Module 2

Reality is Hard

So far we have discussed how to think of 2D functions as landscapes, and we have also seen that we can construct Jacobian vectors, which tell us both the direction and magnitude of the gradient at each point in space.

Lastly, we added one final clutch tool to our toolbox, which allowed us to tackle deck, what kind of feature we were standing on; when we landed on a point with 0 gradient.

These concepts will all be very useful to develop my understanding of optimization problems.

And also Peter says that multivariate Calculus is worth knowing.

However in this session, we going to remind ourselves about two features of real systems, which so far we have avoided.

Firstly for many applications of optimization such as in the training of neural networks, you going to be dealing with a lot more than 2 dimensions, potentially hundreds or thousands of dimensions

This means we can no longer draw our nice surface, all of the same maths still applies, but we now have to use our 2D intuition to guide us, and enable us to trust the maths

Secondly, as we mentioned, briefly before, even if you do just have a 2D problem, very often we may not have a nice analytical function to describe it, and calculating each point could be very expensive.

Even though in principal a plot could possibly be drawn, you would not be able to afford either a super computer time or laboratory staff to possibly populate this thing.

Thirdly, all the lovely functions we have dealt with so far, were smooth and well behaved, &

However, what if our function contains a sharp feature, like a discontinuity. This would certainly make navigating in Sandpit a bit more confusing.

Lastly there are a variety of factors that may result in a function being noisy, which I am sure you can imagine, might make our Jacobian vectors pretty useless, unless we careful.

This brings us to the second topic in the lesson

Question: If we don't ever have the function we are trying to optimize, how on earth are we suppose to build the Jacobian out of the partial derivative?

This leads us to another area of research called numerical methods

There are many problems that either do not have a nice explicit formula for the answer or do have a formula, but solving it directly, would take a lifetime.

To fight back against the universe
Mocking us in this way, we have developed
a range of techniques that allow us
to generate approximate solutions

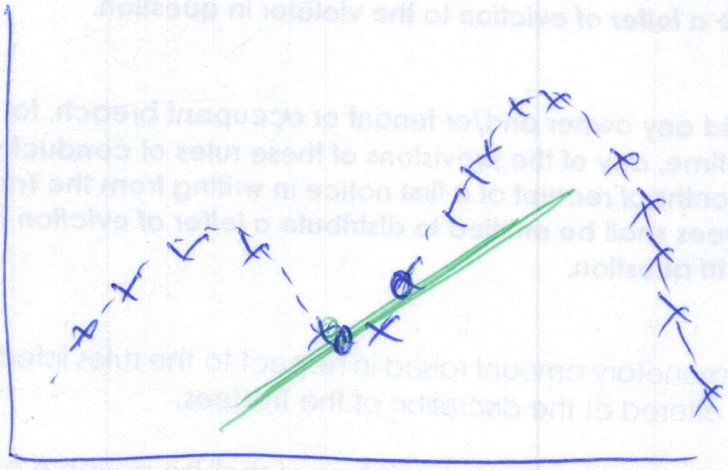
one particular approach that is relevant to
our discussion to ~~of~~ the Jacobian, actually
takes us right back to the first few
lessons of this course, where we defined
the derivative
we started by using an approximation, based
on rise over run, calculated over a small
interval

gradient at $x \approx \left(\frac{f(x+\Delta x) - fx}{\Delta x}\right)$

and then looked at what happens, when this interval approached 0

all that we doing, with the finite difference method, is accepting that we not going to work at the value of the gradient at every single point in space.

We just going to use the points, that we do have, and build an approximation for the gradient based on that.



In the example Shaun has we have already calculated lots of points on this one dimensional function.

But Clearly this is not practical for higher dimensional scenarios.

So, all we do is take this logic one step further and say, If I start with an initial location, and we would like to approximate the function, we would simply approximate each partial derivative in turn.

So taking a small step in x, allows us to calculate approximate partial derivative in x And small step in y, gives us an approximate partial derivative in y.

$$J = \left[\frac{f(x+\Delta x, y) - f(x,y)}{\Delta x}, \frac{f(x, y+\Delta y) - f(x,y)}{\Delta y} \right]$$

Two things to bear in mind here, ~~how~~ how big should our little steps be, ~~will~~ Well this have to be a balance

If α is too big, you make a bad approximation, for obvious reasons.
But if it's too small, we may run into some numerical issues
Just remember, when the computer calculates the value of the function at a point, it only stores it to a certain number of significant figures
So if your point is too close, your computer may not register any change at all!
Second, what happens if the data is a bit noisy
To deal with this case, many approaches have been developed.
Perhaps the simplest is to calculate the gradient using a few different step sizes

And take some kind of average.

∴ once we leave the world of nice smooth functions, and enter the real world of noisy data and computational expensive functions, things start to get a lot more interesting.