Continuous Optimisation

Newton-Ralphson method

When you search this on YouTube, you get explanations of how it works for approximating the solutions to an equation, say $y(x) = 0$. This can also be rephrased as minimising $f(x) = (y(x))^2$. This is when the method is applied in its original form. The whole idea behind the method is that for a scalar function $f(\mathbf{x})$, you can approximate as a quadratic in $\mathbf{t} = \mathbf{x} - \mathbf{x_n}$ by Taylor approximation around $\mathbf{x_n}$, like this :

$$f(\mathbf{x}) = f(\mathbf{x_n} + \mathbf{t}) \approx f(\mathbf{x_n}) + (\nabla f(\mathbf{x_n}))\mathbf{t} + \frac{1}{2}\mathbf{t^T H\, t}$$

and then minimise that approximation to get the next iteration, $\mathbf{x_{n+1}}$ by setting the gradient wrt $\mathbf{t}$ to be 0, like this :

$$\nabla_{\mathbf{t}} f(\mathbf{x}) = \nabla f(\mathbf{x_n} + \mathbf{t}) = 0 + \nabla f(\mathbf{x_n}) + \frac{1}{2}\mathbf{Ht} + \frac{1}{2}\mathbf{t^T H} = \nabla f(\mathbf{x_n}) + \mathbf{Ht} = 0$$

Note : The Hessian $\mathbf{H}$ is a symmetric matrix and thus $\mathbf{t^T H} = \mathbf{Ht}$ (don't try to prove this by symbolic manipulation, but by writing down the matrix as a bunch of column vectors) .

Thus, we get $\mathbf{t} = -\mathbf{H}^{-1}(\nabla f(\mathbf{x_n}))$. Putting it all together, we get the **final equation** :

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \mathbf{H}^{-1}\nabla f(\mathbf{x_n})$$

Applying this general method to $f(x) = (y(x))^2$ gives us $x_{n+1} = x_n - [2y'^2 + 2yy'']^{-1}[2yy'] = x_n - \left[\frac{yy'}{y'^2+yy''}\right]_{x_n}$. Since $y(x_n) \approx 0$, so we just have $x_{n+1} = x_n - \frac{y(x_n)}{y'(x_n)}$

This is the equation that you usually see all around.

To see the general method in action, read my friend's blog :

Blogs - Optimizing bivariate functions

Understanding optimization of bivariate functions using Newton's method and L-BFGS.

https://kishan-ved.github.io/Blogs/posts/secondorder/

Condition number

Suppose you are trying to solve an equation $f(\mathbf{x}) = \mathbf{c}$ but some numerical method, and after a lot of (n) iterations, you end up at the approximate value $\mathbf{x}_n$ which gives the value of the function as $f(\mathbf{x}_n)$, you know the relative error in the value of f , which is $\frac{||f(\mathbf{x}_i)-\mathbf{c}||}{\mathbf{c}}$, but you don't know the relative error in $\mathbf{x}$, namely $\frac{||\mathbf{x_n}-\mathbf{x}_\infty||}{||\mathbf{x}_\infty||}$. To get an upper bound on this error, we define the condition number as

$$K = \max_n \frac{\left(\frac{||\mathbf{x_n}-\mathbf{x}_\infty||}{||\mathbf{x}_\infty||}\right)}{\left(\frac{||f(\mathbf{x}_i)-\mathbf{c}||}{\mathbf{c}}\right)}$$

For example, for a matrix equation $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} = \mathbf{c}$, if the error in $\mathbf{x}$ is $\delta\mathbf{x}$, then as

$$\frac{\left(\frac{||\delta\mathbf{x}||}{||\mathbf{x}||}\right)}{\left(\frac{||\delta\mathbf{f}||}{||\mathbf{f}||}\right)} = \frac{||\mathbf{A}\mathbf{x}||}{||\mathbf{x}||}\left(\frac{||\mathbf{A}\,\delta\mathbf{x}||}{||\delta\mathbf{x}||}\right)^{-\mathbf{1}} \leq \sigma_{max}\ {\sigma_{min}}^{-1}$$

where σ_{max} and σ_{min} are the maximum and minimum singular values for $\mathbf{A}$, we have the condition number as σ_{max}

σ_{min} .

Gradient Descent

Given a function $f(\mathbf{x})$ to be minimised, step along the negative gradient by doing:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i(\nabla f_{(\mathbf{x_i})})^T$$

The transpose is because we usually consider the gradient to be a row vector.

The step size γ_i depends on $\mathbf{x}_i$.

Gradient Descent with Momentum

Rather that viewing the gradient as the velocity (or momentum), if we view it as acceleration (or force), then we get the gradient descent with momentum.

It is able to escape local minimums, unlike the normal gradient descent, and takes bigger steps when it knows it's going in the right direction, thus completing faster.

We use the change in $\mathbf{x}$ in the last update, denoted as $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$ as well to calculate the next step, like this :

$$\Delta\mathbf{x}_{i+1} = \mathbf{x}_{i+1} - \mathbf{x}_i = -\gamma_i(\nabla f)_{(\mathbf{x}_i)})^T + \alpha\Delta\mathbf{x}_i$$

The reason for adding the $\alpha < 1$ is to include "friction" , because otherwise we can end up in an infinite loop (like a friction-less ball moving in bowl like surface and always reaching the same height when its speed is 0, and thus never stopping (both velocity and acceleration being 0))

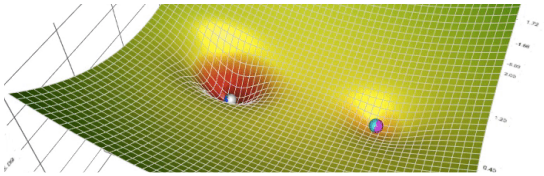
Here's a nice article explaining this (and more variations) visually :

A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSPro

A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSProp…

Why can AdaGrad escape saddle point? Why is Adam usually better? In a race down different terrains, which will win?

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentu…



Step Size

Sure, you can take steps of constant sizes, but you can do more. Say you want to minimise $f(\mathbf{x})$ and you have reached a point $\mathbf{x_n}$. Now, you want to descend along the gradient again. The question is, till when ? The answer is: till we are **descending,** that is, till $f(\mathbf{x})$ is decreasing. Basically, till $\frac{d}{d\gamma} f(\mathbf{x_n} - \gamma \nabla f(\mathbf{x_n})) = 0 \iff (\nabla f(\mathbf{x_n} - \gamma \nabla f(\mathbf{x_n})))(\nabla f(\mathbf{x_n}))^T = 0$. The solution to this equation would be the exact value of optimum step size. But most of the times you don't have an analytical solution. This can be approximated as $(\nabla f(\mathbf{x_n}) - \gamma (\nabla f(\mathbf{x_n}))\mathbf{H}_f)(\nabla f(\mathbf{x_n}))^T \implies \gamma = \frac{||\nabla f(\mathbf{x_n})||^2}{\nabla f(\mathbf{x_n})\mathbf{H}_f(\nabla f(\mathbf{x_n}))^T}$. But this is computationally expensive unless you have a very simplified form, and is more or less useless, because if you are calculating the hessian anyway, just use Newton's method, as it will give a better result.

What we really need is a cheap way to find when γ should increase and when not. This is where **backtracking line search** comes in.

You start with $\gamma = 1$ and reduce γ by a fixed scaling factor $\beta < 1$ while $f(\mathbf{x_n} - \gamma \nabla f(\mathbf{x_n})) \geq f(\mathbf{x_n}) - \frac{\gamma}{2}||\nabla f(\mathbf{x_n})||^2$ is true. You go to the next step when γ doesn't satisfy this inequality. One way to speed this up is to start the iteration with the chose value of γ in the last step. If it is already small enough to not satisfy the inequality, you keep increasing by β until the last γ that doesn't satisfy the condition, or you can just start from $\gamma = 1$ again.

Lagrange Multipliers for Equations

This is a rather nice method where the whole idea is that to minimise (locally) a function $f(\mathbf{x})$ under the constraints $g_i(\mathbf{x}) = 0$, which can be written compactly as $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, you consider the Lagrangian $L(\mathbf{x}, \mathbf{a}) = f(\mathbf{x}) + \mathbf{a^T g}(\mathbf{x})$ and minimise (locally) this new function under no constraints by setting the gradient to 0.

Basically, our new problem becomes $\nabla f(\mathbf{x_0}) + a_1 \nabla g_1(\mathbf{x_0}) + a_2 \nabla g_2(\mathbf{x_0}) + \ldots = 0$ which can be read as " $\nabla f(\mathbf{x_0}), \nabla g_1(\mathbf{x_0}), \nabla g_2(\mathbf{x_0}) \ldots$ " are linearly dependent row vectors, still following the old constraints, $\mathbf{g}(\mathbf{x_0}) = \mathbf{0}$.

Proof :

Consider any point in the locality of minima $\mathbf{x_0}$ of f , say $\mathbf{x_1} = \mathbf{x_0} + t\mathbf{b}$ with $g(\mathbf{x_1})$ where $\mathbf{b}$ is a unit vector and t is a scalar. Both $t, \mathbf{x_1}$ are in fact functions of $\mathbf{b}$. Now, suppose $\mathbf{x_1} \to \mathbf{x_0}$, then we have $\lim_{\mathbf{x_1} \to \mathbf{x_0}} \frac{g_i(\mathbf{x_1}) - g_i(\mathbf{x_0})}{t} = \nabla g_i(\mathbf{x_0})\ \mathbf{b} = 0$.

Now, since $\mathbf{x_0}$ is the solution to the minimisation problem (minimise f under constraints), thus the function f must not change on going in any direction (by an infinitesimal distance) where the constraint is being followed, say $\mathbf{b}$ for example. What this means is that $\lim_{\mathbf{t} \to \mathbf{0}} \frac{f(\mathbf{x_0} + \mathbf{b}t) - f(\mathbf{x_0})}{t} = \nabla f(\mathbf{x_0})\ \mathbf{b} = 0$. Reiterating we are saying there should no way to move such that f would change and $\mathbf{g} = \mathbf{0}$ is still followed, as otherwise, we can just move in that or its opposite direction to decrease f .

So, what we have just showed is that any unit vector $\mathbf{b}$ perpendicular to all of $\nabla g_i(\mathbf{x_0})$ is also perpendicular to $\nabla f(\mathbf{x_0})$, but that means that $\nabla f(\mathbf{x_0})$ doesn't have any component in the null space formed by all of $\nabla g_i(\mathbf{x_0})$, and thus it is the vector space spanned by the set of $\nabla g_i(\mathbf{x_0})$, that is to say $\nabla f(\mathbf{x_0}), \nabla g_1(\mathbf{x_0}), \nabla g_2(\mathbf{x_0}) \ldots$ are linearly dependent, which is what we wanted to prove.

This method even works for inequality constraints $h(\mathbf{x}) \leq 0$, since that can be converted to an equality by introducing a new variable x_{n+1} in the vector $\mathbf{x}$ and writing $h(\mathbf{x}) + x_{n+1}^2 = 0$. The resultant problem can be simplified a lot and that simplified version has a much more intuitive derivation than the one you would do after applying this trick. We'll discuss it later.

Gradient descent under equality constraints.

Notice that we talked about moving under a bunch of constraints $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ in order to decrease the value of $f(\mathbf{x})$. This is a lot like gradient descent, except here we aren't moving along the gradient $\nabla f(\mathbf{x})$ directly, but its projection on the hyperplane formed by $\nabla g_i(\mathbf{x})$

on the hyperplane formed by $\nabla g_i(\mathbf{x})$.

Lagrangian multiplier method for inequality constraints.

Consider the function $f(\mathbf{x})$ to be locally minimised under the constraints $g_i(\mathbf{x}) \leq 0$. For a solution $\mathbf{x_0}$, first of all, at least one of the inequalities must become an equality, which we usually call as the constraint being tight. This is because if $\forall i,\ g_i(\mathbf{x_0}) < 0$, then $\forall i,\ g_i(\mathbf{x}) < 0$ in all directions in the locality of $\mathbf{x_0}$, and thus we can descend along the gradient to reach a point in this locality that follows all constraints and has a smaller value of f . So now, WLOG, suppose that $g_i(\mathbf{x_0}) = 0\ \forall i \leq k$ and $g_i(\mathbf{x_0}) < 0\ \forall i > k$, then moving a little around $\mathbf{x_0}$ doesn't affect the inequalities $g_i(\mathbf{x_0}) < 0\ \forall i > k$. So we can move following the constraints $g_i(\mathbf{x_0}) = 0\ \forall i \leq k$ as if the other constraints didn't exist. Clearly for $\mathbf{x_0}$ to be a local minima, the projection of gradient $\nabla f(\mathbf{x_0})$ on the hyperplane formed by $G = \{\nabla g_i(\mathbf{x_0}) \mid 1 \leq i \leq k\}$ should be $\mathbf{0}$ and thus it is linearly dependent on these vectors, which means $\nabla f(\mathbf{x_0}) + a_1 \nabla g_1(\mathbf{x_0}) + a_2 \nabla g_2(\mathbf{x_0}) + \cdots + a_k \nabla g_k(\mathbf{x_0}) = 0$. Also, suppose we moved a little along a unit vector $\mathbf{b}$ such that $\nabla g_i(\mathbf{x_0})\mathbf{b} = 0\ \forall\ i \leq k\ ;\ i \neq j$, and $\nabla g_j(\mathbf{x_0})\mathbf{b} < 0$ (Yes, such a unit vector always exists for linearly independent vectors (a subset of G in this case) due to the existence of reciprocal system of vectors) , then we have $\nabla f(\mathbf{x_0})\mathbf{b} = -a_j \nabla g_j(\mathbf{x_0})\mathbf{b}$. So if $a_j < 0$, then $\nabla f(\mathbf{x_0})\mathbf{b} < 0$ and thus we can step along $\mathbf{b}$ to reduce f . This should not happen and thus we have the additional restriction that $a_i \geq 0\ \forall i$.

Thus any critical point $\mathbf{x_0}$ in the old problem is also part of a critical point $(\mathbf{x_0}, \mathbf{a_0})$ of the function $L(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{n} a_i g_i(\mathbf{x})$ constrained by $a_i \geq 0,\ g_i \leq 0\ \ \forall i$. Here we can use the fact that if $g_j(\mathbf{x_0}) < 0$, then a_j has to be necessarily 0. (or else there are points in the locality where L is bigger and points where it is smaller found by decreasing or increasing a_j , and thus we are not at the minimum). So, in effect, in our partial differential equations, we are still considering the gradients of only those constraint functions which evaluate to 0 at $\mathbf{x_0}$, that is their constraint is tight.

Of course the fact that $g_i(\mathbf{x_0}) < 0 \implies a_i = 0$ also tells us that $a_i > 0 \implies a_i \neq 0 \implies g_i(\mathbf{x_0}) \not< 0 \implies g_i(\mathbf{x_0}) = 0$. Basically, at least one of a_i or $g_i(\mathbf{x_0})$ must be 0. Thus you can divide this problem in $2^n - 1$ cases based on whether $a_i = 0$ or $a_i > 0$, and solve each case by setting gradient to 0. Remember that the case where $g_i(\mathbf{x_0}) < 0\ \forall i \implies a_i = 0\ \forall i$ is not allowed and this was established at the very start

Gradient descent under general constraints

If you have reached a point $\mathbf{x_1}$ somewhere in the process, then to decide the direction in which you should step, you consider the set G_0 which contains the gradients of all the constraint functions for an equality constraint. You add to this set the gradient of any constraint function g_1 for the inequality constraint $g_1(\mathbf{x_0}) \leq 0$ for which the constraint is tight and the projection of the gradient ∇f on the hyperplane spanned by G_0 has a negative dot product with ∇g_1 to get the updated set G_1. Then you add another such tight inequality constraint function's gradient, which has a negative dot product with the projection of ∇f on G_1 , and thus get the updated set G_2. You keep repeating this process until you can't find another constraint function that satisfies this rule. The projection of ∇f on the hyperplane formed by this final set, say G_k gives us the direction to move in.

Notice that here we are quite literally constraining the gradient by not letting its projection, which would give us the optimum direction to move in under only the constraints involved in the set G_i , have a negative dot product with the gradient of any other tight constraint function . If it was negative, we would break the constraint by descending along this projection as then g_i would increase on such a step and become positive. It's basically like we update our optimal direction to move in as more constraints are put on us.

Convex function

A function $f : D \to C$ is convex iff $f(\alpha\mathbf{x_2} + (1-\alpha)\mathbf{x_1}) \leq \alpha f(\mathbf{x_2}) + (1-\alpha)f(\mathbf{x_1}) \ \ \forall\, \alpha \in [0,1]\ \forall \mathbf{x_1}, \mathbf{x_2} \in D$.

For example, an upward quadratic is a convex function .

This inequality is called Jensen's inequality, and is the definition of a convex function, not its property .A concave function is where the inequality is switched to $\geq$.

Consider $\mathbf{x_2} = \mathbf{x_1} + \mathbf{x}$, then for a convex function, we have $f(\mathbf{x_1} + \alpha\mathbf{x}) \leq \alpha(f(\mathbf{x_1} + \mathbf{x}) - f(\mathbf{x_1})) + f(\mathbf{x_1})$. This can be rewritten as $\frac{f(\mathbf{x_1}+\alpha\mathbf{x}) - f(\mathbf{x_1})}{\alpha} \leq f(\mathbf{x_1} + \mathbf{x}) - f(\mathbf{x_1})$. From here, it's easy to see that as $\alpha\mathbf{x} \to \mathbf{0}$, we get $\nabla f(\mathbf{x_1})\mathbf{x} \leq f(\mathbf{x_1} + \mathbf{x}) - f(\mathbf{x_1})$. Since this is a general property, we can also write $\nabla f(\mathbf{x_1} + \mathbf{x})(-\mathbf{x}) \leq$

get $\nabla f(\mathbf{x_1})\mathbf{x} \geq f(\mathbf{x_1}+\mathbf{x}) - f(\mathbf{x_1})$. Since this is a general property, we can also write $\nabla f(\mathbf{x_1}+\mathbf{x})(-\mathbf{x}) \geq f((\mathbf{x_1}+\mathbf{x})+(-\mathbf{x})) - f(\mathbf{x_1}+\mathbf{x})$, or simply $\nabla f(\mathbf{x_1}+\mathbf{x})\mathbf{x} \geq f(\mathbf{x_1}+\mathbf{x}) - f(\mathbf{x_1})$. What this means is that $\nabla f(\mathbf{x_1}+\mathbf{x})\mathbf{x} \geq \nabla f(\mathbf{x_1})\mathbf{x} \ \ \forall \mathbf{x_1}, \mathbf{x} \in D$, or rewritten, $(\nabla f(\mathbf{x_1}+\mathbf{x}) - \nabla f(\mathbf{x_1}))\mathbf{x} \geq 0 \ \ \forall \mathbf{x_1}, \mathbf{x} \in D$. Thus it's also true for $\mathbf{x} = t\mathbf{b}$ where $\mathbf{b}$ is some vector. Thus $(\frac{\nabla f(\mathbf{x_1}+t\mathbf{b}) - \nabla f(\mathbf{x_1})}{t})\mathbf{b} \geq 0$. It's easy to see that as $t \to 0$, the thing inside the bracket is just $\mathbf{b^T H}_f$. Thus you have $\mathbf{b^T H b} \geq 0 \ \forall \ \mathbf{b}$.

A constrained problem of minimising $f(\mathbf{x})$ under the constraints $\mathbf{g(x)} = \mathbf{0}$ is called a convex problem if $f(\mathbf{x}), g_i(\mathbf{x})$ are convex functions.

Linear Programming

Consider a the function $f(\mathbf{x}) = \mathbf{c^T x}$. In order to minimise f under the constraint $\mathbf{Ax} \leq \mathbf{b}$, we consider the Lagrangian $L(\mathbf{x}, \mathbf{a}) = \mathbf{c^T x} + \mathbf{a^T}(\mathbf{Ax} - \mathbf{b}) = \mathbf{x^T}(\mathbf{c} + \mathbf{A^T a}) - \mathbf{a^T b}$ and maximise it under the constraints that $a_i \geq 0 \geq g_i$. Taking the gradient wrt $\mathbf{x}$ and setting to 0, we get $\mathbf{c} + \mathbf{A^T a} = \mathbf{0}$.

Thus we just have to maximise $L(\mathbf{x}, \mathbf{a}) = -\mathbf{a^T b}$ as a function of $\mathbf{a}$ under the restriction that $a_i \geq 0$.