Report 1: Line Search

Rob Rau

January 29, 2014

Introduction

Two different line search methods were used to optimize the simple drag coefficient equation

$$C_D = kC_f\frac{S_w}{S} + \frac{C_L^2}{\pi ARe} \tag{1.01}$$

Where k is the form factor that accounts for pressure drag, C_f is the skin friction coefficient, S_w is the wetted surface area of the aircraft, S is the reference planform area, C_L is the lift coefficient, AR is the aspect ratio, and e is the Oswald span efficiency factor. This equation can be modified to be written in terms of AR using the following relations

$$C_f = \frac{0.074}{Re} \tag{1.02}$$

$$Re = \frac{\rho V c}{\mu} \tag{1.03}$$

Where Re is the Reynolds number, V is the aircraft velocity, and c is some characteristic length that the flow follows, in this case, the chord of the wing. The aspect ratio is non-dimensional number that is expressed in terms of the wing span b and and reference planform area.

$$AR = \frac{b^2}{S} \tag{1.04}$$

With these relations and by holding the values of V, S, S_w, ρ, μ, k, and e constant eq. (1.01) can be written as a function the aspect ratio.

$$C_D = k\frac{0.074}{\left(\frac{\rho V S\sqrt{SAR}}{\mu}\right)^{0.2}}\frac{S_w}{S} + \frac{C_L^2}{\pi ARe} \tag{1.05}$$

The goal of the optimizer is to find the aspect ratio, that gives the lowest drag.

Both optimizers were ran in a number of different test cases. Each test case had a different starting point and interval in order to observe how these parameters affect convergence and to compare the two optimizers.

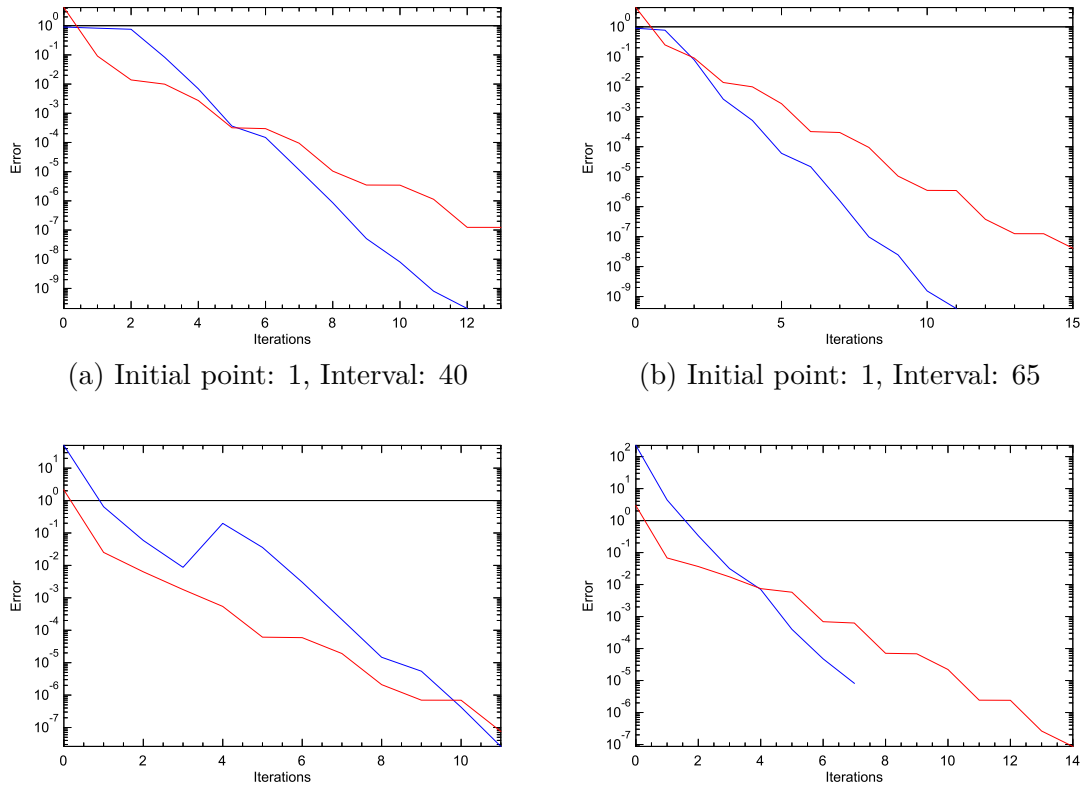
The golden section search works by taking an interval and initial point and reduces the interval until an optimal solution is found. The interior points of the interval are chosen in

such a way that when the interval is reduced, only one more function evaluation is needed before reducing again.

The final optimized value is an aspect ratio of 28.3818 with a drag coefficient of 0.0115725.

Convergence

Both the golden section search and the bracket and zoom algorithm exhibit linear convergence. However, the bracket and zoom algorithm has a higher rate of convergence as seen in fig. 1.



(a) Initial point: 1, Interval: 40

(b) Initial point: 1, Interval: 65

(c) Initial point: 10, Interval: 20

(d) Initial point: 20, Interval: 90

Figure 1: Comparison of convergence rates for the golden section search (red) and the bracket and zoom algorithm (blue)

This general trend seems to be independent of start point and interval. I did not have enough time to compute the exact convergence rate.

Performance

Even though the bracket and zoom algorithm has a slightly higher convergence rate, it consumes more computation time per iteration than the golden section search. The golden section search, on average, consume 1.35 micro seconds of cpu time per iteration where the bracket and zoom algorithm takes about 1.5 micro seconds per iteration. These test were tested using optimization level 3.

While both algorithms only require one function evaluation the bracket and zoom algorithm requires more conditional logic. Logical statements requires branching on the CPU. Moving between execution branches can be more costly than just mathematical statements alone. This branching problem can be exacerbated if there is no regular pattern in the way the program branches thus fooling the CPU's branch predictor, and generating cache misses. Cache misses can severely slow down code execution. This is only a potential issue, and not what I think is happening in this case, but a cautionary tail none the less. In comparison the golden section search has far fewer conditionals and is more mathematical in nature.

It would be interesting to try this code with a different compiler and see if there are any performance differences. The GNU implementation of the D Programming language could benefit from the mature backend code generation optimizations.

Other Thoughts and Observations

Starting each algorithm at different points and using different intervals brought a few things to my attention. The golden section search is limited by the start point and the interval given to it. It cannot decide in which direction to apply that interval, so if the start point is past the optimal point, it will converge on the best local minimum in the interval. The bracket and zoom algorithm doesn't have this problem as it computes the gradient to chose which direction to apply the search interval.

Although it is obvious now, I had thought I had a bug in my bracket and zoom algorithm when running case that started past the physically optimal point, and had a large interval. The optimizer was reporting a negative optimal aspect ratio. This puzzled me until I realized that the drag coefficient was indeed lower at the non-physical solution. This is why constraints will be added later.

Code

The algorithms as they are right now do not support multiple design variables, but the way the underlying framework has been built makes it trivial to extend this functionality. The base objects and interfaces in the Multidisciplinary Design and Optimization Library

(MDOL) are all built to support multiple design variables so the only changes that need to be made are in the specific implementations.

So far the choice of the D Programming language has led to very clean code and an easy to understand and extensible framework. This isn't to say there have been no learning curves or unforeseen issues. When implementing the complex step to use the bracket and zoom algorithm, I was testing it on the test polynomial seen in the notes

$$f(x) = (x-3)x^3(x-6)^4 \tag{5.06}$$

it would incorrectly compute the derivative at certain points. I found that when the step size was set to small ($< 10e-10$), the $(x-6)^4$ portion of the equation would have error in the imaginary portion. Setting the step size to $10e-10$ fixed this, but I will be continually verifying this to be sure.