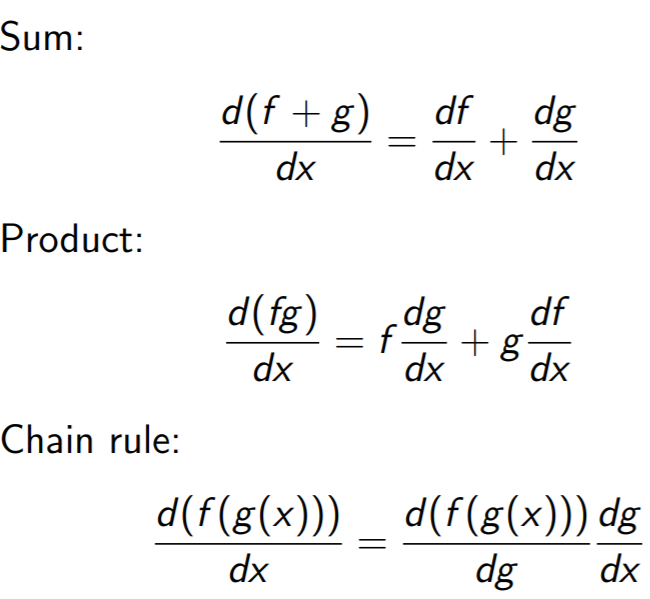
# Multivariable Calculus Differentiation Rules

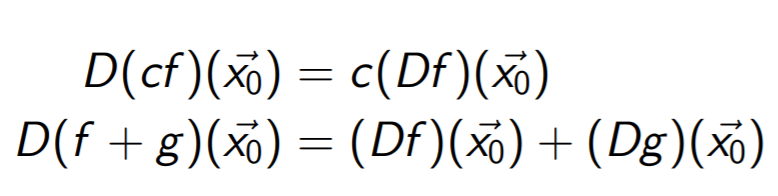
We already know the differentiation rules for univariable Calculus as they are the following:

Additionally, we also have the following rule:

Now, we are going to see the derivation rules for Multivariable Calculus as well.



The first two rules we are going to see are the following:

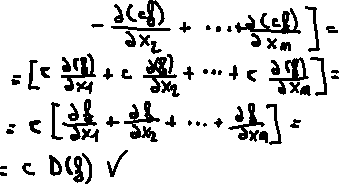
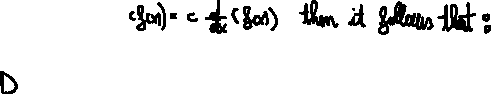
As we can see, the two rules above are analogous to the rules in univariable calculus.

Indeed, the first rule says that the gradient of a function cf is equal to the gradient of the function f multiplied by the scalar c. Remember that the gradient of a function is a matrix containing its partial derivatives.

The second rule says that the gradient of a function h = f + g is equal to the gradient of the function f + the gradient of the function g.

The proof of the first rule is the following:

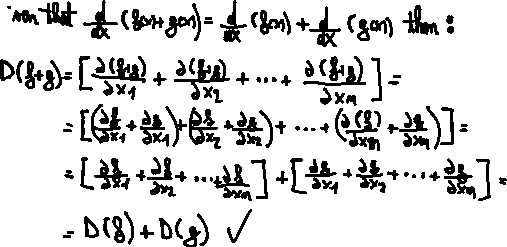
When we compute the partial derivatives of the function cf we are just using univariable calculus as we have shown in a previous lecture about partial derivatives. Therefore, we can use the rule of univariable calculus that I have added in the previous page as the following:



The proof of the second rule is the following:

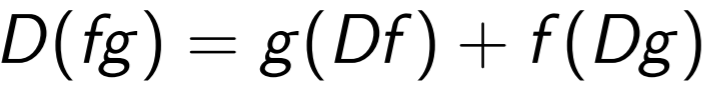
Always remember that partial derivatives obey to the derivation rules of univariable calculus.

Thus:



## Product Rule

The product rule in multivariable calculus is analogous to the product rule in univariable calculus but given that the function f is defined in the following way: f : Rn → R



In other words, the rule says that the gradient of the function fg is equal to the gradient of the function f multiplied by g (scalar) + the gradient of the function g multiplied by the function f (scalar).

The proof of the product rule for multivariable calculus is the following:



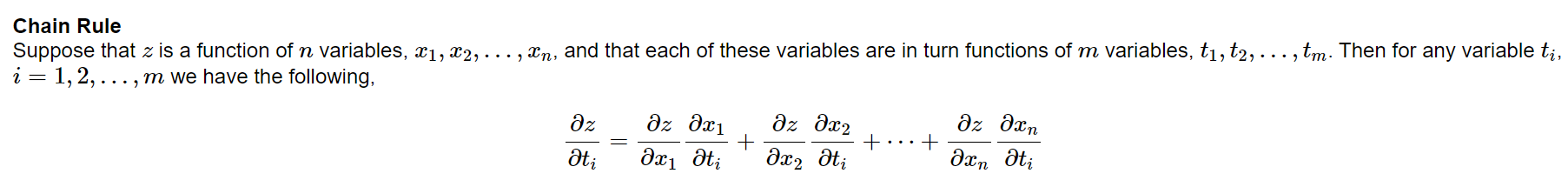
## Multivariable Calculus Chain Rule

The chain rule for multivariable calculus is analogous to the chain rule for univariable calculus.

The chain rule for the univariable calculus is the following:

Instead, the multivariable calculus chain rule is the following:





As we can see the chain rule for multivariable calculus is analogous to the one for univariable calculus with the only exception that there are more variables involved.

## Directional Derivative

Now we are going to introduce the important concept of directional derivative.

A directional derivative is the rate of change of a multivariable function in a given direction.

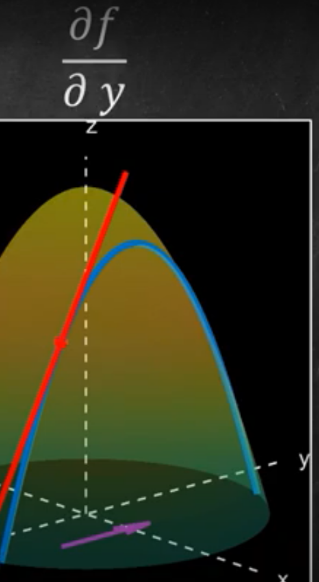
It goes without saying that we can easily represents directions by using vectors and so every time we want to compute the directional derivative of a function in a given point towards a given direction we need to use a vector to represent such a direction.

It turns out that partial derivatives are directional derivatives as well as we will illustrate in the following:

Let f(x,y) be a 3D functional. When we compute the partial derivative of f(x,y) with respect to x, we keep y fixed and let only x change. Therefore, given that we are in a point (x0,y0) we are actually computing the rate of change of the function towards the direction described by the following vector [1,0].

Given that only x varies then we actually obtain a 2D function which is the following g(x0+1s,y0+0s) = g(x0+s,y0) and this function describes where we can go if we starts at (x0,y0) and follow the direction or opposite direction of the vector [1,0] and now it is not anymore a function of x and y but just a function in terms of s.

The following pic provided an example of what 2D function we obtain when we find the partial derivative of a function with respect to y.



Given that we have a point (x0,y0), a function f(x,y), we can obtain a 2D function by using the direction of the vector =(a,b) in the following way:

* x(s) = x0 + sa
* y(s) = y0 + sb
* Then f becomes a function of two functions f(x(s),y(s)).

In order to compute the slope of the above function we just need to apply the chain rule that we have introduced before and so we obtain:



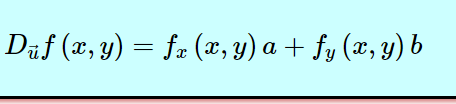
Thus, we have demonstrated that the directional derivative is just the gradient of the function (dot product) the vector that describes the direction.

In order to compute correct results we need to plug in the gradient of the function a point (x,y) which lies in the function and so the point (x0,y0) is one of these.

It is common practice that when we want to se the directional derivative and we provide the direction vector, this must be a unit vector.

## More formality in the directional derivative

The directional derivative of a function f(x,y) in the direction of the vector u is the following:



More generally, given a function f and a vector u the directional derivative can be expressed as:

This latter geometric definition of the dot product tells us that given a direction vector u, we will obtain the steepest increase in the function f when we choose a point **a** in the function which after being plugged in the gradient vector, the latter becomes parallel to the direction vector u.



To sum up, the directional derivative of a multivariable function f is just the dot product between the gradient of the function and the direction vector which is a unit vector.

For a greater understanding of directional derivatives, watch the following video: <https://www.youtube.com/watch?v=GJODOGq7cAY>

Seen it another way, given that we have function f at a point (x0,y0) then ▽f(x0,y0) represent its gradient vector. We know that in order to find the direction derivative of the point (x0,y0) towards a unit vector u then we need to perform the following dot product: ▽f(x0,y0) · u

Since u is a unit vector then the dot product will result in the maximum possible value when u is at 0° from the vector ▽f(x0,y0). In other words, u must have same direction of ▽f(x0,y0). From this we can infer that given that we are at a point (x0,y0) then we obtain the steepest increase in the function value towards the direction of the vector ▽f(x0,y0).

In addition we can also infer that we obtain the steepest descent when the unit vector direction is at the opposite direction of ▽f(x0,y0), that is at 180° from the vector ▽f(x0,y0) because cos 180° = -1. Thus, given that we are at a point (x0,y0) then we obtain the steepest decrease in the function value towards the direction of the vector -▽f(x0,y0).

## Gradient Descent

Finally, we can talk about the GRADIENT DESCENT ALGORITHM.

Gradient descent is a local iterative optimisation algorithm for finding a local minimum of a differentiable function from a given point in the function.

Very informally, the algorithm consists of starting from a point in a function and try to find the local minimum by following the steepest decreasing direction in the function value from that point. We have already shown that given a point in a function f then the steepest decreasing direction is given by the following direction: -▽f().

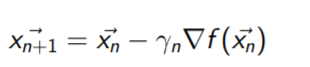
However, the tricky part is the following: How much distance do we have to do in the direction -▽f() ?

Thus the choice of the distance to do is very important because if we perform long distances then we could converge to the local minimum fast but at the same time we are more likely to miss the local minimum and might oscillate back and forth. If we perform short distances we are more likely to reach the local minimum but convergence will be slow.

It is easy to see that long distances are more likely to get us miss the local minimum because given that we start from a point x0 and we go towards the right direction for the local minimum and we arrive at x1, if x1 is ahead of the local minimum then when we perform another iteration of gradient descent we may go towards a direction d which is not towards the local minimum because from x1 the steepest decrease in the function is towards d.

From this, we can see that when using gradient descent we are not guaranteed to find the local minimum.

The iterative algorithm for gradient descent is the following:



In other words given that we are at a point then the deepest descent is in the direction of -▽f().

The ϒn is the step size I was talking above. ϒn is just a constant which when multiplied by the vector -▽f() will give as result a vector with the same direction but different magnitude. Naturally, ϒn cannot be a negative value otherwise we would swap the direction of -▽f() and we would go towards the steepest ascent.

is given by the difference of the vector (our current position) with the vector ϒn ▽f().

In other words, is the point that we reach when we follow the vector - ϒn ▽f() from the point .

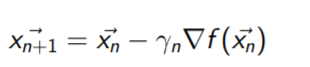
It is important to bear in mind that the choice of ϒ needs not be the same at each iteration.

Now, we are going to analyse a procedure called LINEAR SEARCH that will let us select at each iteration of the GRADIENT DESCENT algorithm, the best possible step size.

## Linear search

Given that gradient descent aims to find the local minimum of a function f then when finding from towards the direction -▽f(), we would like f() to be the smallest function value that we can encounter towards that direction.

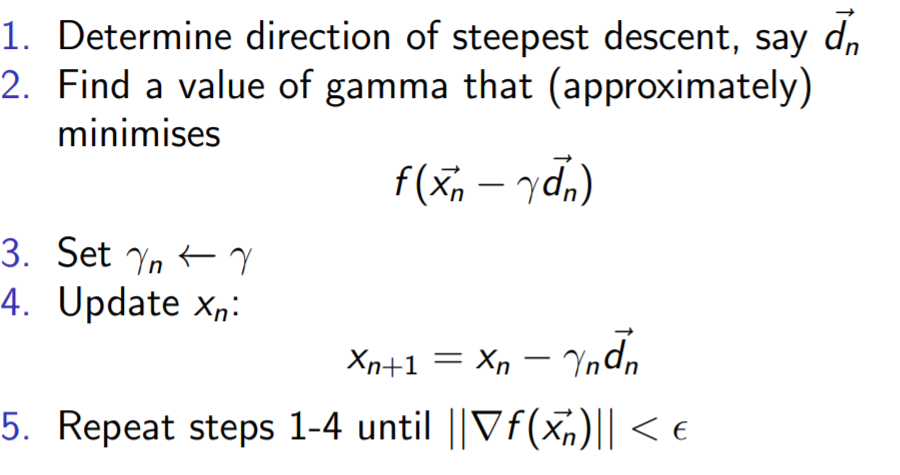
Given that



Then all we have to do is to find the value of ϒ for which the following function is approximately f() minimised. Thus, we could either find the global minimum or a local minimum of the function. This minimisation problem is just a univariable minimisation problem as the only variable involved is ϒ. We minimise the function f() because if there is really a local minimum in that direction of the function f) then this must necessarily be a local or global minimum of the function f(). However, the local or global minimum of f() is not guaranteed to be the local or global minimum of f).

Once, we have found the value for ϒ then we can apply gradient descent and iterate over and over again by finding at each step the value of ϒ that leads to the minimum function value from the current point following the steepest descending direction.

The GRADIENT DESCENT procedure which includes LINEAR SEARCH is the following:



The line 5 of the algorithm above is the stopping criteria of the algorithm. From calculus III, we know that if is a local extrema of a function f and f is differentiable at that point then ) = In other words, the gradient vector of the function at that point is the 0 vector.

Given that we use GRADIENT DESCENT for functions that are differentiable as otherwise we could not find out what the gradient of the function is and so the steepest descent from a point, it follows that the local minimum will have ) = .

Thus, if after having performed gradient descent we arrive at a point such that is approximately 0 then is approximately the 0 vector and so we can stop the algorithm and use the point as approximately our local minimum.

However, this stopping criterion is not necessarily right as we could stop at a point where ) = but that point not be a local minimum but instead a saddle point. Thus, we could stop when is approximately the 0 vector in a point which is close to a saddle point instead of a local minimum.

Given that we use exact local search we are guaranteed to never arrive at a local maximum as we compute the minimum value of the function f() with respect to ϒ.

However, given that we use gradient descent without linear search, we may get stuck even at a local maximum as we could go towards the steepest descending direction but travel too far towards that direction getting stuck in a local maximum.

It is important to notice that the algorithm gets stuck whenever encounters a critical point. A critical point may be a relative minimum, relative maximum or saddle point.