TMA 4180 Optimeringsteori
ALGORITHMS FOR UNCONSTRAINED, LOW-DIMENSIONAL PROBLEMS

H. E. Krogstad, IMF, Spring 2008

1 INTRODUCTION

The standard edition of Matlab contains two functions for the unconstrained minimization of functions with low dimensional arguments. Such problems are quite common in practice, occurring in maximum likelihood estimation, least squares curve-fitting, and other simple situations.

The algorithms that are implemented in Matlab are well-proven and very robust, but not discussed in N&W. The present note therefore gives a short summary of algorithms, but it is recommended to look at the Matlab documentation, both for more information and even better, for implementing some of the test problems on your own.

2 ONE–DIMENSIONAL MINIMIZATION

Minimization of functions of one variable is a very old and well-studied subject, but unfortunately, not discussed in detail in any of the books we are using. N&W suggest to use algorithms using derivatives, but sometimes derivatives may be awkward or time-consuming to find. Moreover, some problems deal with non-differentiable functions.

The function `fminbnd` in Matlab (`fmin` in older versions of Matlab), is based on very classical algorithms. The function requires that we enter a Matlab function for the computation of $f(x)$ and an interval where we have a minimum. In addition, there is a set of options, allowing the users to specify the accuracy and the information amount from the iterations. The function will then return a minimum within the interval, but if that turns out to be one of the endpoints, there are good reasons to be sceptical. It is always a good rule to investigate the problem as far as possible analytically before we put it into the computer!

There is no universally best method for one-dimensional minimization. The feasibility of a method depends on how expensive it is to compute $f(x)$ and its derivatives [3]. If $f(x)$ is *very* hard to compute, we should, at any stage in the solution process, try to use *all* available function values as best as we can. This may well lead to algorithms specially tailored to the problem at hand, e.g. based on high order interpolation. Moreover, in many cases, a quick and crude approximation is all that is really needed.

Finally, it is most probably a waist of time trying to invent new and revolutionary methods for one-dimensional function minimization!

A *triplet* consists of three x-values $\{x_1, x_2, x_3\}$, $x_1 < x_2 < x_3$, and corresponding function values,

$$\{f(x_1),\ f(x_2),\ f(x_3)\}, \tag{1}$$

so that

$$f(x_2) \le \min\{f(x_1), f(x_3)\}. \tag{2}$$

We shall assume that the function is continuous and bounded from below. This will ensure that it really has a minimum (or several) somewhere within the interval $[x_1, x_3]$.

The simplest iteration method using triplets is *interval-halving.* Assume that x_2 is located in the middle of $[x_1, x_3]$, that is,

$$x_2 = \frac{x_1 + x_3}{2}. \tag{3}$$

Compute the function values in the two additional points,

$$x_4 = \frac{x_1 + x_2}{2},\ x_5 = \frac{x_2 + x_3}{2}. \tag{4}$$

This gives us three possibilities for new triplets (based on either $\{x_1, x_4, x_2\}$, $\{x_4, x_2, x_5\}$, or $\{x_2, x_5, x_3\}$), and at least one of them has to be a triplet (think about that, or make a drawing). We choose one (or branch out using all good combinations) for the continuation of the iteration. At each step, the interval enclosing the solution(s) is halved. Each iteration requires 2 function evaluations, and after k iterations, the enclosing interval $[x_1, x_3]$, which at the start had length d, has now length $d/2^k$. Counting everything in terms of function evaluations,

$$\frac{d_k}{d} = \frac{1}{2^k} = \frac{1}{2^{N/2}} = \left(\frac{1}{\sqrt{2}}\right)^N \approx 0.7^N. \tag{5}$$

After 10 function evaluations, the interval has shrunk to 2.8% of its original length. Even if we know that the interval now contains at least one minimum, we may have lost other minima on the way.

However, is possible to be a little smarter than simply halving the interval. Some of you probably know the *Fibonacci numbers* and their *Golden Ratio* limit. The number

$$g = \frac{\sqrt{5} - 1}{2} = 0.618.... \tag{6}$$

divides the interval $[0, 1]$ in what is called the Golden Ratio. The Golden Ratio triplet is special: If we start with $\{0, g, 1\}$, both the triplets $\{0, 1 - g, g\}$ and $\{1 - g, g, 1\}$ are possible Golden Ratio triplets (check it!). By evaluating the function in *one* additional point, $x = 1-g$, we thus obtain *two* new Golden Ratio triplets. At least one of them will take us further (why?), and each time the enclosing interval diminishes by the factor g:

$$\frac{d_k}{d} = g^k = g^N \approx 0.618^N. \tag{7}$$

This is slightly better than above; after 10 iterations the interval is 0.8% of the original!

With a triplet at hand, it is tempting to fit a parabola through the three points and let the minimum of the parabola be the middle point of a new triplet. This will work most of the time, but occasionally the middle point will be situated quite close to an endpoint, and the fitted parabola becomes numerically unstable. The Matlab function `fminbnd` checks this and alternates between Golden Ratio and parabolic fits [1]. It is informative to let `fminbnd` print out information about the method it uses for each iteration (see the Matlab documentation for the necessary *Option*).

There are several ways of finding an initial triplet. You may use a priori knowledge about the behaviour of the function or simply investigate the realistic interval by creating a table of function values.

It is obvious that the algorithm could miss the global minima if there are several local minima inside the initial interval.

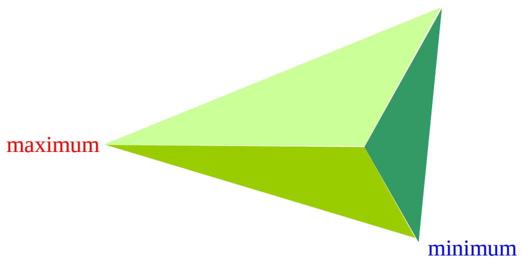

Figure 1: Simplex in $\mathbb{R}^3$ and corners where the function has maxima and minima.

3 THE AMOEBA

This algorithm, which should be called the *Nelder-Mead Method*, falls outside both the Line Search and the Trust Region philosophies. However, it is limited to low dimensional problems since it quickly becomes quite computer-expensive in terms of function evaluations as the dimension of x increases. The curious name comes from its appearance in the form of an "amoeba" that shrinks, expands, turns and moves as the iteration evolves.

A *simplex* is a convex body in $\mathbb{R}^n$ with positive volume and corners which are the only extreme points (an *extreme point* is a point that is not lying on any line segment between two other distinct points in the set). Well-known examples are triangles in $\mathbb{R}^2$ and tetrahedra in $\mathbb{R}^3$. The amoeba is such a simplex, and sometimes the method below is called the simplex method. Since it has nothing to do with the famous simplex method in Linear Programming, this name is somewhat unfortunate and should be avoided.

The *Nelder-Mead Method*

- does not require function derivatives
- needs relatively few function evaluations per iteration
- can start from anywhere
- adapts to all kind of functions
- seems to be rather difficult to analyze

The algorithm starts by defining a simplex as illustrated in Fig. 1 for $\mathbb{R}^3$, and then determines the corners with the maximum and the minimum values of the objective function.From this simplex, we make a new simplex by means of one of four *basic operations*:

- Reflection
- Expansion
- Contraction
- Scaling

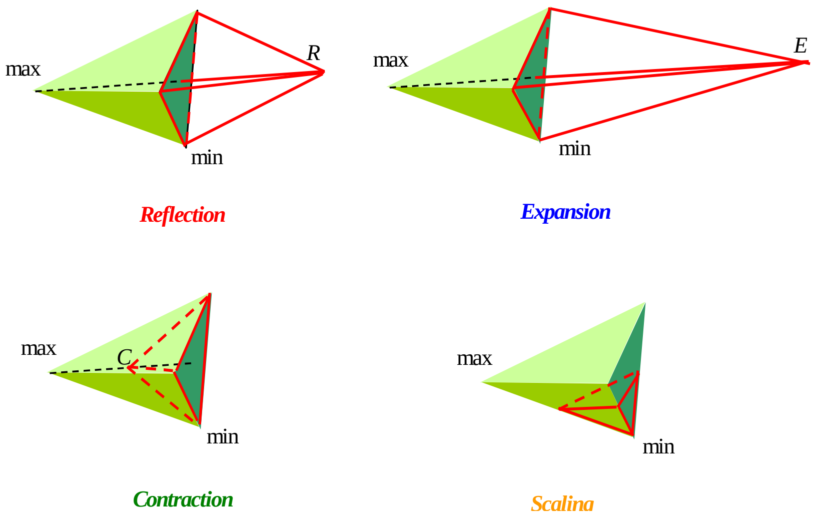

Figure 2: *Reflection* (R): Reflects the maximum point though the center point of the opposing face. *Expansion* (E): Same as reflection, but doubles the distance to the center point. *Contraction* (C): Halves the distance to the center point. *Scaling*: Contracts the simplex 50% towards the minimum point.

(The `fminsearch` function in Matlab uses an additional *outer contraction* operation). The four operations are all illustrated in Fig. 2. The first step of the iteration is to find the reflected point x_R, directly opposite to the center point of the face opposing the corner for the maximum. Based on the value of $f(x_R)$, a complete decision tree for all four operations is given in Fig. 3. Depending on the size of $f(x_R)$, we may have to compute the function contraction point, x_C, or the expansion point, x_E .

The Nelder-Mead Algorithm is very popular. Current hits on Google exceed 12000, and a nice animation in [5] shows the algorithm at work. Apart from [5], references to the algorithm are Paragraph $10-4$ in [4], the Matlab function `fminsearch.m,` and also [2].

References

[1] Forsythe, G. E., M. A. Malcolm, and C. B. Moler: *Computer Methods for Mathematical Computations*, Prentice-Hall, 1976.

[2] Lagarias, J.C., J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," *SIAM Journal of Optimization*, Vol. 9 Number 1, pp. 112-147, 1998.

[3] Luenberger, D. G.: *Linear and Nonlinear Programming*, 2nd ed., Addison Westley, 1984.

[4] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipies in C*, 2nd Edition, Cambridge University Press, 1992.

[5] Wikipedia: `http://en.wikipedia.org/wiki/Nelder-Mead_method`

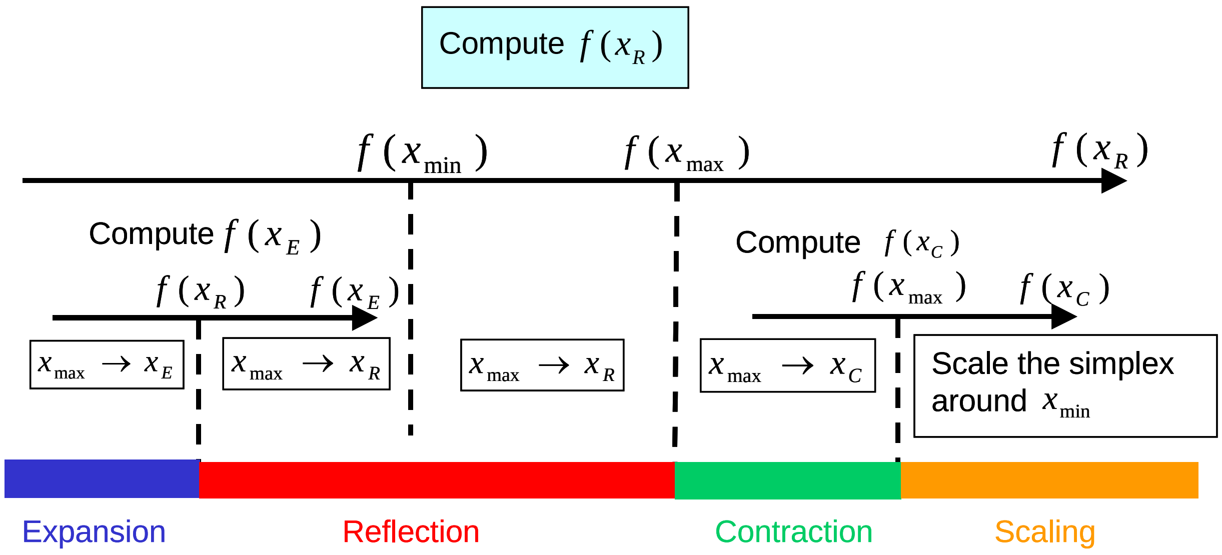

Figure 3: Decision graph for the Amoeba algorithm.