MATH 612: CM4ES&FM

Spring'14 | Work'n'code time | May 12

The Nelder-Mead algorithm

We are going to code the Nelder-Mead algorithm for unconstrained derivative-free optimization. In the description of the algorithm, the word *continue* will be used to mean that we move to the next iteration. (It corresponds to the MATLAB command of the same name.)
The input of the algorithm is:

- a function f of n variables,
- $n+1$ points $x_i \in \mathbb{R}^n$ such that the directions $x_j - x_1$ are linearly independent (we will store them in an $(n+1)\times n$ matrix
- four design parameters,

$$\mu_{\rm ref} = 1, \quad \mu_{\rm exp} = 2, \quad \mu_{\rm con} = \tfrac{1}{2}, \quad \mu_{\rm int} = -\tfrac{1}{2}.$$

- a tolerance value
- a value for the maximum number of evaluations of f.

We will also need a sorting routine: given the $n+1$ points x_i and the values $v_i = f(x_i)$, by sorting we mean to reorder the points so that

$$v_1 \le v_2 \le \ldots \le v_{n+1}.$$

This can be easily done using the MATLAB function `sort`. Some expressions will be shortened for better visibility of the algorithm.

```
compute v_i = f(x_i) for i = 1, ..., n+1
while v_max - v_min > tol
    sort the points
    x̄ = (1/n) Σ_{j=1}^{n} x_j
    d = x̄ - x_{n+1}
    x_ref = x̄ + μ_ref d
    v_ref = f(x_ref)
    if v_1 ≤ v_ref < v_n
        (x_{n+1}, v_{n+1}) = (x_ref, v_ref)          % reflection
        continue
    else if v_ref < v_1          % expension
        x_exp = x̄ + μ_exp d
        v_exp = f(x_ref)
        (x_{n+1}, v_{n+1}) = best of (x_exp, v_exp) and (x_ref, v_ref)
        continue
    else if v_n ≤ v_ref < v_{n+1}          % contraction
        x_con = x̄ + μ_con d
        v_con = f(x_con)
        if v_con < v_ref
            (x_{n+1}, v_{n+1}) = (x_con, v_con)
            continue
        end
    else if v_ref > v_{n+1}          % interior contraction
        x_int = x̄ + μ_int d
        v_int = f(x_int)
        if v_int < v_{n+1}
            (x_{n+1}, v_{n+1}) = (x_int, v_int)
            continue
        end
    end
    compute x_i = x_1 + (1/2)(x_i - x_1) for i = 2, ..., n+1          % shrinking
    compute v_i = f(x_i) for i = 2, ..., n+1
end
```

The algorithm should also be complemented with a counter of the number of evaluations of f. Once this value is larger that the limit we had input, the code should stop.