

# Overview of Unconstrained Local Optimization

“**Optimization**”: finding a **maximum** or a **minimum** of an **objective function**  $f : D \subset \mathbb{R}^d \mapsto \mathbb{R}$ .

“**Local**”: as opposed to **global**; the global optimum is the optimum of all local optima.

“**Unconstrained**”: all points in  $D$  are feasible.

## Important Properties of Objective Function

- domain dense or not
- differentiable or not
  - to what order
  - easy or hard to compute
- convex (or concave) or neither
  - if neither, there may be **local** optima

In the following, let  $f(x)$  be the objective function, and assume we want to minimize it.

(To maximize,  $f(x) \leftarrow -f(x)$  and concave  $\leftarrow$  convex.)

# Methods

- **Analytic**: yields closed form for all local maxima.
- **Iterative**: for  $k = 1, 2, \dots$ , given  $x^{(k-1)}$  choose  $x^{(k)}$  so that  $f(x^{(k)}) \rightarrow$  local maximum of  $f$ .

We need

- a **starting point**:  $x^{(0)}$ ;
- a method to **choose**  $\tilde{x}$  with good prospects of being  $x^{(k)}$ ;
- a method to **decide** whether  $\tilde{x}$  should be  $x^{(k)}$ .

How we **choose** and **decide** determines the differences between optimization algorithms.

## Methods

How to **choose** may be based on derivative information or on some systematic method of exploring the domain.

How to **decide** may be based on a deterministic criterion, such as requiring  $f(x^{(k)}) > f(x^{(k-1)})$ , or the decision may be randomized.

## Metamethods: General Tools

- Transformations (for either analytic or iterative methods).
- Any trick you can think of (for either analytic or iterative methods), e.g., alternating conditional optimization.
- Conditional bounding functions (for iterative methods).

# Convergence of Iterative Algorithms

In an iterative algorithm, we have a sequence  $\left\{ \left( f \left( x^{(k)} \right), x^{(k)} \right) \right\}$ .

The first question is *whether* the sequence converges to the correct solution.

If there is a unique minimum, and if  $x_*$  is the point at which the minimum occurs, the first question can be posed more precisely as, given  $\epsilon_1$  does there exist  $M_1$  such that for  $k > M_1$ ,

$$\left| f(x^{(k)}) - f(x_*) \right| < \epsilon_1;$$

or, alternatively, for a given  $\epsilon_2$  does there exist  $M_2$  such that for  $k > M_2$ ,

$$\left\| x^{(k)} - x_* \right\| < \epsilon_2.$$

## Convergence of Iterative Algorithms

Recall that  $f : \mathbb{R}^d \mapsto \mathbb{R}$ , so  $|\cdot|$  above is the absolute value, while  $\|\cdot\|$  is some kind of vector norm. Usually, we take it to be the Euclidean norm; that is the square root of the sum of the squares of the elements.

There are complications if  $x_*$  is not unique as the point at which the minimum occurs.

Similarly, there are complications if  $x_*$  is merely a point of local minimum.

## Assessing Convergence

In practice, we must *decide* when convergence has occurred; that is, whether the iterations have become close enough to the solution. Since we don't know the solution, we cannot base this decision on the convergence criteria above.

We put faith in our algorithm, and decide convergence has occurred if, for some  $e_1, e_2 > 0$ , either

$$|f(x^{(k)}) - f(x^{(k-1)})| \leq e_1$$

or

$$\|x^{(k)} - x^{(k-1)}\| \leq e_2,$$

or both.

## Assessing Convergence

Notice, that lacking any known value, we trust the algorithm to do the right thing; both  $x^{(k)}$  and  $x^{(k-1)}$  are just values in the algorithmic sequence. The fact that this particular sequence — or any sequence, even ones yielding nonincreasing function values — converges does not really get at the question of whether  $x^{(k)} \rightarrow x_*$ .

Note also the subtle change from “<” to “ $\leq$ ”.

For some special class of functions  $\nabla f(x)$  may exist, and we may know that at the solution,  $\nabla f(x_*) = 0$ . In these cases, we may have another criterion for deciding convergence has occurred:

$$\|\nabla f(x_*)\| \leq e_3.$$

# Rate of Convergence of Iterative Algorithms

If the answer to the first question is “yes”, that is, if the algorithmic sequence converges, the next question is *how fast* the sequence converges. (We address this question assuming it converges to the correct solution. )

The rate of convergence is a measure of how fast the “error” decreases. Any of three quantities we mentioned in discussing convergence,  $f(x^{(k)}) - f(x_*)$ ,  $x^{(k)} - x_*$ , or  $\nabla f(x_*)$ , could be taken to be the error. If we take

$$e_k = x^{(k)} - x_*$$

to be the error at step  $k$ , we might define the magnitude of the error as  $\|e_k\|$  (for some norm  $\|\cdot\|$ ). If the algorithm converges to the correct solution,

$$\lim_{k \rightarrow \infty} \|e_k\| = 0.$$

## Rate of Convergence of Iterative Algorithms

Our interest is in how fast this quantity decreases.

Sometimes there is no reasonable way of quantifying the rate at which  $\|e_k\|$  decreases.

## Rate of Convergence

In the happy case (and a common case for simple algorithms), if there exist  $r > 0$  and  $c > 0$  such that

$$\lim_{k \rightarrow \infty} \frac{\|e_k\|}{\|e_{k-1}\|^r} = c,$$

we say the **rate of convergence** is  $r$  and the **rate constant** is  $c$ .

# The Steps in Iterative Algorithms For a Special Class of Functions

The steps in iterative algorithms are often based on some analytic relationship between  $f(x)$  and  $f(x^{(k-1)})$ . For a continuously differentiable function, the most common relationship is the Taylor series expansion:

$$\begin{aligned} f(x) = & f(x^{(k-1)}) + \\ & (x - x^{(k-1)})^\top \nabla f(x^{(k-1)}) + \\ & \frac{1}{2} (x - x^{(k-1)})^\top \nabla^2 f(x^{(k-1)}) (x - x^{(k-1)}) + \\ & \dots \end{aligned}$$

**Note this limitation: “For a continuously differentiable function, ...”.**

We cannot use this method on just any old function.

*In the following, we will consider only this restricted class of functions.*

## Steepest Descent

The steps are defined by truncating the Taylor series. A truncation to two terms yields the steepest descent direction. For a steepest descent step, we find  $x^{(k)}$  along the path  $\nabla f(x^{(k-1)})$  from  $x^{(k-1)}$ .

If  $\nabla f(x^{(k-1)}) \neq 0$ , then moving along the path  $\nabla f(x^{(k-1)})$  can decrease the function value. If  $f(x)$  is bounded below (i.e., if the minimization problem makes sense), then at some point along this path, the function begins to increase. This point is not necessarily the minimum of the function, of course. Finding the minimum along the path, is a one-dimensional “line search”.

## Steepest Descent

After moving to the point  $x^{(k)}$  in the direction of  $\nabla f(x^{(k-1)})$ , if  $\nabla f(x^{(k)}) = 0$ , we are at a stationary point. This may not be a maximum, but it is as good as we can do using steepest descent from  $x^{(k-1)}$ . (In practice, we check  $\|\nabla f(x^{(k)})\| \leq \epsilon$ , for some norm  $\|\cdot\|$  and some positive  $\epsilon$ .)

If  $\nabla f(x^{(k)}) < 0$  (remember we're minimizing the function), we change directions and move in the direction of  $-\nabla f(x^{(k)})$ .

Knowing that we will probably be changing direction anyway, we often truncate the line search before we find the best  $x^{(k)}$  in the direction of  $\nabla f(x^{(k-1)})$ .

# Newton's Method

At the minimum  $x_*$ ,  $\nabla f(x_*) = 0$ .

“Newton's method” for optimization is based on this fact.

Newton's method for optimization just solves the system of equations  $\nabla f(x) = 0$  using

**Newton's iterative method for solving equations:**

to solve the system of  $n$  equations in  $n$  unknowns,  $g(x) = 0$ , we move from point  $x^{(k-1)}$  to point  $x^{(k)}$  by

$$x^{(k)} = x^{(k-1)} - \left( \nabla g(x^{(k-1)})^\top \right)^{-1} g(x^{(k-1)}).$$

Hence, applying this to solving  $\nabla f(x^{(k-1)}) = 0$ , we have the  $k^{\text{th}}$  step in Newton's method for optimization:

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

The direction of the step is  $d_k = x^{(k)} - x^{(k-1)}$ .

## Newton's Method

For numerical reasons, it is best to think of this as the problem of solving the equations

$$\nabla^2 f(x^{(k-1)})d_k = -\nabla f(x^{(k-1)}),$$

and then taking  $x^{(k)} = x^{(k-1)} + d_k$ .

# The Hessian

The Hessian  $H(x) = \nabla^2 f(x)$  clearly plays an important role in Newton's method; if it is singular, the Newton step based on the solution to

$$\nabla^2 f(x^{(k-1)})d_k = -\nabla f(x^{(k-1)}),$$

is undetermined.

The relevance of the Hessian goes far beyond this, however. The Hessian reveals important properties of the shape of the surface  $f(x)$  at  $x^{(k-1)}$ .

The shape is especially interesting at a stationary point; that is a point  $x_*$  at which  $\nabla f(x) = 0$ .

## The Hessian

If the Hessian is negative definite at  $x_*$ ,  $f(x_*)$  is a local maximum.

If the Hessian is positive definite at  $x_*$ ,  $f(x_*)$  is a local minimum.

If the Hessian is nonsingular, but neither negative definite nor positive definite at  $x_*$ , it is a saddlepoint.

If the Hessian is singular, the stationary point is none of the above.

# The Hessian

In minimization problems, such as least squares, we hope the Hessian is positive definite, in which case the function is convex. In least squares fitting of the standard linear regression model, the Hessian is the famous  $X^T X$  matrix.

In maximization problems, such as MLE, it is particularly interesting to know whether  $H(x)$  is negative definite everywhere (or  $-H(x)$  is positive definite everywhere). In this case, the function is concave.

When  $H(x)$  (in minimization problems or  $-H(x)$  in maximization problems) is positive definite but nearly singular, it may be helpful to regularize the problem by adding a diagonal matrix with positive elements:  $H(x) + D$ .

One kind of regularization is ridge regression, in which the Hessian is replaced by  $X^T X + dI$ .

## Modifications of Newton's Method

In the basic Newton step, the direction  $d_k$  from  $x^{(k-1)}$  is the best direction, but the point  $d_k + x^{(k-1)}$  may not be the best point. In fact, the algorithm can often be speeded up by not going quite that far; that is, by “damping” the Newton step and taking  $x^{(k)} = \alpha_k d_k + x^{(k-1)}$ . This is a line search, and there are several ways of doing this. In the context of least squares, a common way of damping is the **Levenberg-Marquardt** method.

Rather than finding  $\nabla^2 f(x^{(k-1)})$ , we might find an approximate Hessian at  $x^{(k-1)}$ ,  $\widetilde{H}_k$ , and then solve

$$\widetilde{H}_k d_k = -\nabla f(x^{(k-1)}).$$

This is called a **quasi-Newton** method.

## Modifications of Newton's Method

In MLE, we may take the objective function to be the log likelihood, with the variable  $\theta$ . In this case, the Hessian,  $H(\theta)$ , is  $\partial^2 \log L(\theta; x) / \partial \theta (\partial \theta)^\top$ . Under very general regularity conditions, the expected value of  $H(\theta)$  is the negative of the expected value of  $(\partial \log L(\theta; x) / \partial \theta) (\partial \log L(\theta; x) \partial \theta)^\top$ , which is the Fisher information matrix,  $I(\theta)$ . This quantity plays an important role in statistical estimation. In MLE it is often possible to compute  $I(\theta)$ , and take the Newton step as

$$I(\theta^{(k)}) d_k = \nabla \log L(\theta^{(k-1)}; x).$$

This quasi-Newton method is called **Fisher scoring**.

## More Modifications of Newton's Method

The method of solving the Newton or quasi-Newton equations may itself be iterative, such as a conjugate gradient or Gauss-Seidel method. (These are “inner loop iterations”.) Instead of continuing the inner loop iterations to the solution, we may stop early. This is called a **truncated Newton method**.

The best gains in iterative algorithms often occur in the first steps. When the optimization is itself part of an iterative method, we may get an acceptable approximate solution to the optimization problem by stopping the optimization iterations early. Sometimes we may stop the optimization after just one iteration. If Newton's method is used, this is called a **one-step Newton method**.

# Alternating Conditional Optimization

The computational burden in a single iteration for solving the optimization problem can sometimes be reduced by more than a linear amount by separating  $x$  into two subvectors. The optimum is then computed by alternating between computations involving the two subvectors, and the iterations proceed in a zigzag path to the solution.

Each of the individual sequences of iterations is simpler than the sequence of iterations on the full  $x$ .

# Alternating Conditional Optimization

For the problem

$$\min_x f(x)$$

if  $x = (x_1, x_2)$  that is,  $x$  is a vector with at least two elements, and  $x_1$  and  $x_2$  may be vectors), an iterative alternating conditional optimization algorithm may start with  $x_2^{(0)}$ , and then for  $k = 0, 1, \dots$ ,

1.  $x_1^{(k)} = \arg \min_{x_1} f(x_1, x_2^{(k-1)})$

2.  $x_2^{(k)} = \arg \min_{x_2} f(x_1^{(k)}, x_2)$