Aerodynamics, F1 in schools and Derivative Free Optimisation

Ollie Higgins

Introduction

This year I was the head of engineering at the F1 in school team, RedeagleRacing. One of my main responsibilities was aerodynamic optimisation. This can be done in 2 ways: macro and micro. Macro optimisation is about studying the airflow as a whole, whilst micro is about optimising every individual surface.

The problem I faced with micro-optimisation was finding a way to optimise a set of dimensions in order to minimise drag. We can say $drag = f(z)$ where z is the set of dimensions to optimise. However, we didn't know what that function was, we didn't have gradient information, each sample took at least a minute to calculate in flow simulation, and z was multidimensional.

Note that the only goal was to minimise drag, since downforce was not needed. The car was also built around a specific set of rules. The aim was to design a 48g, gas cannister powered car to complete a 20m track as quickly as possible (around 70km/h). Full set of rules : rules.

Approximation

Each dimension's domain is linear mapped between 0 and 1.

$$m(x) = \frac{x - min}{\max - min}$$

The first step is to approximate the function $A(z)$. I used a radial basis function to do this. $A(z) = \sum w_r * K(|z - z_r|)$

Where $K(D)$ is the RBF kernel, z is the input, w_r is the sample weight and z_r is the sample location. (see more of kernels later).

Given N samples, N equations can be formed. Here is an example with 3 samples:

$$A(z_1) = w_1 * K(|z_1 - z_1|) + w_2 * K(|z_1 - z_2|) + w_3 * K(|z_1 - z_3|)$$
$$A(z_2) = w_1 * K(|z_2 - z_1|) + w_2 * K(|z_2 - z_2|) + w_3 * K(|z_2 - z_3|)$$
$$A(z_3) = w_1 * K(|z_3 - z_1|) + w_2 * K(|z_3 - z_2|) + w_3 * K(|z_3 - z_3|)$$

$$\begin{bmatrix} K(|z_1 - z_1|) & K(|z_1 - z_2|) & K(|z_1 - z_3|) \\ K(|z_2 - z_1|) & K(|z_2 - z_2|) & K(|z_2 - z_3|) \\ K(|z_3 - z_1|) & K(|z_3 - z_2|) & K(|z_3 - z_3|) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} A(z_1) \\ A(z_2) \\ A(z_3) \end{bmatrix}$$

These set of equations can then be solved. I used gaussian elimination to do this, although I'm not going to explain how that works here.

Uncertainty

Uncertainty at z: $U(z)$ can also be calculated.

$$U(z) = \frac{1}{\sum |z - z_r|^{-2}}$$

Although this value of uncertainty has no mathematical basis, it is inspired by the resistance of parallel resistors. Uncertainty will be 0 at a sample point and will grow the further away from samples.

We treated the value at z as being normally distributed with a mean of A(z), and standard deviation U(z): f(z) $\sim N(A(z), U(z)^2)$

This method for calculating uncertainty only considers the proximity of samples to z. Another factor which could be considered is the variance of samples around z. If the samples are close, but very different, uncertainty would increase.

Sampling

The next sample point is the point most likely to be below an adaptive target value.

$$Target = current\ minimum - temperature$$

The current minimum is the smallest value of the function currently found. The temperature is a decaying value. Whilst temperature is high, the algorithm favours exploration, and when low, exploitation. This was inspired by a similar method used in another algorithm called Simulated Annealing.

The point most likely to be below the target, is the point the lowest number of standard deviations below the target:

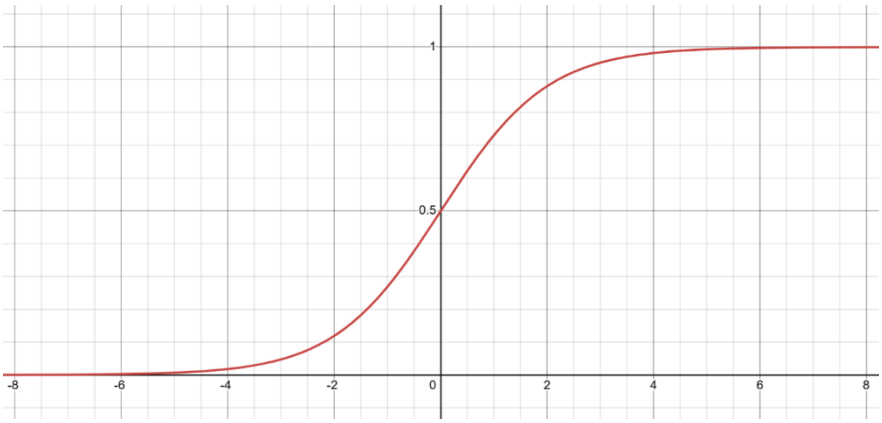
$$f(z) = \frac{A(z) - target}{U(z)}$$

The new problem is minimising $f(z)$ to find the next sample point. This sounds like the same problem that we started with. The 2 key differences are that A: $f(z)$ is instantaneous to calculate, and B: gradient information can be calculated. This means more advanced algorithms, that can make use of gradient information and large number of samples, can be used.

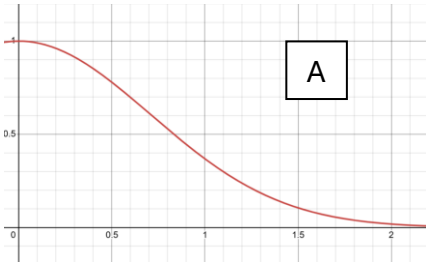
Considering taking a sample from the actual function takes a long time, we can say that this secondary optimisation is insignificant to the complexity or efficiency of the overall algorithm. This is especially true in lower dimensions.

Instead of minimising $f(z)$ it is better to minimise $g(z)$, where:

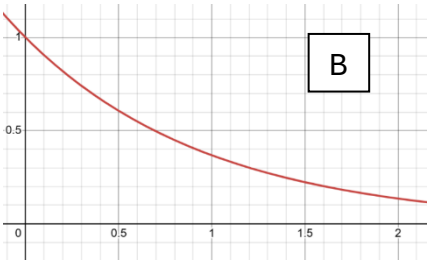
$$g(z) = \frac{1}{1 + e^{-f(z)}}$$



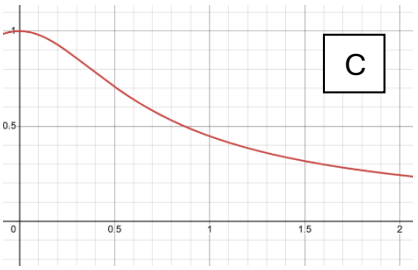
Kernel functions.



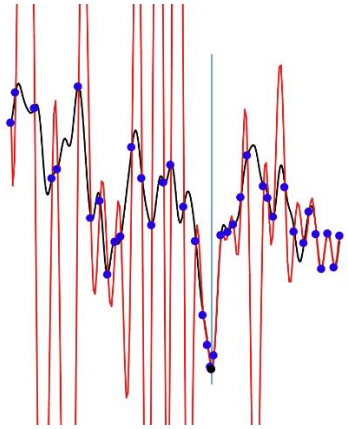
$$y = e^{-x^2}$$



$$y = e^{-x}$$



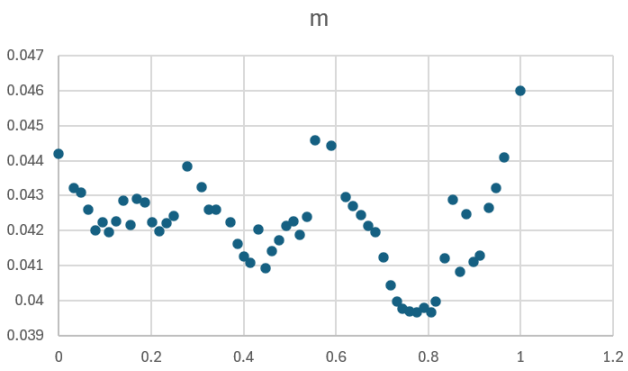
$$y = \frac{1}{\sqrt{1+(ax)^2}}$$



Using smoother easing kernels (A - above) often causes large amounts of overfitting. Left image (red – approximation, black – actual, blue – samples). This can be solved by using a sharper kernel (B - above), but this sacrifices function smoothness. Although I haven't found a robust solution to this, using a kernel such as (C - above) can fix this. The value "a" can be adjusted per sample, with smaller values creating a sharper function. To automate this, samples that are closer together will favour sharper kernels.

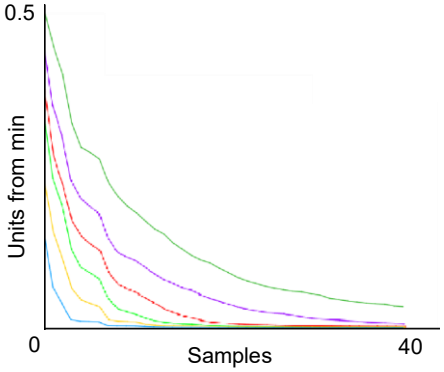
Limitations

As the number of dimensions increases, the amount of space needed to be searched increases. If there are x points in a $1unit^n$ space, the average space around each point will be $\frac{1}{x}$. Therefore, on average, each axis will cover $\frac{1}{\sqrt[n]{x}}$ units. Therefore, the order of the algorithm is to the Nth power. In higher dimensions. due to the increasing space needed to be searched, exploration is favoured. I found that above 5 dimensions the algorithm starts to struggle. Although one of the key factors that was causing this was my secondary optimisation algorithm, which I had limited time to work on. I explain how I overcome this in the application section.



Function samples might have uncertainty in their values. In my case this was due to the chaotic nature of the flow simulation. But if samples were taken experimentally, the same problem would occur. Since the approximation algorithm I used, goes through all sample points, this can cause overfitting. Overfitting will greatly increase the number of local minima to be searched by the algorithm, in turn reducing its efficiency. If this effect is minimal, the problem can be solved by adding a base value to the uncertainty. To fully fix the issue, a different Approximation algorithm would be needed.

Convergence

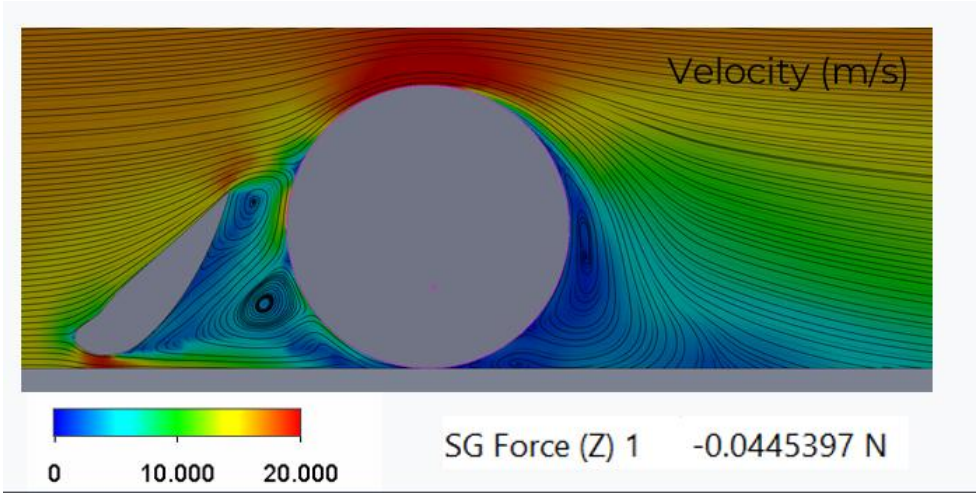


As mentioned above, the convergence rate is slower in higher dimensions, or more noisy functions. To show the effect of noise, I varied the number of octaves of a test function made of sin waves. $f(x) = \sum sin(x * A_i + B) * C_i$ Where, in higher octaves, A is larger, and C is smaller. This test was run in 1D and results were averaged over 1000 tests. Convergence (units from min) is on average inversely proportional to number of samples.

colour	octaves
	2
	5
	10
	20
	100
	10 + noise

When increasing the number of octaves, or adding random noise when sampling the function, convergence is slowed.

Application



Although the application is less mathematically focused, it helps to highlight the effectiveness and drawbacks of the algorithm.

When using the algorithm to optimise my car, often the number of dimensions were greater than 5 for a specific area I was working on. Running an initial iteration of optimisation on all the dimensions was a good starting point. This would help establish each domain before further optimising. Data was exported into excel, and by looking at the top 10 samples, the dimensions which greatest affected drag could be found. These dimensions would then be optimised, for example in a group of 3. Finally, each dimension would be optimised 1 at a time (coordinate descent). By splitting up the dimensions into smaller groups, optimising, then manually adjusting the domains, helped the algorithm exploit more promising areas.

For example, while optimising the front wing, there were initially, 12 dimensions. I reduced this by re modelling the wing using a 4-series NACA aerofoil. This reduced the number of dimensions to 7. An initial iteration showed that the angle of attack, height, camber, and x position were most influential on drag, and so were then optimised.

Conclusion

The main improvements I would make, would be to the approximation algorithm, and the uncertainty calculations as these struggled to handle very noise functions. For example, using Bayesian optimisation, something I didn't have time to research into.

Using this algorithm, I reduced drag by 20% from the previous year. Given that the previous year's car was already optimised, and most of the changes were only small changes to surface profiles, this was a significant improvement.

We won fastest car at our regionals and progressed to nationals. Whilst our car looked competitive in its first run, and if every team had raced once, we would have won, our rear wing broke and that was pretty much our competition over. We placed 10th fasted legally.

So, whilst a large focus was placed on the aerodynamics, we lacked the funding to do proper physical testing with a track.

Thank you for reading.

References

Radial basis function - Radial basis function - Wikipedia

Simulated annealing - Simulated annealing - Wikipedia