Applied and Computational Mathematics
2017; 6(4-1): 72-82
http://www.sciencepublishinggroup.com/j/acm
doi: 10.11648/j.acm.s.2017060401.17
ISSN: 2328-5605 (Print); ISSN: 2328-5613 (Online)

Global Optimization with Descending Region Algorithm

Loc Nguyen

Vietnam Institute of Mathematics, Hanoi, Vietnam

Email address:
ng_phloc@yahoo.com

To cite this article:
Loc Nguyen. Global Optimization with Descending Region Algorithm. *Applied and Computational Mathematics.* Special Issue: Some Novel Algorithms for Global Optimization and Relevant Subjects. Vol. 6, No. 4-1, 2017, pp. 72-82. doi: 10.11648/j.acm.s.2017060401.17

Received: April 8, 2017; **Accepted**: April 10, 2017; **Published**: June 9, 2017

Abstract: Global optimization is necessary in some cases when we want to achieve the best solution or we require a new solution which is better the old one. However global optimization is a hazard problem. Gradient descent method is a well-known technique to find out local optimizer whereas approximation solution approach aims to simplify how to solve the global optimization problem. In order to find out the global optimizer in the most practical way, I propose a so-called descending region (DR) algorithm which is combination of gradient descent method and approximation solution approach. The ideology of DR algorithm is that given a known local minimizer, the better minimizer is searched only in a so-called descending region under such local minimizer. Descending region is begun by a so-called descending point which is the main subject of DR algorithm. Descending point, in turn, is solution of intersection equation (A). Finally, I prove and provide a simpler linear equation system (B) which is derived from (A). So (B) is the most important result of this research because (A) is solved by solving (B) many enough times. In other words, DR algorithm is refined many times so as to produce such (B) for searching for the global optimizer. I propose a so-called simulated Newton – Raphson (SNR) algorithm which is a simulation of Newton – Raphson method to solve (B). The starting point is very important for SNR algorithm to converge. Therefore, I also propose a so-called RTP algorithm, which is refined and probabilistic process, in order to partition solution space and generate random testing points, which aims to estimate the starting point of SNR algorithm. In general, I combine three algorithms such as DR, SNR, and RTP to solve the hazard problem of global optimization. Although the approach is division and conquest methodology in which global optimization is split into local optimization, solving equation, and partitioning, the solution is synthesis in which DR is backbone to connect itself with SNR and RTP.

Keywords: Global Optimization, Gradient Descent Method, Descending Region, Descending Point

1. Introduction

The local optimization problem is specified simply as follows [1]:

$$\begin{cases} \min\limits_{x} f(x) \\ \text{subject to} \begin{array}{l} u_i(x) \le 0, \forall i = \overline{1,m} \\ v_j(x) = 0, \forall j = \overline{1,n} \end{array} \end{cases}$$

Where f called target function is analytic and convex. All constraints u_i and v_j are convex functions. The Lagrangian function is composed as follows [2, p. 215]:

$$L(x,\lambda,\mu) = f(x) + \sum_{i=1}^{m} \lambda_i u_i(x) + \sum_{j=1}^{n} \mu_j v_j(x)$$

Where,

$$\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)^T$$

$$\mu = (\mu_1, \mu_2, \dots, \mu_n)^T$$

Variables λ and μ are called Lagrange multipliers [3]. Note that the notation T denotes transposition operator because λ and μ are column vectors. As default, vectors mentioned in this research are column vectors if there is no additional explanation. If there is no constraint, Lagrangian function is the target function f. The duality principle is derived from local optimization problem as follows [4, p. 160]:

$$\begin{cases} x^* = \operatorname*{argmin}\limits_{x} L(x,\lambda,\mu) \\ (\lambda^*, \mu^*) = \operatorname*{argmax}\limits_{\lambda,\mu \ge 0} L(x,\lambda,\mu) \end{cases}$$

The point x^* is called *local optimizer* or *local minimum point*, which is also the saddle point of Lagrangian function. If x^* is a local optimizer, it satisfies Karush – Kuhn – Tucke (KKT) condition [5] as follows:

$$\begin{cases} \frac{\partial L(x,\lambda,\mu)}{\partial x} = 0 \\ g_i(x) \le 0, h_j(x) = 0, \forall i = \overline{1,m}, \forall j = \overline{1,n} \\ \mu_i \ge 0, \forall i = \overline{1,m} \\ \mu_i g_i(x) = 0, \forall i = \overline{1,m} \end{cases}$$

Where $\frac{\partial L(x,\lambda,\mu)}{\partial x}$ is partial derivative of $L(x, \lambda, \mu)$ with regard to x. In practice, x^* is solved by combination of duality principle and KKT condition. Concretely, suppose a triple (x^*, λ^*, μ^*) is solution of following equation system:

$$\begin{cases} \frac{\partial L(x,\lambda,\mu)}{\partial x} = 0 \\ \frac{\partial L(x,\lambda,\mu)}{\partial \lambda} = \mathbf{0}^T \text{ and } \frac{\partial L(x,\lambda,\mu)}{\partial \mu} = \mathbf{0}^T \\ \mu_i u_i(x) = 0 \text{ and } v_j(x^*) = 0, \forall i = \overline{1,m}, \forall j = \overline{1,n} \end{cases}$$

Where 0 denotes zero vector $0 = (0, 0,\ldots, 0)^T$ whereas $\frac{\partial L(x,\lambda,\mu)}{\partial \lambda}$ and $\frac{\partial L(x,\lambda,\mu)}{\partial \mu}$ are partial derivatives of $L(x, \lambda, \mu)$ with regard to λ and μ, respectively, which are row vectors as follows:

$$\frac{\partial L(x,\lambda,\mu)}{\partial \lambda} = \left(\frac{\partial L(x,\lambda,\mu)}{\partial \lambda_1}, \frac{\partial L(x,\lambda,\mu)}{\partial \lambda_2}, \ldots, \frac{\partial L(x,\lambda,\mu)}{\partial \lambda_m}\right)$$
$$\frac{\partial L(x,\lambda,\mu)}{\partial \mu} = \left(\frac{\partial L(x,\lambda,\mu)}{\partial \mu_1}, \frac{\partial L(x,\lambda,\mu)}{\partial \mu_2}, \ldots, \frac{\partial L(x,\lambda,\mu)}{\partial \mu_n}\right)$$

If such triple (x^*, λ^*, μ^*) satisfies following condition, x^* is a local optimizer:

$$\begin{cases} u_i(x^*) \le 0, \forall i = \overline{1,m} \\ \mu_i^* \ge 0, \forall i = \overline{1,m} \end{cases}$$

Alternately, gradient descent (GD) method [6] is also used to find out x^* with regard to f. This research is inspired from GD method.

The global optimization is general form of local optimization as follows [7, p. 109]:

$$\min_x \{f(x) | x \in S\}$$

Where f is arbitrary and $S = \{x | u_i(x) \le 0, v_j(x) = 0, \forall i = \overline{1,m}, \forall j = \overline{1,n}\}$ is arbitrary set.

A point x^* is called *global optimizer* or *global minimum point* if $f(x^*) \le f(x)$ for all x belonging S. Because the cost of finding out x^* is very expensive, the *approximation solution approach* issues a concept of global ε-optimization with acceptance of small error ε [7, p. 135]. The global optimizer x^* is replaced by a so-called global ε-optimizer $\bar{x}$ as follows [7, p. 123]:

$$f(x) \ge f(\bar{x}) - \varepsilon, \forall x \in S$$

Let z^0 be an arbitrary point and set k=0, the approximation solution approach has two phases to find out $\bar{\boldsymbol{x}}$ as follows [7, p. 124]:

1. From z^k, searching for x^k as a local optimizer.
2. If x^k is proved as the global ε-minimizer then, return $\bar{x} = x^k$. Otherwise, find out a feasible solution z such that $f(z) < f(x^k)$ and then, set k=k+1, set z^k=z, and go back phase 1.

As a convention, superscript numbers (in z^0, z^k, x^k, etc.) denote indices if there is no additional explanation. This research focuses on global optimization, in which I propose a so-called *descending region* (DR) algorithm that is a practical implementation of approximation solution approach. I use the very small value ε as a downward shift to determine the feasible point z in phase 2 and I use GD method to search for local optimizer. As a result, the final point is the global ε-minimizer. In other words, DR algorithm is simpler and it is an advanced variant of GD method. Section 2 is full description of DR algorithm which has two steps for each iteration, corresponding with two phases of theoretical approximation solution. The feasible point z is called descending point. It is possible to consider DR algorithm to be combination of GD method and approximation solution approach, which aims to solve global optimization problem in practical way. GD method follows descending direction which is the opposite of gradient to reach local optimizer. Note that gradient is the first derivative of multivariate function. GD is described in subsection 2.1. There are some alternatives of GD method such as particle swarm optimization (PSO) and Quasi-Newton method.

PSO [8] was firstly invented by James Kennedy and Russell Eberhart in 1995 and later on, it attracts many researchers. PSO is inspired from bird flocking, fish schooling, and swarming theory [8, p. 1942] in which a swarm of individuals (birds, fishes, etc.) called particles moves in the searching space in order to reach the global optimizer. Each particle owns a position and a velocity. The movement of each particle, which is change of its position and velocity, is affected by itself and other particles (swarm) [9]. The swarm's position is the best one over all particles' positions. At the i^{th} iteration, the swarm's position x^i is best if it satisfies most fitness criterion [10, p. 3]; as usual, $f(x^i)$ is as small as possible. After a finite number of movements, the swarm's position reaches to global optimizer. The meaning of velocity is approximate to the meaning of gradient but velocity is not gradient. So PSO does not require target function f to be analytic [9]. Quasi-Newton method [11] is an alternative of Newton's method to find out local optimizer in case that Hessian matrix, which is second derivative of multivariable function, is not existent or not easy to be computed. Quasi-Newton method finds out a solution of the equation created by setting gradient to be zero [11], as local optimizer. Quasi-Newton method updates Hessian matrix by successive gradients after iterations [11].

It is necessary to survey some other researches relevant to global ε-optimization. If target function f is sum of generalized polynomial ratios, Jiao et al. [12] proposed a three-level linear relaxation method, in which the original optimization problem is converted into the three-level linear relaxation programming problem (LRP) [12, pp. 190-192] and then, a branch and bound algorithm [12, pp. 192-193] is proposed to solve such LRP. The algorithm partitions domain space into sub-rectangles and lower bound of each sub-rectangle is calculated based on solution of LRP. The final lower bound is the minimum one over all sub-rectangles. If deviation between upper bound and lower bound is smaller than ε, the global ε-optimal solution of relaxation problem is determined. The convergence of branch and bound algorithm is proved [12, pp. 193-194]. Larsson and Patriksson [13] allowed duality gap to be nonzero in order to find out the near-optimizer which is an approximation of global optimizer with concepts of ε-optimality and δ-complementarity [13, p. 439]. They establish the new condition of global optimality with duality gap size $\varepsilon + \delta$ to solve the near-optimizer [13, p. 441]. The literature of global optimization and relevant subject is very large. Please read the book "*Convex Analysis and Global Optimization*" of Tuy Hoang [7] to comprehend theory of global optimization. This research only focuses on practical implementation of ε-optimality approach. Section 2 is full description of DR algorithm. Section 3 describes how to solve intersection equation by simulated Newton – Raphson (SNR) algorithm in order to find out descending region necessary to DR algorithm. In turn, section 4 describes how to determine the starting point necessary to SNR algorithm, by random testing point (RTP) algorithm. Section 5 is the conclusion with some suggestions in future trend. In general, the global optimization problem is divided into three smaller problems such as local optimization, solving equation, and partitioning which are solved by three algorithms such as DR, SNR, and RTP, respectively. DR algorithm is the most important one, which is connect itself with SNR and RTP.

2. Descending Region Algorithm

Suppose f is the target analytic function that we need to find out its global optimizer with regard that f is scalar-by-vector function (multivariate function).

$$f\colon \mathbb{S} \to \mathbb{R}$$

Where $\mathbb{R}$ is real number field and $\mathbb{S}$ is real vector space such that $\mathbb{S} \subseteq \mathbb{R}^n$. Suppose f is arbitrary but $\mathbb{S}$ is convex. Descending region (DR) algorithm is iterative algorithm whose input is an initial point $\boldsymbol{\omega}^0$ and the output is global optimizer $\mathbf{z}^{**}$. The ideology of DR algorithm is that given a known local optimizer $\mathbf{z}^*$, the better local optimizer is searched only in a so-called *descending region* under the known point $\mathbf{z}^*$. If the target function has global minimum value, DR algorithm is terminated after a finite number of iterations; at that time the local optimizer $\mathbf{z}^*$ approaches the global optimizer $\mathbf{z}^{**}$. In DR algorithm, the descending region is begun by a so-called *descending point* which is defined as the point under the optimizer $\mathbf{z}^*$ and so the next better optimizer is searched under descending point. Suppose $\mathbf{z}^i$ is the descending point at the i^{th} iteration and $\mathbf{z}^i$ is initialized by $\boldsymbol{\omega}^0$. DR algorithm has many iterations and each iteration includes two steps:

1. *Step* 1: Searching for local optimizer step.
2. *Step* 2: Determining descending region step.

Such two steps are implementations of two phases of approximation solution approach [7, p. 124]. In general, Table 1 shows the pseudo-code like for DR algorithm. Note, the input is an arbitrary point $\boldsymbol{\omega}^0$ and the output is global optimizer $\mathbf{z}^{**}$.

Table 1. *DR algorithm.*

```
//Initialization
$\mathbf{z}^0 = \boldsymbol{\omega}^0$
$\mathbf{z}^{**} = \mathbf{z}^* = +\infty$
$i = 0$

Loop
    $\mathbf{z}^*$:= searching for local optimizer with the input $\mathbf{z}^i$ (step 1)
    $\mathbf{z}^i$:= determining descending point with the input $\mathbf{z}^*$ (step 2)
    $\mathbf{z}^{**} = \mathbf{z}^*$
    $i = i + 1$
While (no descending point $\mathbf{z}^i$ found)
```

2.1. Searching for Local Optimizer Step

With the descending point $\mathbf{z}^i$, I apply GD method [1] [6] to find out the local optimizer $\mathbf{z}^*$. Note that $\mathbf{z}^i$ is initialized by $\boldsymbol{\omega}^0$. We know that GD method is also iterative method; suppose at the iteration j^{th} in this method, the next candidate point $\boldsymbol{z}_{j+1}^i$ is computed as follows [6]:

$$\boldsymbol{z}_{j+1}^i = \boldsymbol{z}_j^i + t_j^i \boldsymbol{d}_j^i$$

Where,

- The point $\boldsymbol{z}_j^i$ is the previous candidate point and $\boldsymbol{z}_j^i$ is initialized by the starting point $\mathbf{z}^i$.
- The direction $\boldsymbol{d}_j^i$ is the descending direction, which is the opposite of gradient of function f. We have $\boldsymbol{d}_j^i = -\nabla f(\boldsymbol{z}_j^i)$ where ∇f denotes the gradient of function f.
- The value t_j^i is the length of the descending direction $\boldsymbol{d}_j^i$.

After m iterations, the point $\boldsymbol{z}_j^i$ converges to the local optimizer $\mathbf{z}^*$. If the terminated condition of GD method is equal to $\mathbf{0}$ $\left(\boldsymbol{d}_j^i = \mathbf{0}^T\right)$ then, $\mathbf{z}^*$ is likely local optimizer and we go to step 2. Otherwise, the terminated condition is that GD method runs after m iterations, we compare the current optimizer $\mathbf{z}^*$ with the previous optimizer $\boldsymbol{y}^*$:

- If $\mathbf{z}^*$ is equal to $\boldsymbol{y}^*$, then DR algorithm stops and $\mathbf{z}^*$ is the global optimizer, $\mathbf{z}^{**} = \mathbf{z}^*$.
- If $\mathbf{z}^*$ is not equal to $\boldsymbol{y}^*$, then go to step 2.

2.2. Determining Descending Region Step

Given the local optimizer $\mathbf{z}^*$ found out in step 1 and let ε be a very small pre-defined positive number, $\varepsilon > 0$. The number ε is also called *descending error*. Let $h(\boldsymbol{x})$ be hyper-plane that goes through the level $f(\mathbf{z}^*) - \varepsilon$ and is parallel to the abscissa hyper-plane or domain plane of function f, we have:

$$h(\boldsymbol{x}) = f(\boldsymbol{z}^*) - \varepsilon$$

Let $f^* = f(\mathbf{z}^*)$ be the minimum value of target function f at $\mathbf{z}^*$, we have:

$$h(\boldsymbol{x}) = f^* - \varepsilon$$

The intersection between hyper-plane $h(\boldsymbol{x})$ and the function $f(\boldsymbol{x})$ is a contour at level $f^* - \varepsilon$ with note that $f(\boldsymbol{x})$ is a hyper-surface. Following is the equation of such contour:

$$f(\boldsymbol{x}) = h(\boldsymbol{x}) \Leftrightarrow f(\boldsymbol{x}) = f^* - \varepsilon$$

Or,

$$f(\boldsymbol{x}) - f^* + \varepsilon = 0$$

Note that this equation is called *intersection equation*. If the contour is not existent; in other words, if the hyper-plane $h(\boldsymbol{x})$ does not intersect with the surface $f(\boldsymbol{x})$, then DR algorithm is stopped and $\mathbf{z}^*$ is the global optimizer and we have $\mathbf{z}^{**} = \mathbf{z}^*$. Otherwise, suppose $\mathbf{z}^0$ is a point that belongs to the contour; in other words, $\mathbf{z}^0$ is a solution of the equation $f(\boldsymbol{x}) = f^* - \varepsilon$ with note that the method to find out $\mathbf{z}^0$ is proposed later, then we do two important tasks:

- Increasing i by 1, we have $i = i+1$.
- The new descending point $\mathbf{z}^i$ is assigned by the solution $\mathbf{z}^0$ and we have $\mathbf{z}^i = \mathbf{z}^0$ with attention that the index i was increased by 1. After that, we go back step 1.

It is easy to recognize that the next better local optimizer is searched only in the region under the solution $\mathbf{z}^0$. This region is descending region. So this algorithm is called descending region (DR) algorithm and solution $\mathbf{z}^0$ is also called descending point. Although both $\mathbf{z}^i$ and $\mathbf{z}^0$ are called descending points, the point $\mathbf{z}^i$ is known as descending point at the i^{th} iteration and $\mathbf{z}^0$ is solution of the equation $f(\boldsymbol{x}) = f^* - \varepsilon$. In Table 2, the pseudo-code for DR algorithm is refined with note that the input is an arbitrary point $\boldsymbol{\omega}^0$ and the output is global optimizer $\mathbf{z}^{**}$.

***Table 2.** DR algorithm refined.*

```
//Initialization
$\mathbf{z}^0 = \boldsymbol{\omega}^0$
$\mathbf{z}^{**} = \mathbf{z}^* = +\infty$
$\varepsilon$:= very small pre-defined number
$i = 0$

Loop
    //Step 1: Searching for local optimizer
    $\mathbf{z}^*$:= searching for local optimizer with the input $\mathbf{z}^i$
    If $\mathbf{z}^*$ not found then
        break
    Else If $\mathbf{z}^*$ equal previous optimizer then
        $\mathbf{z}^{**} = \mathbf{z}^*$
        break
    End If

    //Step 2: Determining descending region
    $\mathbf{z}^0$:= solution of intersection equation with the inputs $\mathbf{z}^*$ and $\varepsilon$
    $\mathbf{z}^i = \mathbf{z}^0$
    $\mathbf{z}^{**} = \mathbf{z}^*$
    $i = i + 1$
While (no descending point $\mathbf{z}^0$ found)
```

For example, given target one-variable function

$$f(x) = \frac{1}{16}x^4 - \frac{1}{12}x^3 - \frac{3}{4}x^2 - \frac{2}{3}$$

Given initial point $\boldsymbol{\omega}^0 = (-2.76, -1)$, it is easy to receive the first local optimizer $\boldsymbol{x}^* = (-2, -2)$ by applying gradient method. Suppose the very small number ε is 1, the descending point $\mathbf{z}^0$ which is solution of intersection equation $f(x) - f^* + \varepsilon = f(x) - (-2) + 1 = f(x) + 3 = 0$ is solved by the method mentioned in next section; hence we get $\mathbf{z}^0 = (1.84, -3)$. Starting from $\mathbf{z}^0 = (1.84, -3)$, the next local optimizer is found out, $\boldsymbol{x}^* = (3, -4.6)$ by applying gradient method again. The new intersection equation $f(x) + 5.6 = 0$ has no solution and it is concluded that the global optimizer is $\boldsymbol{x}^{**} = (3, -4.6)$. Fig. 1 depicts this example.

3. Method to Determine Descending Region

As aforementioned, descending point $\mathbf{z}^0$ is a solution of the equation of the intersection between hyper-plane $h(\boldsymbol{x})$ and target function $f(\boldsymbol{x})$ and descending region is the region under the solution $\mathbf{z}^0$. So determining descending region is equivalent to solving the intersection equation specified by (1) in order to find out its solution $\mathbf{z}^0$.

$$f(\boldsymbol{x}) - f^* + \varepsilon = 0 \quad (1)$$

Where $f^* = f(\mathbf{z}^*)$ and $\mathbf{z}^*$ is a local optimizer. Note that $-f^* + \varepsilon$ is scalar constant. Let $y = f(\boldsymbol{x}) - f^* + \varepsilon$, we have:

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon - f^* - y$$

Function $g(\boldsymbol{x})$ is called *augmented target function*. The surface specified by $y = f(\boldsymbol{x}) - f^* + \varepsilon$ is the same to the one specified by $g(\boldsymbol{x}) = 0$ [14] with attention that y is the scalar variable while $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T$ is the vector variable. Let ∇g be the gradient of function g, we have [14]:

$$\nabla g(\boldsymbol{x}) = (\nabla f(\boldsymbol{x}), -1) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}, -1\right)$$

Where $\frac{\partial f}{\partial x_i}$ is the partial derivative of f with regard to partial variable x_i. As a convention, gradient is row vector. The value of gradient ∇g at arbitrary point $\boldsymbol{x}^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$ is:

$$\nabla g(\boldsymbol{x}^0) = (\nabla f(\boldsymbol{x}^0), -1) = \left(\frac{\partial f}{\partial x_1}(\boldsymbol{x}^0), \frac{\partial f}{\partial x_2}(\boldsymbol{x}^0), \dots, \frac{\partial f}{\partial x_n}(\boldsymbol{x}^0), -1\right)$$

Where $\frac{\partial f}{\partial x_1}(\boldsymbol{x}^0)$ is the value of partial derivative of f with regard to partial variable x_i of $\boldsymbol{x}^0$. Please distinguish the arbitrary starting point $\boldsymbol{x}^0$ from the initial point $\boldsymbol{\omega}^0$ of DR algorithm mentioned in previous section. The gradient ∇g is normal vector of tangent hyper-plane of g and so this tangent hyper-plane at point $\boldsymbol{x}^0$ is specified by following equation [14]:

$$\nabla g(\boldsymbol{x}^0)\begin{pmatrix} \boldsymbol{x} - \boldsymbol{x}^0 \\ y - f(\boldsymbol{x}^0) + f^* - \varepsilon \end{pmatrix} = 0 \Leftrightarrow \left(\frac{\partial f}{\partial x_1}(\boldsymbol{x}^0), \frac{\partial f}{\partial x_2}(\boldsymbol{x}^0), \dots, \frac{\partial f}{\partial x_n}(\boldsymbol{x}^0), -1\right) * \begin{pmatrix} \boldsymbol{x} - \boldsymbol{x}^0 \\ y - f(\boldsymbol{x}^0) + f^* - \varepsilon \end{pmatrix} = 0$$

We deduce that

$$(x_1 - x_1^0)\frac{\partial f}{\partial x_1}(\boldsymbol{x}^0) + (x_2 - x_2^0)\frac{\partial f}{\partial x_2}(\boldsymbol{x}^0) + \cdots + (x_n - x_n^0)\frac{\partial f}{\partial x_n}(\boldsymbol{x}^0) - (y - f(\boldsymbol{x}^0) + f^* - \varepsilon) = 0$$

So we have:

$$y = f(\boldsymbol{x}^0) - f^* + \varepsilon + \sum_{i=1}^{n}(x_i - x_i^0)\frac{\partial f}{\partial x_i}(\boldsymbol{x}^0)$$

It implies

$$y = \nabla f(\boldsymbol{x}^0)\boldsymbol{x} + f(\boldsymbol{x}^0) - f^* + \varepsilon - \nabla f(\boldsymbol{x}^0)\boldsymbol{x}^0$$

Equation (2) is the hyper-line which represents the intersection between tangent hyper-plane and the hyper-plane $y = 0$ and so it is called *intersection hyper-line*.

$$\nabla f(\boldsymbol{x}^0)\boldsymbol{x} - c^0 = 0 \tag{2}$$

Where c^0 is a scalar value:

$$c^0 = \nabla f(\boldsymbol{x}^0)\boldsymbol{x}^0 + f^* - f(\boldsymbol{x}^0) - \varepsilon$$

Equation (2) has many solutions which are points belonging to it. Now we find out only one solution $\boldsymbol{x}^1$ of (2) with regard that $\boldsymbol{x}^1$ satisfies two following conditions:

1. Point $\boldsymbol{x}^1$ is the projection of $\boldsymbol{x}^0$ on the intersection hyper-line.
2. Point $\boldsymbol{x}^1$ belongs to intersection hyper-line.

The first condition implies that the vector $\boldsymbol{p}^1 = \boldsymbol{x}^0 - \boldsymbol{x}^1$ is parallel to orthogonal vector of intersection hyper-line. It is easy to infer from (2) that such orthogonal vector is $\nabla f(\boldsymbol{x}^0)$. Therefore, the first condition is interpreted by following equation:

$$\boldsymbol{p}^1 = l\nabla f(\boldsymbol{x}^0) \Leftrightarrow \begin{cases} x_1^0 - x_1^1 = l\dfrac{\partial f}{\partial x_1}(\boldsymbol{x}^0) \\ x_2^0 - x_2^1 = l\dfrac{\partial f}{\partial x_2}(\boldsymbol{x}^0) \\ \vdots \\ x_n^0 - x_n^1 = l\dfrac{\partial f}{\partial x_n}(\boldsymbol{x}^0) \end{cases} \Leftrightarrow \begin{cases} x_1^1 + 0 + \cdots + 0 + l\dfrac{\partial f}{\partial x_1}(\boldsymbol{x}^0) = x_1^0 \\ 0 + x_2^1 + \cdots + 0 + l\dfrac{\partial f}{\partial x_2}(\boldsymbol{x}^0) = x_2^0 \\ \vdots \\ 0 + 0 + \cdots + x_n^1 + l\dfrac{\partial f}{\partial x_n}(\boldsymbol{x}^0) = x_n^0 \end{cases}$$

Where $\boldsymbol{x}^1 = (x_1^1, x_2^1, \ldots, x_n^1)^T$ and l are unknowns.

The second condition implies that $\boldsymbol{x}^1$ satisfies (2).

$$\nabla f(\boldsymbol{x}^0)\boldsymbol{x}^1 - c^0 = 0 \Leftrightarrow x_1^1\frac{\partial f}{\partial x_1}(\boldsymbol{x}^0) + x_2^1\frac{\partial f}{\partial x_2}(\boldsymbol{x}^0) + \cdots + x_n^1\frac{\partial f}{\partial x_n}(\boldsymbol{x}^0) = c^0$$

We set up the set of equations so as to determine $\boldsymbol{x}^1$ as below:

$$\begin{cases} x_1^1 + 0 + \cdots + 0 + l\dfrac{\partial f}{\partial x_1}(\boldsymbol{x}^0) = x_1^0 \\ 0 + x_2^1 + \cdots + 0 + l\dfrac{\partial f}{\partial x_2}(\boldsymbol{x}^0) = x_2^0 \\ \vdots \\ 0 + 0 + \cdots + x_n^1 + l\dfrac{\partial f}{\partial x_n}(\boldsymbol{x}^0) = x_n^0 \\ x_1^1\dfrac{\partial f}{\partial x_1}(\boldsymbol{x}^0) + x_2^1\dfrac{\partial f}{\partial x_2}(\boldsymbol{x}^0) + \cdots + x_n^1\dfrac{\partial f}{\partial x_n}(\boldsymbol{x}^0) + 0 = c^0 \end{cases}$$

Such set of equations is called projection set of equations or projection system. Note that the unknowns of projection system are x_1^1, x_2^1,…, x_n^1 and l. Let $\boldsymbol{A}^0$ and $\boldsymbol{b}^0$ be matrix and vector such that:

$$\boldsymbol{A}^0 = \begin{pmatrix} 1 & 0 & \cdots & 0 & \frac{\partial f}{\partial x_1}(\boldsymbol{x}^0) \\ 0 & 1 & \cdots & 0 & \frac{\partial f}{\partial x_2}(\boldsymbol{x}^0) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \frac{\partial f}{\partial x_n}(\boldsymbol{x}^0) \\ \frac{\partial f}{\partial x_1}(\boldsymbol{x}^0) & \frac{\partial f}{\partial x_2}(\boldsymbol{x}^0) & \cdots & \frac{\partial f}{\partial x_n}(\boldsymbol{x}^0) & 0 \end{pmatrix}$$

$$\boldsymbol{b}^0 = \begin{pmatrix} \boldsymbol{x}^0 \\ c^0 \end{pmatrix} = \begin{pmatrix} x_1^0 \\ x_2^0 \\ \vdots \\ x_n^0 \\ c^0 \end{pmatrix}$$

$$c^0 = \nabla f(\boldsymbol{x}^0)\boldsymbol{x}^0 + f^* - f(\boldsymbol{x}^0) - \varepsilon$$

The projection system is re-written, as seen in (3):

$$\boldsymbol{A}^0 \begin{pmatrix} \boldsymbol{x}^1 \\ l \end{pmatrix} = \boldsymbol{b}^0 \tag{3}$$

Matrix $\boldsymbol{A}^0$ is called projection matrix and vector $\boldsymbol{b}^0$ is called projection vector. The determinant of matrix $\boldsymbol{A}^0$ denoted $|\boldsymbol{A}^0|$ is not equal to 0 if and only if the gradient $\nabla f(\boldsymbol{x}^0)$ is not equal to $\boldsymbol{0}^T$. Suppose $|\boldsymbol{A}^0|$ is not equal to 0, it is easy to find out $\boldsymbol{x}^1$ by Cramer's method as follows [15, pp. 136-138]:

$$\boldsymbol{x}^1 = \begin{pmatrix} x_1^1 \\ x_2^1 \\ \vdots \\ x_n^1 \end{pmatrix} = \frac{1}{|\boldsymbol{A}^0|} \begin{pmatrix} |\boldsymbol{A}_1^0| \\ |\boldsymbol{A}_2^0| \\ \vdots \\ |\boldsymbol{A}_n^0| \end{pmatrix}$$

Let $\boldsymbol{A}_j^0$ is the matrix constructed by replacing j^{th} column in matrix $\boldsymbol{A}^0$ by projection vector $\boldsymbol{b}^0$.

I proposed the iterative method which is a simulation of the Newton – Raphson method [16, pp. 67-71] so as to solve (1) based on (3). The proposed method is called *simulated Newton – Raphson* (SNR) algorithm. Suppose we have an approximate solution $\boldsymbol{x}^k$ at the k^{th} iteration, we set up the projection system based on $\boldsymbol{x}^k$ in order to find out the next better solution $\boldsymbol{x}^{k+1}$; hence $\boldsymbol{x}^{k+1}$ is solution of (4).

$$\boldsymbol{A}^k \begin{pmatrix} \boldsymbol{x}^{k+1} \\ l \end{pmatrix} = \boldsymbol{b}^k \tag{4}$$

Where,

$$\boldsymbol{A}^k = \begin{pmatrix} 1 & 0 & \cdots & 0 & \frac{\partial f}{\partial x_1}(\boldsymbol{x}^k) \\ 0 & 1 & \cdots & 0 & \frac{\partial f}{\partial x_2}(\boldsymbol{x}^k) \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \frac{\partial f}{\partial x_n}(\boldsymbol{x}^k) \\ \frac{\partial f}{\partial x_1}(\boldsymbol{x}^k) & \frac{\partial f}{\partial x_2}(\boldsymbol{x}^k) & \cdots & \frac{\partial f}{\partial x_n}(\boldsymbol{x}^k) & 0 \end{pmatrix}$$

$$\boldsymbol{b}^k = \begin{pmatrix} \boldsymbol{x}^k \\ c^k \end{pmatrix} = \begin{pmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_n^k \\ c^k \end{pmatrix}$$

$$c^k = \nabla f(\boldsymbol{x}^k)\boldsymbol{x}^k + f^* - f(\boldsymbol{x}^k) - \varepsilon$$

As aforementioned, the positive number ε is a very small pre-defined number and f^* is the local minimum value at $\mathbf{z}^*$. Note that $\boldsymbol{A}^k$ and $\boldsymbol{b}^k$ are totally determined according to $\boldsymbol{x}^k$ and $\boldsymbol{x}^k$ is initialized by an arbitrary point $\boldsymbol{x}^0$. It is easy to infer that the solution $\boldsymbol{x}^{k+1}$ is calculated as below:

$$\boldsymbol{x}^{k+1} = \begin{pmatrix} x_1^{k+1} \\ x_2^{k+1} \\ \vdots \\ x_n^{k+1} \end{pmatrix} = \frac{1}{|\boldsymbol{A}^k|} \begin{pmatrix} |\boldsymbol{A}_1^k| \\ |\boldsymbol{A}_2^k| \\ \vdots \\ |\boldsymbol{A}_n^k| \end{pmatrix}$$

Where $\boldsymbol{A}_j^k$ is the matrix constructed by replacing j^{th} column in projection matrix $\boldsymbol{A}^k$ by projection vector $\boldsymbol{b}^k$.

It is easy to recognize that $\boldsymbol{x}^{k+1}$ is calculated based on previous $\boldsymbol{x}^k$ and so SNR algorithm is an iterative process. If the descending point $\mathbf{z}^0$ which is solution of (1) is existent then, $\boldsymbol{x}^k$ will approach $\mathbf{z}^0$ after a finite number of iterations. In other words, the descending region which supports us to search for global optimizer is determined.

The convergence of SNR algorithm is dependent on the starting point $\boldsymbol{x}^0$. If $\boldsymbol{x}^0$ is not in the volume where SNR algorithm converges, no solution $\mathbf{z}^0$ is found although there is existence of solutions of (1). Moreover, the closer to descending point $\mathbf{z}^0$ the point $\boldsymbol{x}^0$ is, the faster the convergence speed is. So the way to choose right $\boldsymbol{x}^0$ is very important, which is mentioned in next section.

There are two terminated conditions of SNR algorithm to determine descending point $\mathbf{z}^0$:

- Equation (1) has a solution $\boldsymbol{x}^k$ where $f(\boldsymbol{x}^k)$ is equal or approximated to 0 at the k^{th} iteration, $f(\boldsymbol{x}^k) \approx 0$. At that time we have $\mathbf{z}^0 = \boldsymbol{x}^k$.
- Equation (1) has no solution when the determinant $|\boldsymbol{A}^k|$ is equal to 0 at the k^{th} iteration.

It is necessary to make a simple example of SNR algorithm. Suppose f^*=1, ε=1, and $f(\boldsymbol{x})$ is approximated to $x_1^2 + x_2^2 - 4$ in solution volume where the superscript number "2" denotes the square. Some advanced methods such as feasible length [17] and minimizing square error [18] are suggested to approximate f by Taylor polynomial. The augmented target function is $g(\boldsymbol{x}) = x_1^2 + x_2^2 + \varepsilon - f^* - y = x_1^2 + x_2^2 - 4 - y$. The gradient of f is $\nabla f(\boldsymbol{x}) = (2x_1, 2x_2)$. Given starting point $\boldsymbol{x}^0 = (2, 1)^T$, at the first iteration, we have the initial solution as follows:

$$f(\boldsymbol{x}^0) = 1$$
$$\boldsymbol{A}^0 = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{pmatrix}$$
$$\boldsymbol{b}^0 = (2,1,9)^T$$
$$\boldsymbol{x}^1 = (1.8,0.9)^T$$

The SNR algorithm converges at the 4th iteration with $\boldsymbol{x}^4 = (1.79, 0.89)^T$ and $f(\boldsymbol{x}^4) \approx 0$. So we have the descending point $\mathbf{z}^0 = \boldsymbol{x}^4 = (1.79, 0.89)^T$. In Fig. 2, the target surface f is marked yellow and the tangent hyper-plane of f at $\boldsymbol{x}^4$ ($y = -8 + 3.58x_1 + 1.79y$) is marked blue.

Because the cost of solving (1) by SNR algorithm is significant, it is necessary to test whether (1) has solution or not before finding out the descending point. If there is no existence of solutions of (1), DR algorithm is stopped and we concludes that the current local optimizer $\mathbf{z}^*$ is global optimizer $\mathbf{z}^{**}$. In practice, we often apply DR algorithm into finding out the global optimizer in a given volume $[\boldsymbol{a}, \boldsymbol{b}]$ which is an interval in $\mathbb{R}$, rectangle in $\mathbb{R}^2$ or volume in $\mathbb{R}^3$.

Therefore, if (1) is a polynomial equation, $p(x) = 0$ where $p(x) = f(x) - (f(\mathbf{z}^*) + \varepsilon)$ is a polynomial and x is scalar unknown, there is some methods to determine interval of solutions, for example, given $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x^1 + a_0$ and let A be the largest among absolute values of coefficients, $A = max\{|a_1|, |a_2|, \ldots, |a_{n-1}|, |a_n|\}$, then the upper bound u of all real solutions is calculated as below [6]:

$$u = 1 + \frac{A}{a_n}$$

Where a_n is the coefficient of x^n with attention that the number n denotes the n^{th} power when we often use the superscript number to denote index. If u is smaller than lower bound a of given interval $[a, b]$, then (1) has no solution. We can use another method, Sturm's theorem [6] [19], to determine the number of distinct real solutions located in given interval $[a, b]$. If (1) is arbitrary equation, we still use the terminated condition $|\boldsymbol{A}^k| = 0$ and so how to choose optimal starting point $\boldsymbol{x}^0$ is very important and is mentioned in next section.

There is a case that target function f has infinitely many local optimizers, which means that f has no global optimizer. This case leads DR algorithm to run in infinite loop. So we add one more terminated condition that DR algorithm will stop after M iterations. In Table 3, the pseudo-code for DR algorithm is refined again with note that the input is an arbitrary point $\boldsymbol{\omega}^0$ and the output is global optimizer $\mathbf{z}^{**}$.

Table 3. *Final version of DR algorithm.*

```
//Initialization
$\mathbf{z}^0 = \boldsymbol{\omega}^0$
$\mathbf{z}^{**} = \mathbf{z}^* = +\infty$
$\varepsilon$:= very small pre-defined number
$M$:= the maximum number of iterations
$i = 0$

Loop
    //Step 1: Searching for local optimizer
    $\mathbf{z}^*$:= searching for local optimizer with the input $\mathbf{z}^i$
    If $\mathbf{z}^*$ not found then
        break
    Else If $\mathbf{z}^*$ equal previous optimizer then
        $\mathbf{z}^{**} = \mathbf{z}^*$
        break
    End If

    //Step 2: Determining descending region
    $\mathbf{z}^0 = +\infty$
    $\boldsymbol{x}^0$:= choosing optimal starting point
    $k = 0$

    Loop //Solving intersection equation by SNR algorithm
        If $f(\boldsymbol{x}^k) \approx 0$ then
            $\mathbf{z}^0 = \boldsymbol{x}^k$
            break
        End If

        Constructing matrix $\boldsymbol{A}^k$ and vector $\boldsymbol{b}^k$ according to $\boldsymbol{x}^k$, $\mathbf{z}^*$, and $\varepsilon$
        If $|\boldsymbol{A}^k| = 0$ then
            break
        End If
        Constructing matrices $\boldsymbol{A}_j^k$ according to $\boldsymbol{A}^k$ and $\boldsymbol{b}^k$
        $\boldsymbol{x}^{k+1} = \frac{1}{|\boldsymbol{A}^k|}(|\boldsymbol{A}_1^k|, |\boldsymbol{A}_2^k|, \ldots, |\boldsymbol{A}_n^k|)^T$
        $k = k + 1$
    While (true)

    If $\mathbf{z}^0 = +\infty$ then // If no descending point is found
        $\mathbf{z}^{**} = \mathbf{z}^*$
        break
    Else
        $i = i + 1$
        $\mathbf{z}^i = \mathbf{z}^0$
    End If

While ($i < M$) //prevent infinite loop when there is no global maximum
```

4. Choosing Optimal Starting Point for Solving Intersection Equation

Given (1), the problem needs solved is to select the starting point $\boldsymbol{x}^0$ so that such point is in the volume where SNR algorithm converges. The volume is defined as the sub-space or sub-set, denoted $v(\boldsymbol{a}, \boldsymbol{b})$ or $[\boldsymbol{a}, \boldsymbol{b}]$.

$$v(\boldsymbol{a}, \boldsymbol{b}) = [\boldsymbol{a}, \boldsymbol{b}] = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$$

The *capacity* of volume is defined by following formula:

$$c(\boldsymbol{a}, \boldsymbol{b}) = \prod_{i=1}^{n}(b_i - a_i)$$

Volume can be an interval $v(\boldsymbol{a}, \boldsymbol{b}) = [a, b]$ in $\mathbb{R}$, a rectangle $v(\boldsymbol{a}, \boldsymbol{b}) = [a_1, b_1] \text{ x } [a_2, b_2]$ in $\mathbb{R}^2$ or a volume $v(\boldsymbol{a}, \boldsymbol{b}) = [a_1, b_1] \text{ x } [a_2, b_2] \text{ x } [a_3, b_3]$ in $\mathbb{R}^3$. Note that the volume is infinite volume if any of its bound is infinite, for example:

$$v(\boldsymbol{a}, \boldsymbol{b}) = [\boldsymbol{a}, \boldsymbol{b}] = [-\infty, b_1] \times [a_2, b_2] \times \ldots \times [a_n, +\infty]$$

If the volume is infinite volume, its capacity is positive infinite, $c(\boldsymbol{a}, \boldsymbol{b}) = +\infty$.

Solution volume is defined as the volume in which solution of (1) is existent. It is easy to recognize that $[\boldsymbol{a}, \boldsymbol{b}]$ is the solution volume of function f if and only if there exist two values $\boldsymbol{x}$ and $\boldsymbol{y}$ belonging to $[\boldsymbol{a}, \boldsymbol{b}]$ such that $f(\boldsymbol{x})f(\boldsymbol{y}) \leq 0$ according to Bolzano-Cauchy's theorem [6].

$$[\boldsymbol{a}, \boldsymbol{b}] \text{ solution volume} \Leftrightarrow \exists \boldsymbol{x}, \boldsymbol{y} \in [\boldsymbol{a}, \boldsymbol{b}]: f(\boldsymbol{x})f(\boldsymbol{y}) \leq 0 \quad (5)$$

The aforementioned problem is solved by determining the solution volume [***a***, ***b***] such that [***a***, ***b***] is as small as possible. In other words, given pre-defined volume [***a***, ***b***], what we need to do is to find out the sub-volume [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] so that it satisfies three conditions:

1. It is in [***a***, ***b***]; in other words, [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] ⊆ [***a***, ***b***].
2. It is also solution volume, satisfying (5).
3. It is as small as possible. This condition helps SNR algorithm to converge as fast as possible.

Such sub-volume is called *optimal volume*. Because it is impossible to determine optimal volume by calculating exhaustedly $f(\boldsymbol{x})$ for all ***x*** in [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$], I propose a so-called *random testing point* (RTP) algorithm to find out optimal volume. Firstly, suppose ***x*** and ***y*** are points in [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] such that $f(\boldsymbol{x}) > 0$ and $f(\boldsymbol{y}) < 0$, respectively. Hence, ***x*** and ***y*** are called positive point and negative point, respectively. Let N_+^i and N_-^i be the number of positive and negative points, respectively and let p^i be the ratio of N_-^i to the number of total points N^i in optimal volume [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$], according to (6).

$$p^i = \frac{N_-^i}{N_+^i + N_-^i} = \frac{N_-^i}{N^i} \tag{6}$$

Where $N^i = N_+^i + N_-^i$ is the number of total points and so, p^i is the probability of occurrence of negative points in optimal volume [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$]. RTP algorithm to find out optimal volume is based on two heuristic assumptions:

- If [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] is optimal volume, then the probability $0 < p^i < 1$, in other words, both $N_+^i > 0$ and $N_-^i > 0$.
- The nearer to ½ the probability p^i is, the more likely it is that [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] is optimal volume.

RTP algorithm is iterative algorithm whose input is volume [***a***, ***b***] and output is optimal volume [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$]. Given volume [***a***, ***b***] = [a_1, b_1] x [a_2, b_2] x … x [a_n, b_n] is divided into $n*n$ sub-volumes [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] = [a_1^i, b_1^i] x [a_2^i, b_2^i] x … x [a_n^i, b_n^i] where $i = \overline{1,m}$. RTP algorithm has finitely many iterations. We do two tasks at each iteration:

1. Creating many enough random points in each volume [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$].
2. Counting the number of positive and negative points, N_+^i and N_-^i and calculating the probability p^i of each sub-volume [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] based on N_+^i and N_-^i. Which sub-volume that has probability p^i being larger than 0 and smaller than 1 and nearest to ½ is chosen to be the input for next iteration.

There are two stopped conditions of RTP algorithm:

1. The deviation between probability p^i and ½ or the capacity $c(\boldsymbol{a}^i, \boldsymbol{b}^i)$ is smaller than a small pre-defined number δ.
2. Or, the algorithm reaches the maximum number of iterations.

When the optimal volume is determined, the optimal starting point $\boldsymbol{x}^0$ is any point in such volume. Of course, if there is a random point $\boldsymbol{c}^i$ ∈ [$\boldsymbol{a}^i$, $\boldsymbol{b}^i$] such that $f(\boldsymbol{c}^i) = 0$, then we have $\boldsymbol{x}^0 = \boldsymbol{c}^i$. In general, Table 4 shows the pseudo-code for RTP algorithm whose input is volume [***a***, ***b***] and output is optimal starting point $\boldsymbol{x}^0$.

Table 4. RTP algorithm to find out optimal starting point.

```
//Initialization
Creating many enough random points in [a, b]
$N^-$:= the number of positive points in [a, b]
$N^+$:= the number of negative points in [a, b]
If $N^- > 0$ and $N^+ > 0$ then
    [$a^*$, $b^*$] = [a, b]
Else
    [$a^*$, $b^*$] = Ø
End If
$\delta$:= small pre-defined number
$M$:= the maximum number of iterations
$k = 0$

Loop
    Partitioning [a, b] into $n*n$ sub-volumes [$a^i$, $b^i$]
    min_value = +∞
    [$a^{min}$, $b^{min}$] = Ø

    For each [$a^i$, $b^i$] in [a, b]
        Creating many enough random points in [$a^i$, $b^i$]
        Counting the number of positive and negative points, $N_+^i$ and $N_-^i$
        Calculating $p^i$ according to (6)
        $value = p^i - \frac{1}{2}$ or $value = c(a^i, b^i)$
        If $0 < p^i < 1$ and min_value ≥ value then
            min_value = value
            [$a^{min}$, $b^{min}$] = [$a^i$, $b^i$]
        End If
    End For

    If [$a^{min}$, $b^{min}$] ≠ Ø then
        [$a^*$, $b^*$] = [$a^{min}$, $b^{min}$]
        [a, b] = [$a^*$, $b^*$]
    Else
        break
    End If

    $k = k + 1$
While ($k < M$ and min_value ≥ $\delta$)

If [$a^*$, $b^*$] ≠ Ø then
    $x^0$:= any point in volume [$a^*$, $b^*$]
End If
```

The essence of RTP algorithm is to narrow solution volume according to horizontal axis; similarly, the essence of DR algorithm is to narrow the descending region according to vertical axis. If the descending region is also reduced according horizontal axis, DR algorithm will converge faster. Suppose in horizontal axis, DR algorithm begin seeking local optimizers in given initial volume [***a***, ***b***], such volume is called *searching volume*, which is reduced after each iteration. If the current local optimizer is $\boldsymbol{x}^*$ then, the next searching volume is [$\boldsymbol{x}^*$, ***b***] with assumption that the global optimizer leans only forward. In other words, the next descending point $\mathbf{z}^0$ is searched in [$\boldsymbol{x}^*$, ***b***] according to horizontal axis and below $f(\boldsymbol{x}^*)$ according to vertical axis. Of course, the solution volume of RTP algorithm to find out optimal starting point is [$\boldsymbol{x}^*$, ***b***]. In general, the descending region is reduced according to both horizontal axis and vertical axis. Back the example in section 2.2, given target function,

$$f(x) = \frac{1}{16}x^4 - \frac{1}{12}x^3 - \frac{3}{4}x^2 - \frac{2}{3}$$

Fig. 3 depicts how to find out global optimizer of target function by the improved DR algorithm. In Fig. 3, two descending regions are shaded areas. It is easy to recognize that the descending areas are narrowed according to two axes.

The pseudo-code for DR algorithm is improved as seen in Table 5 with note that the input is an arbitrary point $\boldsymbol{\omega}^0$ and searching volume [$\boldsymbol{a}$, $\boldsymbol{b}$] and the output is global optimizer $\mathbf{z}^{**}$ with note that [$\boldsymbol{a}$, $\boldsymbol{b}$] can be infinite volume.

Table 5. *Improved DR algorithm with optimal starting point.*

```
//Initialization
$\mathbf{z}^0 = \boldsymbol{\omega}^0 \in [\boldsymbol{a}, \boldsymbol{b}]$
$\mathbf{z}^{**} = \mathbf{z}^* = +\infty$
$\varepsilon$:= very small pre-defined number
$M$:= the maximum number of iterations
$i = 0$

Loop
    //Step 1: Searching for local optimizer
    $\mathbf{z}^*$:= searching for local optimizer with the input $\mathbf{z}^i$
    If $\mathbf{z}^*$ not found then
        break
    Else If $\mathbf{z}^*$ equal previous optimizer then
        $\mathbf{z}^{**} = \mathbf{z}^*$
        break
    End If

    //Step 2: Determining descending region
    $\mathbf{z}^0 = +\infty$
    $[\boldsymbol{a}, \boldsymbol{b}] = [\mathbf{z}^*, \boldsymbol{b}]$ or $[\boldsymbol{a}, \boldsymbol{b}] = [\boldsymbol{a}, \mathbf{z}^*]$ if global optimizer leans only forward or
    only backward, respectively. Otherwise, do nothing.

    $\boldsymbol{x}^0$:= choosing optimal starting point with input $[\boldsymbol{a}, \boldsymbol{b}]$ by RTP algorithm.
    $\mathbf{z}^0$:= solution of intersection equation with inputs $\boldsymbol{x}^*$, $\varepsilon$, and $\boldsymbol{x}^0$ by SNR
    algorithm.

    If $\mathbf{z}^0 = +\infty$ then // If no descending point is found
        $\mathbf{z}^{**} = \mathbf{z}^*$
        break
    Else
        $i = i + 1$
        $\mathbf{z}^i = \mathbf{z}^0$
    End If
While ($i < M$)
```

5. Conclusion

The essence of DR algorithm is to solve the linear system equation (4) many enough times, which aims to solve intersection equation (1). In other words, the hazard problem of global optimization is turned back the most common problem with note that linear equation system is always solvable. However, that SNR algorithm to solve (1) via (4) is a simulation of Newton – Raphson method causes a new problem of convergence. This is the weak point of the research that I alleviate by RTP algorithm based on partitioning solution space and generating random testing points in order to find out optimal starting point for fast convergence. Currently, I cannot research out a new method better than SNR algorithm. Therefore DR algorithm will be improved significantly if we can predict that (1) has no solution before solving it or we can predict the solution volume of (1). Sturm's theorem mentioned in section 3 is a good prediction tool but it is only applied into the case that (1) is a polynomial. RTP algorithm with random point generation is also not an optimal tool. However, for my opinion, the prediction approach is potential. In the future, I will research deeply how to approximate (1) into simpler forms such as exponent function and polynomial in order to apply easily prediction tools. For example, (1) can be easily approximated by Taylor polynomial. The smaller the searching volume is, the more accurate the Taylor polynomial is. If the volume is one-dimension interval, optimal degree of Taylor polynomial is equal to or larger than the length of such interval according to [17]. Alternately, Taylor polynomial can be also optimized by minimizing square error according to [18]. I suggest a so-called *segmentation approach* in which the solution volume is split into many small enough segments. Later on, for each segment, approximation methods such as feasible length and minimizing square error are applied to approximate (1) by a Taylor polynomial in such segment so that it is accurate to predict solution volume of such polynomial. Final solution volume of (1) is the best one from many polynomials over all segments. This approach shares the same ideology of volume partitioning with RTP algorithm. It can be more complicated but better than RTP.

There is a question: Can the target function be approximated by polynomial so that it is easy to find out the global optimizer on such target polynomial? The answer is that such global optimizer is imprecise because the target polynomial is an approximation of f which is much varied when domain space of f is large enough. However it is acceptable if we use polynomial approximation to only predict solution volume of (1) and then, use SNR algorithm to find out exactly the descending point $\mathbf{z}^0$.

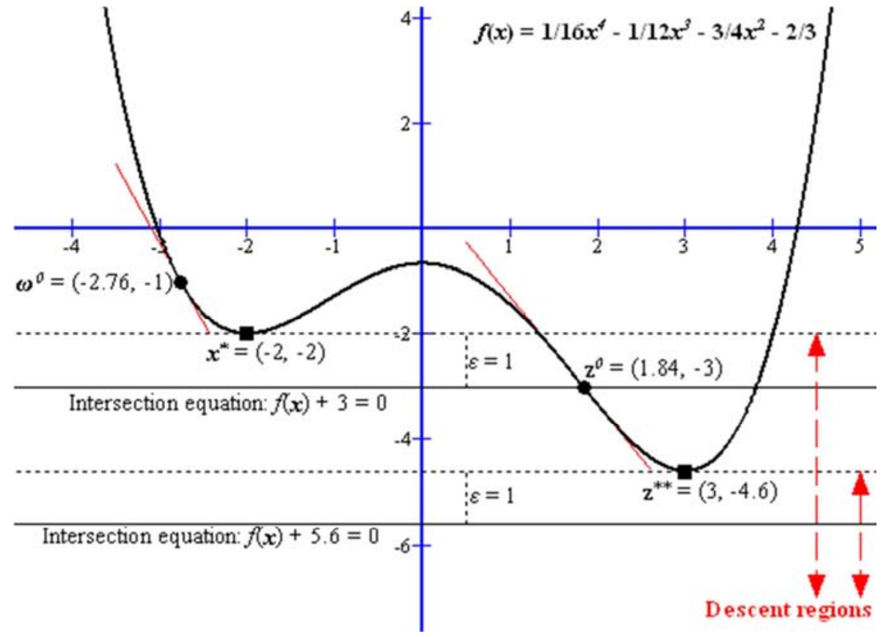

Figure 1. *An example of DR algorithm.*

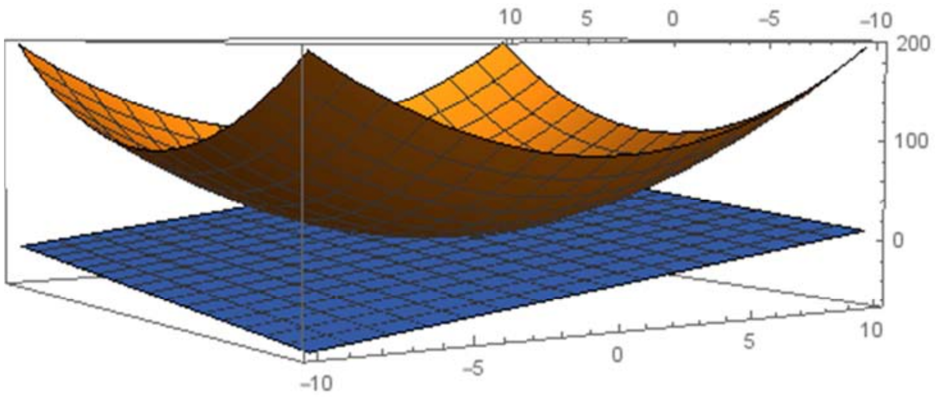

Figure 2. *An example of SNR algorithm.*

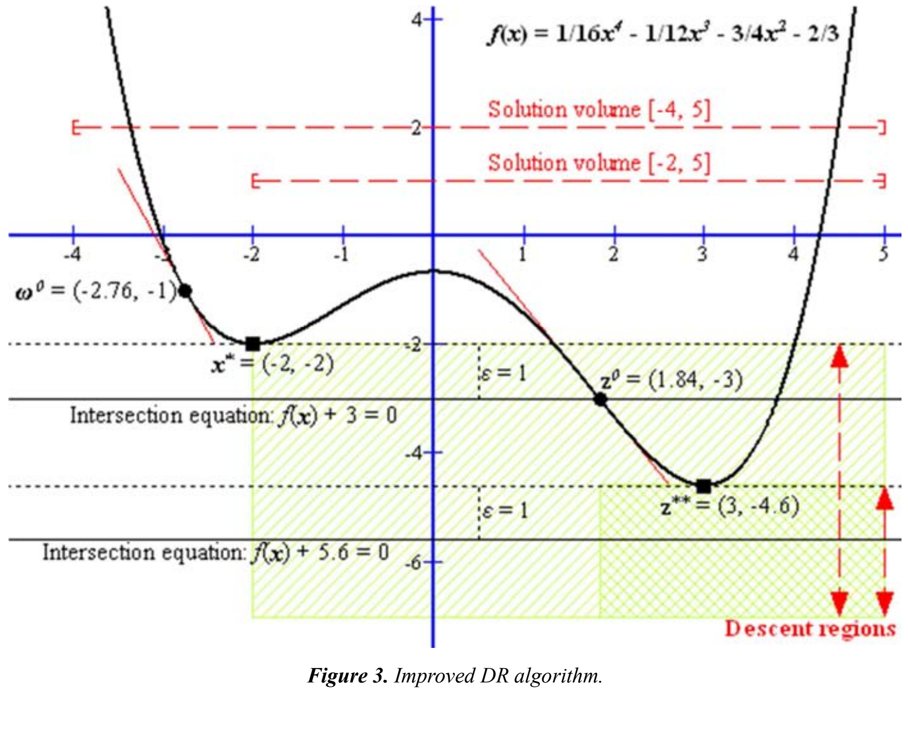

Figure 3. *Improved DR algorithm.*

Acknowledgements

I express my deep gratitude to Professor Le, Dung-Muu and Professor Ta, Duy-Phuong – Vietnam Institute of Mathematics who reviewed and gave me valuable advices to improve this research.

Nomenclature

DR: descending region
GD: gradient descent
RTP: random testing point
SNR: simulated Newton – Raphson

References

[1] M. D. Le and Y. H. Le, "Lecture Notes on Optimization," Vietnam Institute of Mathematics, Hanoi, 2014.

[2] S. Boyd and L. Vandenberghe, Convex Optimization, New York, NY: Cambridge University Press, 2009, p. 716.

[3] Y.-B. Jia, "Lagrange Multipliers," 2013.

[4] A. P. Ruszczyński, Nonlinear Optimization, Princeton, New Jersey: Princeton University Press, 2006, p. 463.

[5] Wikipedia, "Karush–Kuhn–Tucker conditions," Wikimedia Foundation, 4 August 2014. [Online]. Available: http://en.wikipedia.org/wiki/Karush–Kuhn–Tucker_conditions. [Accessed 16 November 2014].

[6] P. D. Ta, "Numerical Analysis Lecture Notes," Vietnam Institute of Mathematics, Hanoi, 2014.

[7] T. Hoang, Convex Analysis and Global Optimization, Dordrecht: Kluwer, 1998, p. 350.

[8] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in Proceedings of IEEE International Conference on Neural Networks, 1995.

[9] Wikipedia, "Particle swarm optimization," Wikimedia Foundation, 7 March 2017. [Online]. Available: https://en.wikipedia.org/wiki/Particle_swarm_optimization. [Accessed 8 April 2017].

[10] R. Poli, J. Kennedy and T. Blackwell, "Particle swarm optimization," Swarm Intelligence, vol. 1, no. 1, pp. 33-57, June 2007.

[11] Wikipedia, "Quasi-Newton method," Wikimedia Foundation, 4 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Quasi-Newton_method. [Accessed 8 April 2017].

[12] H. Jiao, Z. Wang and Y. Chen, "Global optimization algorithm for sum of generalized polynomial," Applied Mathematical Modelling, vol. 37, no. 1-2, pp. 187-197, 18 February 2012.

[13] T. Larsson and M. Patriksson, "Global optimality conditions for discrete and nonconvex optimization - With applications to Lagrangian heuristics and column generation," Operations Research, vol. 54, no. 3, pp. 436-453, 21 April 2003.

[14] P. Dawkins, "Gradient Vector, Tangent Planes and Normal Lines," Lamar University, 2003. [Online]. Available: http://tutorial.math.lamar.edu/Classes/CalcIII/GradientVectorTangentPlane.aspx. [Accessed 2014].

[15] V. H. H. Nguyen, Linear Algebra, Hanoi: Hanoi National University Publishing House, 1999, p. 291.

[16] R. L. Burden and D. J. Faires, Numerical Analysis, 9th Edition ed., M. Julet, Ed., Brooks/Cole Cengage Learning, 2011, p. 872.

[17] L. Nguyen, "Feasible length of Taylor polynomial on given interval and application to find the number of roots of equation," International Journal of Mathematical Analysis and Applications, vol. 1, no. 5, pp. 80-83, 10 January 2015.

[18] L. Nguyen, "Improving analytic function approximation by minimizing square error of Taylor polynomial," International Journal of Mathematical Analysis and Applications, vol. 1, no. 4, pp. 63-67, 21 October 2014.

[19] Wikipedia, "Sturm's theorem," Wikimedia Foundation, 2014. [Online]. Available: https://en.wikipedia.org/wiki/Sturm%27s_theorem. [Accessed 30 August 2014].