Notes on non-linear minimization using Simplex methods

Notice: the explanations on the minimization function development are high level approach discussions. Effectively, these are my working notes and are not particularly well organized or grouped. The code has more detailed information on each section. Readers are welcome to pose questions though response time may be rather slow.

Included in this study are two function minimizers based on the Simplex method (simplexmin and minfun). Both solvers use the Simplex method based on the formula presented in: "Elements of Structural Optimization" R. Haftka / Z. Gurdal - pg 124 – 127. Note: while I have not tried to reverse engineer the simplex method used in fminsearch, based on function performance, there do appear to be differences in how shrinkage and contractions are handled between fminsearch and this approach.

The purpose for creating these minimizers was (in order of precedence):

1. I was curious if the decision / minimization process could be improved with respect to local minimum rejection
   1. Generally, the n dimension minimization solutions employ either Powell’s conjugate direction or Simplex
   2. I had tried the conjugate direction (steepest decent) before and did not like how it ignored slope changes (gradients) until it had reached a minimum along the given vector
   3. I had not developed a Simplex method
2. What would be the trade-off between computational efficiency and “robustness” with respect to rejecting local minimum conditions
   1. Could one even know that a local minimum was driving the solution?
   2. Could one build in checks for alternate directions to a better solution?
   3. How robust could the process be?
3. Numerical methods are just interesting to me and I wanted to learn how to implement the simplex approach
4. In the past I had need for a constrained minimizer and did not have a suitable minimization function (I never really liked the conjugate performance). While I no longer need this minimizer, it remained an open question in my mind that I just had to investigate.

The minimizers here will handle:

1. Linear equalities
2. Linear inequalities
3. Nonlinear equalities
4. Nonlinear inequalities
5. Lower / upper bounds
6. Any combination of a linear, nonlinear, or bound
7. Yes, fmincon will handle these as well, but that relies on users having the optimization toolbox

The simplexmin function is mostly an implementation of the approach outlined in R. Haftka / Z. Gurdal with constraints (linear and nonlinear) added as penalties to the function and hard limits on the input vector based on the bounds. The cost function penalties are soft penalties and can most certainly be improved. The ones used here (exponential growth with the magnitude of the violation) were implemented for speed and simplicity.

The implementation of the hard bounds produced an interesting problem where some pivots have a zero change in function value. This occurred when the initial point was on a bound and the gradient stepped crossed the bound. The limit correction kicked the next step back across the bound and, in some cases, resulted in the initial point being repeated. This made the solution look insensitive to that particular input and the resulting solution process was negatively affected. Similar issues could occur with the penalties were a significant step “over” the linear / nonlinear constraint could result the gradient appearing to be inverted (quite agressively in some cases) rather than headed toward the constraint boundary. In that case, the solution would “wander” around and usually not find the best solution.

As the simplex solution method progresses and the solver gets closer to a minimum, the test points tend to cluster which means that the next step tends to be smaller in magnitude that the last step. If a bound or penalty is hit once the solution space has contracted, it is less likely an aggressive step will be made to reach a better minimum. Without some major change to the Simplex theory, the only solution was to force a re-assessment of the solution after the local optimum had been found to test for a local minimum (and that is not a sufficient criteria to ensure a global minimum was reached).

The published Simplex approach I referenced was for an unconstrained, unbounded problem. Adding the bounds / constraints noted above forced a modification to the approach to attempt to look for an alternate minimum. The rather simple solution was to put a loop around the Simplex solver and re-call the solution using the prior attempts best result until no improvements were observed. As each re-call starts with a step size based on the default step criteria, a second solution test might be able to step out of a local minimum when the prior solution could not.

This post convergence retesting of the Simplex results adds something on the order of 10% to 30% more function calls than the single call approach for “smooth” functions. Basically, the minimum found was the true minimum and the retest just finds the same solution again. For functions with local minima, the number of function calls can go up significantly with this approach, but it usually does return a better solution for that effort. Finally, as is generally the case with function minimization, initial conditions have a significant effect on performance.

The Simplex re-calling modification noted above led to a second observation that the default step criteria (alpha, beta, gamma, eta, gs) can also have a significant effect on the solution when there are multiple local minimums. For “smooth” functions without local minim, the selection of the default step sizes is not as important. But locally complicated functions can be quite sensitive to the initial step sizes selected. From this observation, the minfun solver was generated which took the Simplex solver above and wrapped it in a secondary solver to define the best initial step sizes. Essentially, it uses brute force to re-try the problem with multiple initial step sizes with tuning of the initial step sizes based on prior performance. The lowest cost result is kept.

Included in this package is a test function “testmin.m”. Interested readers are welcome to use it as a test platform. I recommend running the script a few lines at a time in debug mode. An excel file is included of the results obtained as well.

**Unconstrained problems (**[**https://en.wikipedia.org/wiki/Test\_functions\_for\_optimization**](https://en.wikipedia.org/wiki/Test_functions_for_optimization)**)**

As one would expect, for smooth unconstrained problems, the added checks to the simplex method did nothing but increase computational costs. Note that all three solvers got the same result.

|  |  |
| --- | --- |
|  |  |
|  |  |

Function evaluations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Sphere | Rosenbrok | Beale | Booth |
| Fminsearch | 73 | 154 | 104 | 97 |
| Simplexmin | 97 | 209 | 108 | 113 |
| Minfun | 721 | 549 | 379 | 428 |

For some functions, the ability to assess different step sizes was quite valuable even for smooth functions when the initial condition was not well placed. The Easom case is particularly interesting as the initial gradients were quite small leading to pre-mature exiting of the solver. For the Bukin case, the initial pass resulted in a gradient below the threshold at -7.5, 0.5. Restarting allowed the solver to find the rather small gradient along the fold.

|  |  |
| --- | --- |
|  |  |
|  |  |

Finally, the ability to test for local minima (simple retest using simplexmin and the more aggressive step size variation in minfun) was quite valuable for cases with complicated functions. Note that simplexmin usually was better than the reference and minfun was better than simplexmin.

|  |  |
| --- | --- |
|  |  |
|  |  |

However, the solution improvements did have a cost and sometimes the result did not improve though the cost did. For example, the simplex case for Styblinski did not improve the result over the reference and the minfun solution cost was 2 orders of magnitude greater.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Rastrigin | Rastrigin | Ackley | Styblinski |
| Fminsearch | 56 | 121 | 52 | 78 |
| Simplexmin | 136 | 470 | 138 | 147 |
| Minfun | 7608 | 41535 | 16388 | 19571 |

**Constrained problems (**[**https://en.wikipedia.org/wiki/Test\_functions\_for\_optimization**](https://en.wikipedia.org/wiki/Test_functions_for_optimization)**)**

In some instances the introduction of constraints did not affect the solvers much as the functions were still smooth (first row below). Never the less, the costs did increase with the inclusion of more checks. In others, the local rechecks helped (second row) though the solution for the Townsend case (bottom right) is a “better” local minimum but not the global minimum.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | |  | | |
|  | | |  | | |
|  | **Rosenbrok 2** | **Simionescu** | | **Rosenbrok 1** | **Townsend** |
| **Fminsearch** | **68** | **34** | | **70** | **40** |
| **Simplexmin** | **182** | **244** | | **464** | **87** |
| **Minfun** | **23838** | **738** | | **1038** | **120359** |

However, for a few cases, the ability to reassess step sizes really improved the result (the Townsend on the left is the same as above with different initial conditions).

|  |  |
| --- | --- |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | **Townsend** | **Mishra** |
| **Fminsearch** | **30** | **33** |
| **Simplexmin** | **268** | **158** |
| **Minfun** | **120985** | **25142** |