**Abstract**

I develop several different optimization (viz. minimization) algorithms and then test them against two different problems from various starting points to find their strengths and weaknesses. I observe that in general there is a trade-off between robustness of the algorithm and program complexity.

**Method**

Using MATLAB 2016a, I wrote scripts comparing MATLAB’s native optimisation function (fminsearch), Newton’s method, Broyden-Fletcher-Golfarb-Shanno (hereafter BFGS), Steepest descent, Conjugate gradient search, and Nelder-Mead search methods for two different problems:

* c(x,y) = 7.9 + 0.13x + 0.21y – 0.05x2 – 0.016y2 – 0.007xy (which is well-behaved)
  + from starting points (-9, 1), (-8, 18), (1, 19) and (9, 10)
* Rosenbrock’s function, viz. f(x,y) = 100(y - x2)2 + (1 – x)2
  + from starting points (-1, 1), (0, 1) and (2, 1)

The first is roughly a ‘trough’ shape, whereas the second is a well-known problem for testing optimization algorithms: <https://en.wikipedia.org/wiki/Rosenbrock_function>

For each algorithm and starting point, iterations were each counted and reported, and then graphed against a contour map of the problem function. Details of the algorithms and their performance can be found in the attached MATLAB scripts.

**Findings**

For the both problems, Newton’s method is clearly the ‘best’, always converging and with the minimum of iterations regardless of starting point. However, it does require the most knowledge about the problem involved, and is the most restrictive of methods in its requirements. (Namely, a Hessian matrix must be constructable for the problem, and known beforehand.)

Again for the well-behaved problem, all the algorithms converged on the correct answer with more or less alacrity. The primary differences were number of iterations, total running time (which I did not measure), and implementation difficulty. Nor did I discover any anomalous starting locations, i.e. starting points that would cause the algorithm to misbehave. Further, and unsurprisingly, Nelder-Mead took the most iterations (around 25) by a significant (though not huge) margin, followed closely by fminsearch (which has some native optimizations that make it faster). Of note, the conjugate gradient method took significantly fewer iterations than steepest descent, rivalling BFGS in that regard.

As mentioned above, Newton’s method works for both of these problems regardless of starting point. (I tried points such as (100,100), (pi, -pi/2) as well as those listed in the problem.) However, it should be stated *again* that it requires deep knowledge of the problem, whereas all other methods construct that knowledge programmatically. Further, it is entirely possible (depending on the problem architecture) for the required Hessian matrix to become too close to singular to allow resolution of the method.

With an error calculation based either on the difference in function value or difference in BFGS demonstrates starting point dependency. Starting at (-1, 1) it eventually converges on the correct minimum, though it takes a large number of iterations. At other starting points it manages to find the trough, but it fails to converge on the global minimum.

However, if instead you calculate your error based on the difference between the Hessian approximations for one guess to another (so, from B\_k to B\_k+1) the BFGS method converges for all tested solution points. Nevertheless, it loses a lot of its ‘fast-ness’ for a complicated function like Rosenbrock, such that the iterations taken begin to rival the Nelder-Mead method. Further, it still shows a kind of ‘soft’ starting-point dependency; certain starting points can take *a lot* of iterations. (Example: for Rosenbrock’s function, starting at [-1, 1] it takes 168 iterations.)

There are a few qualifications to this as well. Though to a lesser degree than Newton’s method, it does still rely on the possibility of a Hessian matrix, albeit not a specific one, nor one known beforehand. Also, the points chosen by the method are not according to a predictable-at-a-glance path, so that both location (and error, calculated any which way) can swing widely from iteration to iteration.

Steepest descent shows even greater starting point dependency, quickly finding the minimum at one starting point and requiring a startlingly-large number of iterations to converge to the minimum at others This is probably because the use of the gradient requires perpendicular movements, and the Rosenbrock function quickly forms a narrow trough, thereby making the movements possible only very small.

Conjugate gradient, as an algorithm intended to improve steepest descent, performs as expected in that role. It takes fewer iterations (except at the first point, which only takes both algorithms 2 steps). It really shines well for the second point, where steepest descent takes a staggering ~5,000 iterations to converge to the minimum, whereas the conjugate gradient method takes ~2,000.

Interestingly, for both steepest descent and conjugate gradient, the algorithm can become a fair bit more accurate with intelligent choice of the minimization parameter (α), but at the same time unintelligent choice easily leads to lack of resolution as the algorithm vacillates back and forth between answers.

Nelder-Mead, as mentioned above, converges on the correct minimum (that is, f(x,y) = 0 at (1,1)) for all starting points tested. Further, the mean number of iterations is 53, with little variation in that number (though large variation in the path). Of interest, however, is that, while starting point doesn’t seem to matter, *for both problems* how you choose the points of your starting simplex does matter; without care, you can choose points that will break the algorithm.

**Conclusion**

Correct, efficient solutions to optimization problems require some anterior knowledge about the nature of the problem to be solved. Without such foreknowledge, it is entirely possible that a ‘more efficient solution method’ will either not work at all, or worse, give incorrect or misleading results. Sometimes this can be overcome if the tolerances of your solution allow (such as not needing the absolute minimum, but rather just some low value), but even that can be unpredictable. Furthermore, there appears to be a tradeoff between algorithm robustness and number of iterations (which is a weak proxy of computing time), though even that appearance can be deceptive, since for more complicated or less well-behaved problems a more robust and well-designed algorithm may pay off in fewer iterations, since it doesn’t flail around as much.

Further, depending on the problem algorithms may exhibit starting point dependency, quickly spiking the necessary iterations to complete the optimization.

Ultimately, the choice of method is ultimately dependent both on the problem and the intuition of the researcher. BFGS offers a good middle-ground for the problems tested, but Nelder-Mead appears to be the most robust. Newton is the fastest by far, but requires both an intricate knowledge of the problem and a bunch of previous calculation that none of the other methods require and may not be available for real problems. Conjugate gradient and steepest descent show as decent middle-grounds, requiring not even an approximate Hessian matrix and significantly fewer iterations (or complexity) than a simplex search, but also require the problem to be relatively ‘homogenous’, i.e. problems of a sort that directionality of the solution is not a big factor. Specifically, steepest descent and conjugate gradient work best for problems without regions of great gradient variation. Finally, the Nelder-Mead simplex search appears not to be influenced much by the nature of the problem, making it a good choice when the properties of the problem are ill-known.