**Coursework 1 Part 1: Data-driven Optimisation**

*Team Abyss:* Zhongqi Zhuang, Elton Lam, John Huang and Nicholas Gerard

*Department of Chemical Engineering, Imperial College London, U.K.*

**1. Preface**

In pursuit of a solution for this coursework, various algorithms recommended in literature for black box function optimisation under limited evaluations were tested, including Powell's method, Adam, and Bayesian Optimisation. Nevertheless, none of these showed performance levels commensurate with the effectiveness demonstrated by Scipy's Cobyla algorithm [1]. As a result, consensus was reached within the group that it was necessary to fight fire with fire, leading to the conception of Jordanla, an algorithm engineered to operate on a similar foundation, with the explicit intent of challenging Cobyla's (Kobe) supremacy.

**2. Intuition**

**2.1 Big picture**

The foundational concept behind Jordanla is the construction and iterative refinement of a simplex, referred to in this context as a polygon. A simplex, in multidimensional optimization, is a geometric figure consisting of *n* + 1 vertices in an *n*-dimensional space. The Jordanla algorithm starts by initializing a polygon around a given starting point. Each vertex of the polygon represents a potential solution, and the objective function is evaluated at each of these points.

Following the iterative update process, the polygon should shrink and converge to the optimal solution. Jordanla identifies the vertex of the polygon with the worst objective function value and then seeks to replace this point with a better one. It does this through a reflection process, creating a new point by reflecting the worst vertex across the centroid of the remaining vertices. This step is guided by the intuition that moving away from the least desirable solution is likely to yield improvement. If this reflected point proves to be a better solution, it replaces the worst vertex; otherwise, the algorithm employs bisection to shrink the polygon, ensuring a more cautious exploration of the solution space, as the optimal solution is likely to be contained within the polygon.

**2.2 Selection of initial point**

Starting from a lower initial value of the black box function is advantageous, particularly when dealing with unimodal objective functions. Jordanla possesses a characteristic gradual progression towards the optimal solution and lacks the ability to rapidly traverse the solution space. When employing a random search methodology to identify the lowest starting point, it is possible for the algorithm to begin its search from positions far away from the optimum. Consequently, the algorithm exhibits consistent performance and can expedite the discovery of optima when the chosen starting point is located in the middle of the specified bounds. In the context of this specific coursework, the choice was made to set the algorithm’s starting point at x-values equal to zero.

A step size of three was adopted for initializing the polygon, which stems from the fact that the objective function is subjected to uniform random shifts of size plus or minus three units. Based on the assumption that most of the optima of the black box functions tend to be at x-values of zero, the likelihood of locating optima, particularly in the case of unimodal functions, outside the bounds of plus or minus three units is low.

**3. Methodology**

**3.1 Initialising and evaluating polygon**

The algorithm begins by initializing a polygon, in the multidimensional solution space with *n* + 1 vertices for an *n*-dimensional problem. The vertices are generated around the chosen starting point, each displaced by a fixed step size in one of the dimensions. Each vertex of the polygon, representing a potential solution, is then evaluated using the black box function.

**3.2 Updating the polygon**

After evaluating the vertices, the algorithm identifies the vertex with the worst function value. The next step is to generate a new point that could potentially replace this least desirable vertex.

The algorithm calculates the centroid of the remaining vertices and reflects the worst vertex across this centroid, creating a new potential solution. This reflection step is governed by the intuition that moving away from a poor solution, in a direction informed by better solutions, is likely to yield improvement.

If the reflected point offers a better function value than the worst vertex, it replaces the latter in the polygon, signifying a successful move towards an optima. If the reflected point does not yield an improvement, the algorithm employs a bisection strategy and shrinks the polygon by moving the worst vertex halfway towards the centroid. When this happens, an optima is likely to be located within the polygon and thus it is desired to avoid taking aggressive steps that might skip over the optima.

**3.3 Iterations and termination**

These steps are repeated iteratively, refining the polygon while moving towards an optima. The iterations continue until one of the termination conditions is met - either the time limit is exceeded or the maximum allowed function evaluations is reached. Upon termination, the algorithm outputs the vertex of the polygon with the lowest function value.

**4. Pseudocode**

Jordanla(*func, x\_dim, bounds, iter\_tot*)

*start\_point ← array of zeros of length*

*x\_dim*

*n ← x\_dim + 1*

*p ←* Initialise\_p*(start\_point, n, x\_dim)*

*func\_vals ←* Evaluate\_func\_at\_p(*p, func*)

*iters ← n*

***for*** *i* ***from*** *0* ***to*** *iter\_tot - n*

*p, func\_vals ←* Update\_p(*p, func\_vals, bounds*)

*iters ← iters + 1*

***if***Time\_exceeded()***or*** *iters > iter\_tot*

*break*

*best\_idx ←* ***index of min value in*** *func\_vals*

*best\_x ← p[best\_idx]*

*best\_f ← func\_vals[best\_idx]*

***return*** *best\_x, best\_f*

Initialise\_p(*start\_point, n, x\_dim, step\_size=3*)

*p ← array of zeros with size (n, x\_dim)*

*p [0] ← start\_point*

***for*** *i* ***from*** *1* ***to*** *n*

*p [i, i - 1] ← step\_size*

*centroid\_p ←* Mean(*p*)

*p\_centred ← p – centroid\_p*

***return*** *p\_centred*

Update\_P(*p, func\_vals, bounds*)

*worst\_x ← index of max value in func\_vals*

*centroid ← mean of p excluding p[worst\_x]*

*reflection ← centroid + (centroid - p[worst\_x])*

*reflection ←* Clip\_to\_bounds(*reflection, bounds*)

*reflection\_val ← func(reflection)*

***if*** *reflection\_val < func\_vals[worst\_x]*

*polygon[worst\_x] ← reflection*

*func\_vals[worst\_x] ← reflection\_val*

***else***

*p[worst\_x] ← (p [worst\_x] + centroid)/2*

***return*** *polygon, func\_vals*

**5. Figure of algorithm**

**A flowchart of a algorithm

Description automatically generated**

**6. References**

[1]<https://www.docdroid.net/QTS9gns/powell1994-pdf#page=4>