

**Course Name: Optimization Theory and Algorithms**  
**Professor Name: Dr. Uday K. Khankhoje**  
**Department Name: Electrical Engineering**  
**Institute Name: Indian Institute of Technology Madras**  
**Week - 03**  
**Lecture - 18**

**Unconstrained Optimization - Overview of Algorithms and Choosing a Descent Direction**

Overview of Algorithms

So, now the next thing is having gotten this, let us just talk sort of give an overview of the four main families of algorithms that we will talk about in this course. The basic philosophy always is you know start from some point  $x_0$ , then you are going to go undergo a sequence of iterations. Let us call these iterations. Very rarely would you have a case where you go from start to the solution in one shot; that happened in high school problems, not in real-life engineering. Hopefully, you land up at the solution,  $x^*$ .

Now, what distinguishes all of these algorithms is how you go from  $x_k$  to  $x_{k+1}$ ; that is where all the details differ. As I have mentioned, there are two families over here: one is called line search and the other is called trust region. Our overall optimization of going from  $x_0$  to  $x^*$ , when I talk about the real nuts and bolts, what is it going from  $x_k$  to  $x_{k+1}$ .

④ Conjugate Gradient Method →

$$p_k = -\nabla f_k + \beta_k p_{k-1}$$

Descent directions → requirements.

**OPTIMIZATION THEORY AND ALGORITHMS**

If I write it like this, supposing this is my  $x_k$ , I am standing over here, here is  $x_{k+1}$ . We are in a vector space; that means for example,  $x_k$  and  $x_{k+1}$  are  $n$ -dimensional vectors. Between these two points, I need to specify two things: one is a direction and a length. That is how I will get from here to there.

So, this direction I can give it a symbol  $p_k$ . The common symbol for how much I should walk along this direction is  $\alpha_k$ . I can think of this as length and this as direction. Having done this, I get a chance to show you formal notation in optimization which is used a lot. Let me write it down.

(4) Conjugate Gradient Method  $\rightarrow$

$$p_k = -\nabla f_k + \beta_k p_{k-1}$$

Descent directions  $\rightarrow$  requirements.

① The direction of steepest descent is  $-\nabla f_k$

② A descent direction is legitimate if it makes an angle strictly less than  $\pi/2$  with the previous direction.

**OPTIMIZATION THEORY AND ALGORITHMS**

$$x_{k+1} = x_k + \alpha_k p_k$$

I am at  $x_k$ , but I do not know what  $p_k$  to take and I do not know what  $\alpha_k$  to walk. My optimization process is a journey or a process to find out  $\alpha_k$  and  $p_k$ . Those are the variables of my optimization. I will write this as


$$\text{minimize } \min \{ \alpha_k, p_k \}$$

When I write min, it means minimize whatever is to the right of it. So, minimize min. Below the word, I should write down what are the variables of my optimization. So, what are the variables of my optimization?  $\alpha_k$  and  $p_k$ .

If I just write min, this would mean minimize this whole thing. You may have seen this notation written in a lot of papers, and now you know what that means. The variables of optimization are below, and the objective function is to the right of min; this is what it means.

There is another related thing that you will see: you will find argmin. Now, what does that mean? It means basically arg means arguments. So, what are the arguments to this objective function? They are  $\alpha^*$  and  $p^*$ . So, the optimization problem was to minimize, but at each step, I need to solve this optimization problem argmin means tell me what are the arguments that minimize this function; they would be  $\alpha^*$  and  $p^*$ , which would get me to  $x_{k+1} = x_k + \alpha^* p_k$ .

So, that is the meaning of this. Whenever you see a textbook or a paper saying min or argmin, look for what is below. Those are the knobs in your hand. Nothing else can be changed. For example, notice  $x_k$  was not in my hand. Why? Because my algorithm has found itself at  $x_k$ .



Overview of Algorithms.

Start from  $x_0$  → iterations  $\{x_k\}$  → Soln  $x^*$

$x_{k+1}$

line search      trust region

$\min_{\alpha} f(x_k + \alpha p_k)$

length  $\alpha_k$

$x_k$   $\rightarrow$   $x_{k+1}$

$p_k$  direction


**OPTIMIZATION THEORY AND ALGORITHMS**

It's given to me. I cannot change it. What I can change is where I go from there. So I have a freedom in choosing  $p_k$ . I have a freedom in figuring out how much to walk along that.

My first-order, second-order theorems are going to help me figure out these directions and distances. That is the name of the game. We will start by talking about line search algorithms because they are the most popular algorithms. In fact, you know what everyone loves these days; machine learning takes one of those members of that family.

### Line Search Algorithms

Line search algorithms correspond to different choices of  $p_k$ . The first and the simplest is the steepest descent or gradient descent; the two mean the same thing. Over here you do not try to find out  $p_k$  at every step; you take it as a recipe, and I take this to be equal to the negative of the gradient. This is your steepest descent or gradient descent algorithm.



Handwritten notes on a whiteboard illustrating the line search method for finding the search direction  $p_k$  at point  $x_k$ . The notes show a point  $x_k$  and a direction vector  $p_k$  leading to  $x_{k+1}$ . The search direction is defined as:

$$\{\alpha^*, p^*\} = \underset{\alpha, p_k}{\operatorname{arg\,min}} f(x_k + \alpha p_k)$$

The text "line search" is written above the equation. A stick figure is drawn on the left side of the whiteboard. In the bottom right corner, a man is visible, looking at the whiteboard.

**OPTIMIZATION THEORY AND ALGORITHMS**

Why we choose this, I will prove it to you shortly. This is the first one. The second one is very powerful, called the Newton method. In this, again we will prove it; I will note down the expression that you have is the Hessian comes in with an inverse.

$$p_k = -H_k^{-1} \nabla f_k$$

But this is not enough; you also need the Hessian to be positive definite. These are, that is why I have written it as an AND condition. If the Hessian is positive definite, then you can plug in this expression and get your  $p_k$ .

The third method was invented because the second method ended up being very expensive. If I have a function of many variables, calculating second-order derivatives can become very computationally expensive. So, people came up with tricks to approximate the Hessian. Therefore, the word itself is explanatory; they are called quasi-Newton methods.

For all points where you are running the iteration, the quasi-Newton method basically does not work with the Hessian; it does something like this:

$$B_k = \text{approximation of the Hessian}$$



Line Search Algorithms → different choices of  $p_k$ .

① Steepest descent / gradient descent

$$p_k = -\nabla f_k$$

② Newton method,  $p_k = -(\nabla^2 f_k)^{-1} \nabla f_k$  AND

$\nabla^2 f$  to be P.D.

③ Quasi Newton,  $p_k = -(B_k) \nabla f_k$

↪ Cheaper approx of  $(\nabla^2 f_k)^{-1}$

Just one word of why this Newton method is very popular; we will state this as we develop in more detail. The rate of convergence, remember we had spoken about the rate at which sequences can converge, turns out that the Newton method has a quadratic rate of convergence whereas the steepest descent has a linear rate of convergence.

So, that can mean the difference between hours and minutes in your simulation or your problem getting solved. So, this is quadratic. If you have a high-performance computing cluster where you can compute the Hessian, use it; do not go for gradient descent.

The fourth method is something very interesting; it is based on a first-order method, but much more clever than a first-order method. Many of you may have heard of it; it is called the conjugate gradient method. We will spend a fair amount of time on this conjugate gradient method as well.

Here, the cleverness comes from how I choose my  $p_k$ . The  $p_k$  if I just wrote this as negative gradient, it becomes gradient descent. What is clever is that it does not take just this; it modifies it from the previous direction it had been in.

So, it writes it as a linear combination of the negative gradient. Now, how we choose this  $\beta_k$  is there is a lot of very elegant and clever math over here, and this gives us a little faster rate of convergence than the conjugate gradient. The other great advantage of methods 1 and 4 is that they can easily be generalized to non-linear methods, non-convex functions, and so on. Any questions so far?

Now, we have mentioned everywhere different choices of  $p_k$ . We want to find out are there any restrictions on how we can choose  $p_k$ ; how do we design  $p_k$ ? These formulas are given to us; we will derive them. But before we go to that, let us just write down a little bit more rigorous conditions on what this  $p_k$  should satisfy.

These are what are called descent directions, and we are talking about the requirements on them. Descent is clear because if I am at some point, I want to minimize it; I need to go in a direction where there is a descent of the function value happening.

They are called descent directions. The first condition is that the direction of steepest descent anyone wants to guess? The negative gradient. If you were in a hurry to climb down a hill for example, literally take the example of a hill; if you stand here, the best direction to go is this way.

So, the first requirement is that my objective function decreases when I move in the direction of  $p_k$ . It is that simple. We can write it as

$$\nabla f_k^T p_k < 0$$

So, this is a condition for descent. We want our  $p_k$  to be something such that if we take the dot product with the gradient of the function, it is negative; then that means I am walking downhill; if it is positive, I am walking uphill.

The second condition is a bit subtle, but it becomes very important when we start looking at various forms of optimization and whether the optimization methods converge or not. So, one of the things that we want from our descent direction is that we can ensure that we do not take a step that is too big because if you take a step that is too big and we will prove this and show you examples of it, you might overshoot the minimum.

So, we have to be careful about that. So, the second condition we will write down is the length or the norm of the step should be controlled. So, for example, I can say that we have a norm; we have some upper bound  $M$  and say that  $\|p_k\| < M$ .

Now, you can tweak  $M$  depending on your optimization problem. So, these are the two conditions. You need to have a direction  $p_k$  such that the angle between  $p_k$  and  $\nabla f_k$  is acute, and the second one is the length of the step should be controlled. So, those are the two things that we want.

As I said, these algorithms typically you can pick from there. Now, we will talk about the line search algorithms, and one very good way of thinking about line search is that at each iteration I am going to generate a direction  $p_k$  from the previous iteration. But I am going to try different lengths along that direction.


So, the way to write this is you can think of the step as a one-dimensional optimization problem. So, if you write down

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

So, I am trying to minimize  $f$  along a line. I have already selected my direction; it is known. I am going to see what is the best length along that direction; that is the essence of the line search algorithm.

The notation is sort of standard; the two notations you will see quite often is either this  $\phi(\alpha)$  or some papers may write it as  $f_k(\alpha)$ , in which case you have  $f_k(\alpha) = f(x_k + \alpha p_k)$ . Now what are you going to do? You are going to optimize  $\alpha$ .

So, you will be looking for a point  $\alpha^*$  such that  $\alpha^*$  gives you the minimum function value when you move along the line defined by  $p_k$ . You are going to call that your optimal length, and we will use that to move to the next iteration

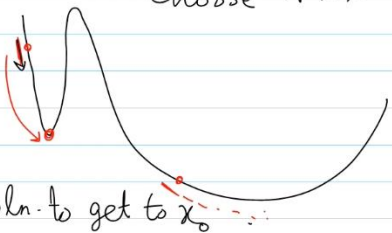


How to choose  $x_0$ ?


- ↳ Choose at Random.
- ↳ Start from  $\vec{0}$
- ↳ Solve a simpler problem.

Pick random  $\{x_k\}$   
choose  $\text{Max} \{\|\nabla f(x_k)\|\}$

↳ use that soln. to get to  $x_0$



**OPTIMIZATION THEORY AND ALGORITHMS**



$$x_{k+1} = x_k + \alpha^* p_k$$

So, that is the method of line search; it is very powerful because I am already using a good direction. Now, I just need to find a good length. It is very fast, but it can also be very costly to do this line search because you might need to do a lot of iterations to do that, especially if it is a very complicated function.

So, the goal of line search is to try and reduce the cost of each step as much as possible. The first thing I want to show you is that how do you know when to stop? How do you know when to stop, and there are conditions that are known for this as well. Typically, what happens is you compute the value of the gradient at  $x_k$  and you check for its magnitude.

If it is too small, it means the function is flat, and the condition typically we will write is that

$$\|\nabla f_k\| < \epsilon$$

So, it is telling you that my function has stopped changing or it is nearly stationary. I am at a point where I do not need to go anymore. So, typically, this is one of the common conditions that you will see.

So, you will see that when it is less than some small tolerance, and you will notice the epsilon is typically very small, something like  $10^{-4}$  or  $10^{-5}$ , or depending on how precise you want your solution to be. And of course, what happens is this will depend on your optimization problem, but it is a good criterion to start with.

The second thing is this that at each step you are not going to move; if your step does not give you a sufficiently good improvement, I will stop. So, you will typically see that I can measure the ratio of improvements in my function value; I can say that the improvement should be larger than  $\eta$ , and  $\eta$  typically will be between 0.01 to 0.1.

So, these are the two conditions that are sort of common and how you can check when to stop. We will illustrate this through examples; we will give you a lot of nice problems in practice.

The third thing is we are just not interested in how to pick the direction. We also want to talk about what do you do if the algorithm fails. What do I mean by that? Well, sometimes the optimization may get stuck in a local minimum, and it would not be able to get out of it.

So, one of the common strategies to try and escape local minima is to perturb your variables. So, when I have gotten to this point, I can do something like this; I can go back to some previous point or a random point nearby and start the algorithm again. This is what is called perturbing the starting point. In fact, it is a very very common practice in optimization algorithms, and you may see it in machine learning.

The last thing is just to summarize. So, line search algorithms, as I have mentioned to you, are generally popular because they are simple to implement and they are easy to work with. In the next class, we will take a deep dive into gradient descent; we will look at the proofs and the geometry of optimization because it is one of the simplest methods, and once you understand that, we will move on to the more complicated ones.