An Evolutionary Computation Method for Initial Step Length Selection to Improve Run Time of a Line Search Optimization Algorithm

Abstract

Line search algorithms typically employ fairly simplistic strategies for initial step length selection that is subsequently improved upon such that it satisfies Wolfe conditions. We think evolutionary computation will help especially if the user of the algorithm is naïve about the mathematical properties of the function and as a result chooses the seed solution for the algorithm at a fair distance from the optima. We think this is not all that a bad assumption to have in software development research especially in the applied sciences. We report improved strategy performance for the Booth function.

Literature Review

An unconstrained optimization algorithm popularly uses the iterative Line Search algorithm which sets a starting point for the step length and subsequently computes one that satisfies Wolfe conditions. A successful program converges with a pre-fixed tolerance. A popular strategy in Nocedal and Wright (2006) hereafter referred to as NW uses steepest descent, is easy to implement and is fairly intuitive: InitialStepLength(iteration) = WolfeStepLength(iteration-1)*Gradient(iteration-1)*UnitDescent(iteration-1)/ Gradient(iteration)*UnitDescent(iteration). This strategy is hereafter referred to as S1. Another algorithm in NW hereafter referred to as A1 chooses 0 as the starting step length for each iteration. We use a combination of both the strategies to improve run time.

Our Proposed Algorithm Strategy

Only for the first iteration we use A1's initial step length. For subsequent iterations we use a modified version of S1. We bracket the computation of the step length within a bracket and draw values from this bracket that will satisfy Wolfe conditions. For the first iteration, we begin with an initial step length of zero, compute the step length that satisfies Wolfe conditions and modify it using a seed multiplier to estimate an initial step length for the next iteration. Based on the step length computed in the next iteration we revise the multiplier either upwards or downwards using evolutionary computation.

Results and Conclusion

We use the 2-dimensional Booth function to test our algorithm's performance. The minimizer of the function is (1,3) and f(1,3) = 0: f(x,y) = (x+2y-7)^2+(2x+y-5)^2. A cursory look at the function makes it evident that the function reaches its minimum value when the expressions within the bracket are set to 0 and solved simultaneously. We chose this function because users typically treat the optimization algorithm as a black box that pops out a solution and even such cursory observations are routinely not made. Our algorithm converges in 18 iterations and improves upon A1 which converges in 23 iterations but we report a higher error of 0.58% vs A1's 0.34%. The S1 algorithm doesn't converge even after 50 iterations. We choose the seed solution at (7,9) to demonstrate the power of our algorithm. Going forward, we will look to improve run time further using advanced numerical linear algebra as we implement the algorithm in a professional software.

References

Global Optimization Test Problems. Retrieved June 2013, from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm.

Nocedal, J and Wright, S.J. 2006. Numerical Optimization. Springer.

[Type here]

Wolfe Conditions. Retrieved 2021, from https://pages.mtu.edu/~msgocken/ma5630spring2003/lectures/lines/lines/node3.html