

Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$, and a second Wolfe condition parameter $\sigma = 0.9$.

Step-1 Trial-3

The step size is multiplied by 0.5 to obtain 2.5, and the first Wolfe condition is checked again:

$$f([1, 2] + 2.5 \cdot [-1, -1]) \leq 7 + 1 \times 10^{-4} \cdot 2.5 \cdot [4, 5]^\top [-1, -1]$$
$$3.25 \leq 6.998$$

The first Wolfe condition is satisfied.

Example: Backtracking line search

Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$, and a second Wolfe condition parameter $\sigma = 0.9$.

Step-2 Trial-1

The candidate design point $\mathbf{x}' = \mathbf{x} + \alpha\mathbf{d} = [-1.5, -0.5]$ is checked against the second Wolfe condition:

$$\nabla_{\mathbf{d}} f(\mathbf{x}') \geq \sigma \nabla_{\mathbf{d}} f(\mathbf{x})$$
$$[-3.5, -2.5]^\top [-1, -1] \geq \sigma\, [4, 5]^\top [-1, -1]$$
$$6 \geq -8.1$$

The second Wolfe condition is satisfied.

Approximate line search terminates with $\mathbf{x} = [-1.5, -0.5]$.

Terminations conditions

- Maximum iterations.
- Absolute improvement. If the change is smaller than a given threshold, it will terminate:

$$f(x_k) - f(x_{k+1}) < \epsilon_a$$

- Relative improvement. If the change is smaller than a given threshold, it will terminate:

$$f(x_k) - f(x_{k+1}) < \epsilon_r |f(x_k)|$$

- Gradient magnitude. We can also terminate based on the magnitude of the gradient:

$$|\nabla f(x_{k+1})| < \epsilon_g$$

The choice of descent direction

- So we assumed that **d** is a valid descent direction,
- Then we said that we could use the line search method to obtain a sufficient decrease.
- Repeating it for many times, we hoped to arrive at the local minimum.

- How can we find a valid descent direction?

Gradient descent

An intuitive choice for the descent direction is the direction of steepest descent (the direction opposite the gradient ∇f)

$$\mathbf{g}_k = \nabla f(\mathbf{x}_k)$$

where $\mathbf{x}_k$ is our design point at descent iteration k.

In gradient descent, we typically normalize the direction of steepest descent

$$\mathbf{d}_k = -\frac{\mathbf{g}_k}{||\mathbf{g}_k||}$$

Gradient descent

If we optimize the step size at each step, we have

$$\alpha_k = \arg\min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{d}^k)$$

The optimization above implies that the directional derivative equals zero. Since

$$\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)^T \mathbf{d}_k = 0$$

$$\mathbf{d}^{k+1} = -\frac{\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)}{||\nabla f(\mathbf{x}_k + \alpha_k)||}$$

If you multiply the above equation's both side with $\mathbf{d}_k$, you can see that these two consecutive directions are orthogonal.

$$(\mathbf{d}_{k+1})^T \mathbf{d}^k = 0$$

More Advanced Methods: Coming Soon...

Since we will need these to understand Neural Networks, we will come back to this subject and continue with

- Conjugate gradient (read the wiki article)
- Momentum
- Adagrad
- RMSprop
- ...