Leap Gradient Algorithm

Sergey Nikitin

May 23, 2014

Abstract
Leap gradient algorithm is able to solve global optimization problems. It does not require any convexity from an optimization problem. However, if a target function is smooth than after performing (if necessary) several leaps the algorithm naturally becomes the standard gradient descent (or ascent) near the global extremum. Moreover, under some generic conditions the paper presents an efficient recursive numerical procedure for calculating the global extremum of polynomials.

Keywords: Gradient descent, Optimization, Power series, Recursive algorithm, Polynomial

1 Introduction
Gradient descent method is widely used to solve various practical optimization problems. Its main idea can be briefly outlined as follows. In order to solve

$$f(x) \rightarrow \min$$ (1)

one first needs to choose an initial candidate $x_0$ for the solution and then iteratively improve it with

$$x_{k+1} = -\alpha \cdot \nabla f(x_k) + x_k$$ (2)

where

$$\nabla f(x) = \left( \frac{\partial}{\partial x_1} f(x), \frac{\partial}{\partial x_2} f(x), \ldots, \frac{\partial}{\partial x_n} f(x) \right)$$

is the gradient of $f(x)$ and the step-size $\alpha > 0$ is a number yet to be determined.

There are many approaches to choose the step-size $\alpha > 0$. The problem is to choose the step-size large enough so that the iterative procedure (2) does not stall and at the same time small enough so that it is convergent. In order to expose the underlying difficulty associated with this choice let us assume for a moment that our target function $f(x)$ admits the second derivatives and

$$\Delta x_k = x_k - x_{k-1}.$$ 

Subtracting the next equation (3) from (2)

$$x_k = -\alpha \cdot \nabla f(x_{k-1}) + x_{k-1}$$ (3)
yields
\[ \Delta x_{k+1} = -\alpha (\nabla f(x_k) - \nabla f(x_{k-1})) + \Delta x_k \]

Let
\[ |\Delta x_{k+1}|^2 = \Delta x_{k+1} \cdot \Delta x_{k+1} \]
denote the dot product of \( \Delta x_{k+1} \) with itself. Consider the real valued function
\[ \phi(t) = | -\alpha (\nabla f(t \cdot \Delta x_k + x_{k-1}) - \nabla f(x_{k-1})) + t \cdot \Delta x_k |^2 \]

Since
\[ \phi(0) = 0 \]
and
\[ \phi(1) = | -\alpha (\nabla f(x_k) - \nabla f(x_{k-1})) + \Delta x_k |^2 \]
in accordance with the calculus mean value theorem there exists a number
\[ 0 < t^* < 1 \]
such that
\[ |\Delta x_{k+1}|^2 = | -\alpha (\nabla f(x_k) - \nabla f(x_{k-1})) + \Delta x_k |^2 = \phi(1) - \phi(0) = \frac{d}{dt} \phi(t^*) \]

On the other hand,
\[ \frac{d}{dt} \phi(t^*) = 2(-\alpha (\nabla f(x_k) - \nabla f(x_{k-1})) + t^* \cdot \Delta x_k) \cdot ((I - \alpha \frac{\partial^2}{\partial x^2} f(x_k^*)) \Delta x_k) \]
where
\[ x_k^* = t^* \cdot \Delta x_k + x_{k-1} \]

\( I \) denotes the identity matrix and \( \frac{\partial^2}{\partial x^2} f \) is the matrix of the second derivatives for \( f \).

Now let us introduce the function
\[ \psi(\tau) = 2(-\alpha (\nabla f(\tau \cdot \Delta x_k + x_{k-1}) - \nabla f(x_{k-1})) + \tau \cdot t^* \cdot \Delta x_k) \cdot ((I - \alpha \frac{\partial^2}{\partial x^2} f(x_k^*)) \Delta x_k) \]

Notice that
\[ \psi(1) = \frac{d}{dt} \phi(t^*). \]

It follows from the calculus mean value theorem that there exists \( 0 < \tau^* < 1 \) such that
\[ \psi(1) - \psi(0) = \frac{d}{d\tau} \psi(\tau^*) \]
Since \( \psi(0) = 0 \) we have
\[ |\Delta x_{k+1}|^2 = \frac{d}{dt} \phi(t^*) = \frac{d}{d\tau} \psi(\tau^*) \]
and consequently

\[ |\Delta x_{k+1}|^2 = 2 \cdot t^* \left( (I - \alpha \frac{\partial^2 f(x_k^{**})}{\partial x^2}) \Delta x_k \right) \cdot \left( (I - \alpha \frac{\partial^2 f(x_k^*)}{\partial x^2}) \Delta x_k \right) \]

where

\[ x_k^{**} = \tau \cdot t^* \cdot \Delta x_k + x_{k-1} \]

The formula (4) gives us a recipe for choosing the step-size \( \alpha \). For example, \( \alpha > 0 \) has to be chosen large enough so that the iterative procedure (2) does not stall and at the same time

\[ \max_{0 < r < 1} \| I - \alpha \frac{\partial^2 f(t \cdot \Delta x_k + x_{k-1})}{\partial x^2} \| < \delta < 1 \]

where \( \delta > 0 \) is a real number and \( \|A\| \) stays for a norm of a linear operator. More detailed elaboration on this account one can find in the literature [6], [8], [19].

Though (4) seems to give a complete answer to the question about the step-size \( \alpha \) its practical applications are limited due to numerical difficulties. Instead, in many applications \( \alpha > 0 \) is calculated with one of the following rules: constant step-length rule (\( \nabla f \) is a Lipschitz function), Armijo’s rule [1], Goldsteun’s approach [5], Wolfe’s conditions [15], [16]. Their detailed descriptions and further references can be found in the books [6], [11], [12]. More recent achievements of calculating the step-size are presented, for example, in papers [3], [13] and [19].

The main advantage of the gradient descent algorithm is its simplicity in realizations for wide range of practical problems. On the other hand its main disadvantage is in limitations imposed by the initial guess of a starting point and then its subsequent conversion to a suboptimal solution. This paper gives practical recipes on how to overcome those limitations and how to equip the gradient descent algorithm with abilities to converge to a global extremum. It is achieved via evolutionary leaps towards the global extremum. The new proposed leap gradient algorithm (referred in this paper as LGA) does not need any convexity conditions that are often imposed on the target function. Moreover, the leap gradient algorithm naturally merges into the standard gradient descent procedure when the target function is convex or when the algorithm operates in the close proximity to the global extremum.

The recursive application of LGA yields an algorithm for calculating extrema for polynomials of one and/or several variables. The main difference of the newly proposed recursive LGA from the well established Sturm’s procedure [10], [14], [17], [19] is that LGA does not intend to locate any polynomial roots instead its aim is to establish the point of extremum by means of small gradient descent (ascent) steps and possible evolutionary leaps. Moreover, LGA extends naturally to multivariable polynomials while other methods need special modifications and certain restrictions on the polynomials [7], [13]. As far as performance concerned, LGA is expected to outperform Sturm’s method for polynomials with sufficiently large number of roots. Another approach for calculating extrema of polynomials is the topping algorithm [2]. The topping method reduces the complexity of the extremum problem by considering its restrictions onto subsets of smaller dimensions, in particular the \((n-1)\)-dimensional sphere in \( \mathbb{R}^n \). The scope of the method could be larger than just extrema of polynomials. It can also handle some classes of rational functions. However, performance of the
method will depend on special properties of polynomials and their restrictions on to the respective subsets. LGA is supposed to outperform the topping algorithm when applied to polynomials on multidimensional parallelepipeds. For more general domains LGA needs modifications. For other functions (e.g., with fast convergent expansions in spherical harmonics) the LGA performance depends on the method of polynomial approximation for the function under consideration and the topping approach might have advantages. In conclusion, note that LGA together with the method of topping functions [2] promises to lead us to exhaustive solutions for many practical optimizations problems.

2 Leaps towards the global extremum

It is well known from a calculus course how to calculate a global extremum for a real function of one variable on an interval \([a, b]\) of real numbers, \(\mathbb{R}\). Now let us describe an iterative procedure that will lead us to the global minimum for \(f(x)\) on \([a, b]\).

Consider the optimization problem

\[
f(x) \rightarrow \min_{x \in [a, b]} \]

The initial state \(x_0\) of our iterative procedure is taken to be equal to \(a\), the left end of the interval \([a, b]\).

\[x_0 = a.\]

Suppose we performed already \(k\) steps of the procedure and arrived at \(x_k\). Then the next value \(x_{k+1}\) depends on the solution of the optimization problem

\[
\frac{f(x) - f(x_k)}{x - x_k} \rightarrow \min_{x \in [x_k, b]} \]

Let \(x_k^\star\) be the solution of (5). Then the next iteration is based on the following.

**Leap Gradient Algorithm (LGA).**

- If \(x_k^\star = x_k\) and the right derivative

\[
\lim_{x \to x_k^+} \frac{f(x) - f(x_k)}{x - x_k} \geq 0,
\]

then the iterative procedure stops at \(x_k\) and \(f(x)\) reaches its global minimum on \([a, b]\) at \(x_k\).

- If \(x_k^\star > x_k\) and either

\[
\frac{f(x_k^\star) - f(x_k)}{x_k^\star - x_k} \geq 0,
\]

or

\[x_k^\star = b\]

then the iterative procedure stops at \(x_k^\star\) and \(f(x)\) reaches its global minimum on \([a, b]\) at \(x_k\).
Figure 1: Leap Gradient Algorithm for a function of one variable

- If \( x^*_k = x_k \) and the right derivative
  \[
  \lim_{x \to x_k^+} \frac{f(x) - f(x_k)}{x - x_k} < 0,
  \]
  then we follow the standard gradient descent procedure and make a small step to the right, \( x_{k+1} = x_k + h \), where \( h \) is the gradient descent step-size.

- If \( x^*_k > x_k \) and
  \[
  \frac{f(x^*_k) - f(x_k)}{x^*_k - x_k} < 0,
  \]
  then we perform the evolutionary leap and take \( x_{k+1} = x^*_k \).

The leap gradient procedure is illustrated in Fig. 1.

If the target function is one time differentiable on \((a, b)\) then its evolutionary leaps (from \((a, b)\)) are solutions of the following constrained optimization problem.

\[
\frac{d}{dx} f(x) \rightarrow \min_{x \in Q} \quad (6)
\]

where

\[
Q = \{ x; f(x) = \frac{d}{dx} f(x) \cdot (x-x_k) + f(x_k), \quad (x-x_k) \cdot \frac{d}{dx} f(x) \leq 0 \quad \text{and} \quad x \in (x_k, b) \} \quad (7)
\]

and \( x_k \) is the respective iteration of the leap gradient algorithm.

Indeed, if \( f(x) \) is differentiable then the necessary condition of \( x \) to be the solution for (5) is

\[
\frac{d}{dx} \left( \frac{f(x) - f(x_k)}{x - x_k} \right) = 0.
\]
Differentiating yields

\[ \frac{d}{dx} \frac{f(x)}{x - x_k} - \frac{f(x) - f(x_k)}{(x - x_k)^2} = 0. \]

After multiplying the equation with \((x - x_k)^2\) we obtain

\[ (x - x_k) \frac{d}{dx} f(x) - (f(x) - f(x_k)) = 0 \]

and the necessary condition for an evolutionary leap to occur at \(x \in (x_k, b)\) follows

\[ f(x) = (x - x_k) \cdot \frac{d}{dx} f(x) + f(x_k). \quad (8) \]

Hence, (5) takes the form (6), (7).

Let \(x^*\) be an evolutionary leap from \((x_k, b)\). Consider the function

\[ \varphi(t) = t(x^* - x_k) \cdot \frac{d}{dx} f(t(x^* - x_k) + x_k) + f(x_k) - f(t(x^* - x_k) + x_k) \]

It follows from (8) that

\[ \varphi(1) = 0 \]

and \(\varphi(0) = 0\). By Rolle's Theorem from calculus we conclude that there exists

\[ 0 < t^* < 1 \]

such that

\[ \frac{d}{dt} \varphi(t^*) = 0. \]

That yields

\[ \left( \frac{d}{dx} \right)^2 f(t^*(x^* - x_k) + x_k)(x^* - x_k)^2 = 0. \]

That means a non trivial evolutionary leap is only possible over an inflection point. In fact, this will also take place for multivariable target functions described in the next section.

**Example**

Consider the optimization problem

\[ f(x) = x^4 + a \cdot x^3 + b \cdot x^2 + c \cdot x + d \rightarrow \min_{x \in \mathbb{R}} \]

where \(a, b, c, \) and \(d\) are real numbers. Let us describe the set of points available for evolutionary leaps. Since the target function \(f(x)\) is a fourth degree polynomial the Tailor expansion centered at \(x\) and evaluated at \(x_k\) yields

\[ f(x_k) = f(x) + \frac{d}{dx} f(x) \cdot (x_k - x) + \frac{d^2}{dx^2} f(x) \cdot \frac{(x_k - x)^2}{2!} + \]

\[ + \frac{d^3}{dx^3} f(x) \cdot \frac{(x_k - x)^3}{3!} + \frac{d^4}{dx^4} f(x) \cdot \frac{(x_k - x)^4}{4!} \]
This formula together with (8) implies that a non-trivial evolutionary leap $x \neq x_k$ satisfies the following condition.

$$\frac{d^2}{dx^2} f(x) \cdot \frac{1}{2!} + \frac{d^3}{dx^3} f(x) \cdot \frac{(x_k - x)}{3!} + \frac{d^4}{dx^4} f(x) \cdot \frac{(x_k - x)^2}{4!} = 0$$

Replacing in this formula $f(x)$ with the fourth degree polynomial gives us the following quadratic equation for non-trivial leaps.

$$3 \cdot x^2 + 2(x_k + a) \cdot x + x_k^2 + ax_k + b = 0$$

This equation has real roots only when

$$2x_k^2 + ax_k + 3b - a^2 \leq 0$$

In turn, the latter has real roots only when

$$3a^2 \geq 8b \quad (10)$$

Hence, leap gradient procedure for the target function (9) might have non-trivial evolutionary leaps if (10) is in place. Otherwise, the leap gradient procedure coincides with the standard gradient descent algorithm. One can also obtain (10) by looking at the second derivative of the target function,

$$\frac{d^2}{dx^2} f(x) = 4 \cdot 3 \cdot x^2 + 3 \cdot 2 \cdot a \cdot x + 2 \cdot b$$

The inequality (10) guarantees the existence of the inflection points for the target function, and as we pointed earlier the non-trivial evolutionary leap is possible only over an inflection point. In this simple example the presence of inflection points is a guarantee for the feasibility of non-trivial evolutionary leaps.

### 3 Leap gradient algorithm for multi-variable functions

The leap gradient algorithm presented in the previous section for functions of one variable essentially follows the same scenario in order to calculate the global minimum for a multivariable function.

Consider the optimization problem

$$f(x) \rightarrow \min_{x \in Q \subset \mathbb{R}^n}$$

(11)

where $Q$ is a compact connected subset of $n$-dimensional real space $\mathbb{R}^n$. It is assumed throughout this paper that $\mathbb{R}^n$ is equipped with the scalar product

$$x \cdot y = \sum_{k=1}^{n} x_k \cdot y_k \quad \text{for} \quad x, y \in \mathbb{R}^n$$

and $|x|$ denotes the absolute value of $x \in \mathbb{R}^n$,

$$|x| = \sqrt{x \cdot x}.$$
Let $x_0$ be the initial guess for the solution of (11) and $x_k$ be its improvement after $k$ iterations. Let $x^*_k$ be the solution of the following optimization problem.

$$\frac{f(x) - f(x_k)}{|x - x_k|} \rightarrow \min_{x \in Q \subset \mathbb{R}^n}$$ (12)

Then the next iteration $x_{k+1}$ is based on the following.

**Leap Gradient Algorithm (LGA).**

- If $x^*_k = x_k$ and
  \[ \lim_{t \to 0^+, \ x_k + t \cdot h \in Q} \frac{f(x_k + t \cdot h) - f(x_k)}{t} \geq 0, \]
  for any unit vector $h \in \mathbb{R}^n$, then the iterative procedure stops at $x_k$ and $f(x)$ reaches its global minimum on $Q$ at $x_k$.

- If $x^*_k \neq x_k$ and
  \[ \frac{f(x^*_k) - f(x_k)}{|x^*_k - x_k|} \geq 0, \]
  then the iterative procedure stops at $x^*_k$ and $f(x)$ reaches its global minimum on $Q$ at $x_k$.

- If $x^*_k = x_k$ and there exists a unit vector $h \in \mathbb{R}^n$ such that
  \[ \lim_{t \to 0^+, \ x_k + t \cdot h \in Q} \frac{f(x_k + t \cdot h) - f(x_k)}{t} < 0, \]
  then we calculate $x_{k+1}$ in accordance with the gradient descent procedure.

- If $x^*_k \neq x_k$ and
  \[ \frac{f(x^*_k) - f(x_k)}{|x^*_k - x_k|} < 0, \]
  then we perform the evolutionary leap and take $x_{k+1} = x^*_k$.

In various applications of LGA it is sometimes useful to use modifications of (12). For example, as discussed later in this paper, for polynomials it is convenient to replace (12) with

\[
\frac{f(x) - f(L_{jx_k}(x))}{Pr_j(x - x_k)} \rightarrow \min_{x \in Q \subset \mathbb{R}^n}
\]

or with

\[
\frac{f(x) - f(L_{jx_k}(x))}{|Pr_j(x - x_k)|} \rightarrow \min_{x \in Q \subset \mathbb{R}^n}
\]

where $|x| = \max\{x, -x\}$, $Pr_j$ is the projection onto the $j$-th axes of coordinates,

$$Pr_j(x) = x_j$$
and
\[ L_{j_{k}}(x) = (x_1, \ldots, x_{j-1}, x_{j_k}, x_{j+1}, \ldots, x_n) \]
where \( x_{j_k} \) is the \( j \)-th entry of \( x_k \). In other words, \( L_{j_{k}}(x) \) is the projection onto the plane defined as
\[ x_j = x_{j_k}. \]

For smooth functions the next statement contains necessary conditions for \( y \in \mathbb{R}^n \) to be an evolutionary leap.

**Theorem 1** Let \( f(x) \) be a smooth function. If \( y \in \mathbb{R}^n \) is an evolutionary leap from \( x \in \mathbb{R}^n \) then the gradient \( \nabla f(y) \) is parallel to \( y - x \). Moreover,
\[ \nabla f(y) = \frac{f(y) - f(x)}{|y - x|}, \frac{(y - x)}{|y - x|} \]  
(13)

**Proof.**

An evolutionary leap \( y \in \mathbb{R}^n \) from \( x \) provides the minimum for
\[ \frac{f(y) - f(x)}{|y - x|}. \]
Therefore
\[ \nabla_y \left( \frac{f(y) - f(x)}{|y - x|} \right) = 0 \]
Calculating the gradient yields
\[ \frac{1}{|y - x|} \cdot \nabla f(y) - \frac{f(y) - f(x)}{|y - x|^3} (y - x) = 0 \]
and the necessary condition follows.

**Q.E.D.**

LGA for the optimization problem
\[ f(x) \to \min_{x \in Q \subset \mathbb{R}^n} \]
generates the sequence
\[ x_1, x_2, \ldots, x_k, x_{k+1}, \ldots \]  
(14)
such that
\[ f(x_1) > f(x_2) > \ldots > f(x_k) > f(x_{k+1}) > \ldots \]  
(15)
is strictly monotonically descending. Moreover, if \( x_{k+1} \) is an evolutionary leap from \( x_k \) then
\[ |x_q - x_k| > |x_{k+1} - x_k| \]
for all \( q > k + 1 \).

It directly follows from (13) and (15) that
\[ \nabla f(x_k) \neq 0 \]
if \( x_k \) is an evolutionary leap. It implies the feasibility of the switch to the gradient descent method after each evolutionary leap. Moreover, it yields the following statement.
**Theorem 2** Let \( Q \subseteq \mathbb{R}^n \) be compact with a smooth boundary \( \partial Q \). Let \( f(x) \) be continuously differentiable. Then LGA always converges either to the solution of the following problem

\[
  f(x) \rightarrow \min_{x \in Q}
\]

or to the point \( \bar{x} \in \partial Q \) such that

\[
  \frac{f(\bar{x})}{dn} < 0,
\]

where \( n \) is the unit normal vector to \( \partial Q \) that points away from \( Q \).

**Proof.**

If LGA terminates after a finite number of steps, then at its last step \( x_k \)

\[
  \frac{f(x) - f(x_k)}{|x - x_k|} \geq 0 \quad \text{for all} \quad x \in Q
\]

That means \( x_k \) delivers the global minimum for \( f(x) \) on \( Q \).

On the other hand, LGA generates the sequence (14) such that (15) is valid. Since \( Q \) is compact and \( f(x) \) is continuous, without loss of generality (taking a subsequence if needed), we state that there exist limits

\[
  \lim_{k \to \infty} x_k = x^*
\]

and

\[
  \lim_{k \to \infty} f(x_k) = f(x^*).
\]

If \( \bar{z} \) delivers the global minimum for \( f(x) \) on \( Q \) and \( \bar{z} \neq x^* \) then in accordance with LGA

\[
  \frac{f(x^*) - f(x_k)}{|x^* - x_k|} \leq \frac{f(\bar{z}) - f(x_k)}{|\bar{z} - x_k|} < 0.
\]

Taking the limit as \( x_k \to x^* \) yields the existence of the unit vector \( h \in \mathbb{R}^n \) such that

\[
  \nabla f(x^*) \cdot h < 0
\]

If \( x^* \) is an interior point of \( Q \), then \( \nabla f(x^*) \cdot h \geq 0 \) for all unit vectors \( h \) and the obtained contradiction proves the theorem. Otherwise, \( h \) is the unit normal vector to \( \partial Q \) that points away from \( Q \).

**Q.E.D.**

Theorem 2 gives us the natural condition under which LGA always delivers the global minimum.
**Corollary 1** Let $Q \subset \mathbb{R}^n$ be compact with a smooth boundary $\partial Q$. Let $f(x)$ be continuously differentiable and

$$\frac{f(\bar{x})}{d\bar{n}} \geq 0 \quad \text{for all } x \in \partial Q$$

where $\bar{n}$ is the outward unit normal vector to $\partial Q$. Then LGA always converges to the solution of the following problem

$$f(x) \rightarrow \min_{x \in Q}$$

In conclusion to this section we discuss the evolutionary leaps and their relationship with inflection points. Let $x \in \mathbb{R}^n$ be an inflection point for $f(x)$ if one can find $h \in \mathbb{R}^n$ such that $h \neq 0$ and

$$h \cdot \frac{\partial^2 f(x)}{\partial x^2} h = 0.$$

The next statement shows that an evolutionary leap is always a jump over an inflection point.

**Theorem 3** Let $f(x)$ be two times continuously differentiable. If $y \in \mathbb{R}^n$ is an evolutionary leap from $x$, then one can find $t \in (0, 1)$ such that

$$(y - x) \cdot \frac{\partial^2 f(z)}{\partial x^2} \bigg|_{z = x + t \cdot (y - x)} (y - x) = 0$$

**Proof.**

Multiplying (13) with $(y - x)$ yields

$$\nabla f(y) \cdot (y - x) = f(y) - f(x).$$

Consider the real function

$$\varphi(\tau) = \tau \cdot \nabla f(x + \tau \cdot (y - x)) \cdot (y - x) + f(x) - f(x + \tau \cdot (y - x)).$$

Since $\varphi(1) = \varphi(0) = 0$ it follows from Rolle’s theorem that there exists $t > 0$ such that

$$\frac{d}{d\tau} \varphi(\tau) = 0 \quad \text{for } \tau = t$$

and

$$\frac{d}{d\tau} \varphi(\tau) = t \cdot (y - x) \cdot \frac{\partial^2 f(z)}{\partial x^2} \bigg|_{z = x + t \cdot (y - x)} (y - x) = 0; \quad \text{for } \tau = t$$

completes the proof.

Q.E.D.
Note that
\[
    h \cdot \frac{\partial^2 f(x)}{\partial x^2} h \neq 0
\]
for all \( h \neq 0 \) and \( x \) implies the existence of not more than one critical point (if any) and this critical point is the global extremum. Moreover, the standard gradient method will deliver the global extremum.

The next section presents the numerical recursive leap gradient procedure that leads us to the global extremum of a real function \( f(x) \) on the segment \([a, b]\) as long as there exists a natural number \( n \) such that
\[
    \frac{d^n}{dx^n} f(x) \neq 0 \quad \text{on} \quad [a, b].
\]

### 4 Recursive leap gradient procedure

LGA replaces the optimization problem
\[
    f(x) \rightarrow \min_{x \in [a, b]}
\]
with
\[
    \frac{f(x) - f(x_k)}{x - x_k} \rightarrow \min_{x \in [x_k, b]}
\]
where \( x_k \) is calculated at the previous step of LGA. It leads us to the following recursive procedure executed at each iteration of LGA.

\[
    g_0(x) = f(x)
\]

and
\[
    g_{m+1}(x) = \frac{g_m(x) - g_m(x_k)}{x - x_k}.
\]

In order to complete an iteration of LGA for \( g_m(x) \) one needs to calculate
\[
    g_{m+1}(x) \rightarrow \min_{x \in [x_k, b]}
\]
and one recursively applies LGA again to find the respective minimum of \( g_{m+1}(x) \).

The next theorem shows when each iteration of LGA is completed after a finite number of such recursive steps.

**Theorem 4** If a real function \( f(x) \) is at least \( n \)-times continuously differentiable on the segment \([a, b]\) and
\[
    \frac{d^n}{dx^n} f(x) > 0 \quad \forall x \in [a, b]
\]
then each iteration of LGA for
\[
    f(x) \rightarrow \min_{x \in [a, b]}
\]
can be completed in a finite number of steps.
Proof.

The function can be represented by Taylor expansion centered at $x_k$,

$$f(x) = f(x_k) + \sum_{j=1}^{n-1} \frac{1}{j!} \frac{d^j}{dx^j} f(x_k) \cdot (x-x_k)^j + \int_0^1 \frac{d^n}{dx^n} f(x_k + t \cdot (x-x_k)) \cdot \frac{(x-x_k)^n (1-t)^n-1}{(n-1)!} dt$$

for all $x \in [x_k, b]$. In accordance with notations (16) and (17) we have

$$g_m(x) = \frac{1}{m!} \frac{d^m}{dx^m} f(x_k) + \sum_{j=m+1}^{n-1} \frac{1}{j!} \frac{d^j}{dx^j} f(x_k) \cdot (x-x_k)^j - m + \int_0^1 \frac{d^n}{dx^n} f(x_k + t \cdot (x-x_k)) \cdot \frac{(x-x_k)^n (1-t)^n-1}{(n-1)!} dt$$

and

$$g_{n-1}(x) = \frac{1}{(n-1)!} \frac{d^{n-1}}{dx^{n-1}} f(x_k) + \int_0^1 \frac{d^n}{dx^n} f(x_k + t \cdot (x-x_k)) \cdot \frac{(1-t)^n-1}{(n-1)!} dt \cdot (x-x_k).$$

Under the conditions of the theorem $g_{n-1}(x)$ achieves its minimum at $x_k$ and one can move forward with the next step of LGA for $g_{n-2}(x)$ that assures the convergence to the global minimum after a finite number of steps.

Q.E.D.

LGA recursive procedure yields the efficient numerical method of calculating the global extremum for a polynomial on a segment of the real line.

**Theorem 5** For any polynomial

$$p(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 + \cdots + p_n \cdot x^n$$

on a segment $[a, b]$ LGA recursive procedure delivers the global extremum for $p(x)$ in a finite number of steps.

Proof.

The proof is conducted by mathematical induction with respect to the degree of polynomial. As a basis of mathematical induction and for the purpose of illustrations let us consider finding the global minimum on $[a, b]$ for the quadratic polynomial

$$p(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 \quad (p_2 \neq 0)$$

In accordance with notations (16), (17)

$$g_0(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2$$

$$g_1(x) = p_1 + 2p_2 \cdot a + p_2 \cdot (x-a).$$

If $p_2 > 0$ then $g_1(x)$ has its minimum at $x_0 = a$. If

$$p_1 + 2p_2 \cdot a \geq 0$$

...
then according to LGA the global minimum is reached at $x = a$ and the procedure stops. Otherwise,

$$p_1 + 2p_2 \cdot a < 0$$

LGA leads us to $x_1 = a + h$, where the step size $h$ is dictated by the standard gradient descent algorithm applied at this point. We follow the standard gradient descent when calculating $x_1, x_2, x_3, \ldots x_k$ as long as

$$p_1 + 2p_2 \cdot x_k < 0.$$ 

The gradient descent either stops at $b$ and the global minimum is located at $b$ or, according to LGA, it stops at the first $x_k$ such that

$$p_1 + 2p_2 \cdot x_k \geq 0$$

and $x_k$ delivers the global minimum.

If $p_2 < 0$ then the minimum for $g_1(x)$ is located at $b$. According to LGA, if

$$p_1 + 2p_2 \cdot a + p_2 \cdot (b - a) \geq 0$$

then the minimum is at $a$. Otherwise, the minimum is at $b$. The basis of the mathematical induction is established.

The step of mathematical induction follows directly from LGA recursive procedure. Indeed, suppose that LGA recursive procedure delivers, in a finite number of steps, the global minimum for any polynomial of degree less than $n$. However, following notations (16) and (17), in order to calculate the global minimum for a polynomial $g_0(x)$ of $n$-th degree we need to calculate the global minimum for $g_1(x)$, a polynomial of $(n - 1)$-st degree. Hence, the statement follows by mathematical induction.

Q.E.D.

5 Horner’s LGA: numerical recursive procedure for polynomial extrema.

Horner’s algorithm (see, e.g., [4], [9]) plays the central role in numerical realization of the recursive LGA procedure for polynomials. LGA reduces the optimization problem

$$f(x) \rightarrow \min_{x \in [a, b]}$$

to

$$\frac{f(x) - f(x_k)}{x - x_k} \rightarrow \min_{x \in [x_k, b]}$$

where $x_k$ is calculated at the previous step of LGA. If

$$f(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 + \cdots + p_n \cdot x^n$$
is a polynomial with real coefficients then so is
\[
\frac{f(x) - f(x_k)}{x - x_k} = q_0 + q_1 \cdot x + q_2 \cdot x^2 + \cdots + q_{n-1} \cdot x^{n-1}
\]
where coefficients \(q_0, q_1, \ldots q_{n-1}\) are calculated as follows.

**Horner’s Algorithm**

- \(q_{n-1} = p_n\)
- \(q_{j-1} = x_k \cdot q_j + p_j\), where \(j = n - 1, n - 2, \ldots, 1\)

Recursive LGA \((16), (17)\) described in section 4 is reduced to a finite number of iterations for Horner’s Algorithm until the resulted polynomial is a linear function. For a linear function on a segment the solution of the optimization problem is trivial and therefore the recursive procedure delivers the global extremum.

Horner’s LGA relies on the following two sub-routines.

**Horner\_Algorithm**\((\{p_0, p_1, \ldots, p_n\}, z)\)\{
\[
q_{n-1} = p_n
\]
for \(i = n - 1, \ldots, 0\) do
\[
q_{i-1} = z \cdot q_i + p_i
\]
done
return \(\{q_0, q_1, \ldots, q_{n-1}\}\)
\}

and

**Horner\_Evaluate**\((\{p_0, p_1, \ldots, p_n\}, z)\)\{
\[
result = p_n
\]
for \(i = n - 1, \ldots, 0\) do
\[
result = z \cdot result + p_i
\]
done
return result
\}
Horner's LGA calculates the global minimum for a polynomial on a segment \([a, b]\) as follows.

\[
\text{Horner LGA}(\{p_0, p_1, \ldots, p_n\}, [a, b], \text{step}_size) \{
\]

\[
\text{if } (n == 1 && p_1 \geq 0) \text{ return } a
\]

\[
\text{if } (n == 1 && p_1 < 0) \text{ return } b
\]

\[
\{q_0, q_1, \ldots, q_{n-1}\} = \text{Horner Algorithm}(\{p_0, p_1, \ldots, p_n\}, a)
\]

\[
s = \text{Horner LGA}(\{q_0, q_1, \ldots, q_{n-1}\}, [a, b], \text{step}_size)
\]

\[
s_{\text{prev}} = a
\]

Let \(q = \{q_0, q_1, \ldots, q_{n-1}\}\)

while (\text{Horner Evaluate}(q, s) < 0 && s < b)

\[
do
\]

\[
\text{if } (s == s_{\text{prev}})
\]

\[
do
\]

\[
s_{\text{prev}} = s
\]

\[
q = \text{Horner Algorithm}(\{p_0, p_1, \ldots, p_n\}, s_{\text{prev}} + \text{step}_size)
\]

\[
s = \text{Horner LGA}(q, [s_{\text{prev}} + \text{step}_size, b], \text{step}_size)
\]

\[
done
\]

\[
else
\]

\[
do
\]

\[
s_{\text{prev}} = s
\]

\[
q = \text{Horner Algorithm}(\{p_0, p_1, \ldots, p_n\}, s_{\text{prev}})
\]

\[
s = \text{Horner LGA}(q, [s_{\text{prev}}, b], \text{step}_size)
\]

\[
done
\]

\[
done
\]

\[
return s
\]

\}

Horner’s LGA also is able to deliver global extrema for polynomials of several variables. For this purpose one can follow the scenario outlined in [2]. We use the polynomials of two variables in order to illustrate this approach. Increasing the number of variables makes the calculations more difficult but does not add significant changes to the ideology of the approach.

Consider a polynomial of two variables

\[ p(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{ij}x^i \cdot y^j \]  

(18)

on a convex compact set \( Q \). Let \( (\bar{x}, \bar{y}) \in Q \) be an arbitrary point from \( Q \). In order to find the global minimum of \( p(x, y) \) on \( Q \) introduce the parametric family of polynomials defined as follows.

\[ f_\phi(t) = p(\bar{x} + t \cdot \cos(\phi), \bar{y} + t \cdot \sin(\phi)) \]

where \( t \in [0, r(\phi)] \) and

\[ Q = \{ (\bar{x} + t \cdot \cos(\phi), \bar{y} + t \cdot \sin(\phi)); \ 0 \leq \phi \leq 2 \cdot \pi \ \text{and} \ t \in [0, r(\phi)] \} \]

For each \( \phi \in [0, 2 \cdot \pi] \) we can apply LGA to the polynomial of one variable \( f_\phi(t) \) and obtain the function

\[ M(\phi) = \min_{t \in [0, r(\phi)]} (f_\phi(t)) \]

Then we choose \( \bar{\phi} \) that delivers the smallest value for \( M(\phi) \). Finally, the polynomial \( p(x, y) \) achieves its minimum on \( Q \) at

\[ x = \bar{x} + \bar{t} \cdot \cos(\bar{\phi}) \]
\[ y = \bar{y} + \bar{t} \cdot \sin(\bar{\phi}) \]

with \( \bar{t} \) being the point from \([0, r(\bar{\phi})]\) where \( f_\bar{\phi}(t) \) reaches its minimum.

In applications instead of calculating \( M(\phi) \) for each \( \phi \in [0, 2 \cdot \pi] \) one selects a finite set of points \( \{\phi_j\}_{j=1}^{N} \) from \([0, 2 \cdot \pi]\) and then takes \( \bar{\phi} \) as \( \phi_k \) with the minimal value \( M(\phi_k) \) among \( \{M(\phi_j)\}_{j=1}^{N} \). The size of the set \( \{\phi_j\}_{j=1}^{N} \) is dictated by the shape of \( Q \) and the acceptable margin of error for the solution of the minimization problem. In general, it could be a costly operation from the numerical point of view. That leads us to the next simplified numerical approach which is much more cost efficient in terms of numerical operations. However, it might miss the global minimum and deliver only a suboptimal solution.

Consider

\[ p(x, y) \to \min_{(x, y) \in Q} \]

where \( p(x, y) \) is defined in (18) and

\[ Q = \{ (x, y); \ a_1 \leq x \leq b_1, \ a_2 \leq y \leq b_2 \} \]

Let \( (x_0, y_0) \) be an arbitrary point from \( Q \). Then we take \( y_1 = y_0 \) and \( x_1 \) is the solution of the minimization problem

\[ p(x, y_0) \to \min_{x \in [a_1, b_1]} \]
Next, \( x_2 = x_1 \) and \( y_2 \) is the solution for
\[
p(x_1, y) \rightarrow \min_{y \in [a_2, b_2]}.
\]
That yields the sequence of points
\[
(x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k), \ldots
\]
For even \( k = 2 \cdot m \) (\( m = 1, 2, \ldots \)) we take \( x_k = x_{k-1} \) and \( y_k \) is the solution of the minimization problem
\[
p(x_{k-1}, y) \rightarrow \min_{y \in [a_2, b_2]}.
\]
For odd \( k = 2 \cdot m + 1 \) (\( m = 0, 1, 2, \ldots \)) we have \( y_k = y_{k-1} \) and \( x_k \) is the solution of the problem
\[
p(x, y_{k-1}) \rightarrow \min_{x \in [a_1, b_1]}.
\]
The algorithm terminates when \((x_{k+1}, y_{k+1}) = (x_k, y_k)\) or when the value
\[
(x_{k+2} - x_k)^2 + (y_{k+2} - y_k)^2
\]
is smaller than a predefined number. This procedure admits a straightforward generalization to polynomials with any number of variables. It can be implemented as a recursive function with respect to the number of variables. It is more efficient than the algorithm based on the topping functions. However, it might converge to a suboptimal solution instead of the global extremum.

For polynomials with several variables one can propose an even more speedy numerical minimization algorithm which however suffers the same benign drawback of possible convergence to a suboptimal minimum. Consider
\[
p(x_1, x_2, \ldots, x_n) = \sum_{i_1 + i_2 + \ldots + i_n = 0}^m p_{i_1 i_2 \ldots i_n} \cdot x_1^{i_1} \cdot x_2^{i_2} \cdot \ldots \cdot x_n^{i_n}
\]
where \( i_1 i_2 \ldots i_n \) are natural numbers and \( \{p_{i_1 i_2 \ldots i_n}\} \) are real coefficients. The optimization problem is the following.
\[
p(x) \rightarrow \min_{x \in Q}
\]
where \( x = (x_1, x_2, \ldots, x_n) \) and
\[
Q = \{(x_1, x_2, \ldots, x_n); \ a_i \leq x_i \leq b_i \text{ for } i = 1, 2, \ldots, n\}.
\]
The recursive LGA procedure is constructed with respect to the degree of the polynomial. For a polynomial of the first degree
\[
p(x) = p_0 + \sum_{i=1}^n p_i \cdot x_i
\]
the solution for the optimization problem is

\[ x = (s_1, s_2, \ldots, s_n) \]

where

\[ s_i = \begin{cases} 
  b_i & \text{for } p_i < 0 \\
  a_i & \text{for } p_i \geq 0 
\end{cases} \]

The recursion is based on the following version of LGA.

Let \( x(k) \) denote the \( k \)-th iteration of our LGA procedure with

\[ x(0) = (a_1, a_2, \ldots, a_n). \]

At each step we consider the optimization problem

\[
\frac{p(x) - p(L_{ix}(k)(x))}{Pr_i(x - x(k))} \rightarrow \min_{x \in Q(k)} \tag{19}
\]

where

\[ L_{ix}(k)(x) = (x_1, \ldots, x_{i-1}, x_i(k), x_{i+1}, \ldots, x_n), \]

\( x_j(k) \) denotes the \( j \)-th entry of \( x(k) \),

\[ Q(k) = \{(x_1, x_2, \ldots, x_n); x_i(k) \leq x_i \leq b_i \text{ for } i = 1, 2, \ldots, n\}. \tag{20} \]

and

\[ Pr_i(x) = x_i \]

is the projection onto the \( i \)-th axes. Then the LGA iterations are defined by the following rules.

**Leap Gradient Algorithm (LGA).**

- Select the variable \( x_i \) such that \( p(x) \) has at least the second power of \( x_i \). Then find the solution \( x^*(k) \) for the optimization problem \( 19, 20 \).

- If \( x^*(k) = x(k) \) and the partial derivative

\[
\frac{\partial}{\partial x_i} p(x(k)) \geq 0, \tag{21}
\]

then the iterative procedure stops at \( x(k) \) and one considers the minimization problem

\[
p(L_{ix}(k)(x)) \rightarrow \min_{L_{ix}(k)(x) \in Q} \tag{22}
\]

Note that \( i \)-th coordinate of \( L_{ix}(k)(x) \) is fixed at \( x_i(k) \) and \( 22 \) has the dimension reduced by one. Let \( \bar{x} \) be the solution for \( 22 \). Then \( L_{ix}(k)(\bar{x}) \) delivers the global minimum for \( p(x) \) on \( Q \). Indeed, it follows from \( x^*(k) = x(k) \) and \( 21 \) that for all \( x \in Q \)

\[
p(x) \geq p(L_{ix}(k)(x)). \tag{23}
\]
On the other hand, since $\bar{x}$ is the solution for (22) we have
\[ p(L_{ix}(k) (\bar{x})) \geq p(L_{ix}(k) (x)) \quad \forall x \in Q \] (24)

Combining (23) and (24) we conclude that $p(x)$ reaches the global minimum on $Q$ at $L_{ix}(k) (\bar{x})$.

- If $x^*(k) \neq x(k)$ and either
  \[ \frac{p(x^*(k)) - p(L_{ix}(k) (x^*(k)))}{Pr_i(x^*(k) - x(k))} \geq 0, \]
  or
  \[ x^*(k) = (b_1, b_2, \ldots, b_n) \]
  then the iterative procedure stops at $x^*(k)$ and $p(x)$ reaches its global minimum on $Q$ at $L_{ix}(k) (\bar{x})$, where $\bar{x}$ is the solution of the optimization problem (22).

- If $x^*(k) = x(k)$ and the partial derivative
  \[ \frac{\partial}{\partial x_i} p(x(k)) < 0, \]
  then we make a small step in the direction of $x_i$ axis,
  \[ x(k+1) = (x_1(k), x_2(k), \ldots, x_{i-1}(k), x_i(k) + h, x_{i+1}(k), \ldots, x_n(k)) \]
  where $h$ is the fixed step-size.

- If $x^*(k) \neq x(k)$ and
  \[ \frac{p(x^*(k)) - p(L_{ix}(k) (x^*(k)))}{Pr_i(x^*(k) - x(k))} < 0, \]
  then we perform the evolutionary leap and take $x(k+1) = x^*(k)$.

LGA suggests the following recursive procedure.
\[ g_0(x) = p(x) \]
and
\[ g_{m+1}(x) = g_m(x) - g_m(L_{ix}(k) (x)) \]
\[ Pr_i(x - x(k)) \]
where $i$ is chosen so that $g_m(x)$ has at least the second power of $x_i$. By LGA the optimization problem for $g_m(x)$ is reduced to the optimization problem for $g_{m+1}(x)$. After a finite number of recursive steps $g_{m+1}(x)$ becomes a polynomial of one variable or a linear function of $x$ that admits a simple solution of the optimization problem and therefore our recursive procedure delivers the extremum for the original function. Formal program that realizes this scenario is similar to the one presented for polynomials.
of one variable. A polynomial of several variables is defined by a multidimensional array

\[ P[m_1, m_2, \ldots, m_n] = \{ p(i_1, i_2, \ldots, i_n); 0 \leq i_j \leq m_j, j = 1, 2, \ldots, n \} \]

corresponding to the polynomial

\[ f(x_1, x_2, \ldots, x_n) = \sum_{i_1=0}^{m_1} \sum_{i_2=0}^{m_2} \ldots \sum_{i_n=0}^{m_n} p(i_1, i_2, \ldots, i_n) \cdot x_1^{i_1} \cdot x_2^{i_2} \cdots x_n^{i_n}. \]

The next procedure calculates the coefficients of the polynomial

\[ \frac{f(x) - f(L_\ell(x))}{P \delta_{k}(x-z)}. \]

**Horner Algorithm**

\[ \text{Horner Algorithm}(P[m_1, m_2, \ldots, m_n], k, z) \}\n
\[ S_k = \{(i_1, i_2, \ldots, i_{k-1}, i_k+1, \ldots, i_n); 0 \leq i_j \leq m_j, j = 1, 2, \ldots, n \} \]

*For all multi-indices \( \vec{i} \) from*

*invoke Horner Algorithm for polynomials of one variable,

\[ q_i = \text{Horner Algorithm}(\{p_{i_1, i_2, \ldots, i_{k-1}, 0, i_k+1, \ldots, i_n, \ldots, p_{i_1, i_2, \ldots, i_{k-1}, i_k+1, \ldots, i_n}\}, z_k). \]

*It yields the array

\[ Q[m_1, m_2, \ldots, m_{k-1}, m_k - 1, m_{k+1}, \ldots, m_n] = \{ q_i \}_{\vec{i} \in S_k}. \]

*Return \( Q[m_1, m_2, \ldots, m_{k-1}, m_k - 1, m_{k+1}, \ldots, m_n]. \)

\}\n
In order to evaluate the polynomial \( f(x_1, x_2, \ldots, x_n) \) at \( z \in \mathbb{R}^n \) consider the procedure.
\texttt{Horner\_PreEvaluate}(P[m_1, m_2, \ldots, m_n], k, z)\{ \\
\text{For all multi-indices } \vec{i} \text{ from } \\
S_k = \{(i_1, i_2, \ldots, i_{k-1}, i_k + 1, \ldots, i_n); 0 \leq i_j \leq m_j, j = 1, 2, \ldots n\} \\
\text{invoke } \texttt{Horner\_Evaluate} \text{ for polynomials of one variable,} \\
q_i = \texttt{Horner\_Evaluate}\{(p_{i_1, i_2, \ldots, i_{k-1}, 0, i_k+1, \ldots, i_n}; \ldots; p_{i_1, i_2, \ldots, i_{k-1}, m_k, i_k+1, \ldots, i_n}, \vec{z}_k\}. \\
\text{It yields the array} \\
Q[m_1, m_2, \ldots, m_{k-1}, m_{k+1}, \ldots, m_n] = \{q_i\}_{i \in S_k}. \\
\text{Return } Q[m_1, m_2, \ldots, m_{k-1}, m_{k+1}, \ldots, m_n]. \\
\}\}

Now one can numerically evaluate the polynomial \(f(x_1, x_2, \ldots, x_n)\) at \(z \in \mathbb{R}^n\) with the help of the following subroutine.

\texttt{Horner\_Evaluate}(P[m_1, m_2, \ldots, m_n], z)\{ \\
Q = P[m_1, m_2, \ldots, m_n] \\
\text{for } k = 1, 2, \ldots, n \\
\text{do} \\
Q = \texttt{Horner\_PreEvaluate}(Q, k, z)\{ \\
\text{done} \\
\text{After } n \text{ iterations } Q \text{ becomes a number.} \\
\text{Return } Q. \\
\}\}

Now we are able to describe formally Horner’s LGA for polynomials of several variables.

\texttt{Horner\_LGA}(P[m_1, m_2, \ldots, m_n], [a_1, b_1], \ldots, [a_n, b_n], \text{step\_size})\{ \\
\text{Let } q[n] \text{ be an array} \\
q[n] = \{q(1), \ldots, q(n)\}. \\
\text{Let } \vec{k} \text{ denote the multi-index with all entries but one equal to zero and k-th non-zero entry is equal to 1.} \\
\text{Let Counter = 0.} \\
\text{for } k = 1, 2, \ldots, n
do
  if ($m_k == 1$)
    do
      $\text{Counter} = \text{Counter} + 1.$
      if ($p(k) < 0$)
        do
          $q(k) = b_k$
        done
      else
        do
          $q(k) = a_k$
        done
    done
  if ($m_k == 0$)
    do
      $q(k) = a_k$
    done
  if ($m_k > 1$)
    do
      go to "START"
    done
  done
if ($\text{Counter} == 1$)
  do
    return $q$.
  done

START:
Choose the integer $k$ such that $m_k \geq 1$. Let $a = \{a_1, \ldots, a_n\}$. 

23
Let $Q = [m_1, m_2, \ldots, m_k, m_k-1, m_{k+1}, \ldots, m_n]$.

$Q = \text{Horner Algorithm}(P[m_1, m_2, \ldots, m_n], k, a)$

$s = \text{Horner LGA}(Q, [a_1, b_1], \ldots, [a_n, b_n], \text{step size})$

$Q = \text{Horner PreEvaluate}(P, k, s)$

$q = \text{Horner LGA}(Q, [a_1, b_1], \ldots, [a_n, b_n], \text{step size})$

for $i = 1, 2, \ldots, k-1$

    do

    $s_i = q_i$

    done

for $i = k, 2, \ldots, n-1$

    do

    $s_{i+1} = q_i$

    done

prev = $a$

while ($\text{Horner Evaluate}(Q, s) < 0 \&\& s(k) < b$)

    do

    if ($s == \text{prev}$)

        do

            prev = $s$

            $Q = \text{Horner Algorithm}(P[m_1, m_2, \ldots, m_n], k, \{\text{prev}_1, \ldots, \text{prev}_k + \text{step size}, \ldots, \text{prev}_n\})$

            $s = \text{Horner LGA}(Q, [\text{prev}_1, \text{b}_1], \ldots, [\text{prev}_k + \text{step size}, \text{b}_k], \ldots, [\text{prev}_n, \text{b}_n], \text{step size})$

            $Q = \text{Horner PreEvaluate}(P, k, s)$

            $q = \text{Horner LGA}(Q, [\text{prev}_1, \text{b}_1], \ldots, [\text{prev}_k + \text{step size}, \text{b}_k], \ldots, [\text{prev}_n, \text{b}_n], \text{step size})$

        for $i = 1, 2, \ldots, k-1$

            do

                $s_i = q_i$

            done

        for $i = k, 2, \ldots, n-1$

    done
\begin{verbatim}
do
  s_{i+1} = q_i
done
done
else
do
  prev = s
  Q = Horner\textunderscore Algorithm(P[m_1, m_2, \ldots, m_n], k, prev)
  s = Horner\textunderscore LGA(Q, [prev_1, b_1], \ldots, [prev_n, b_n], step\_size)
  Q = Horner\textunderscore PreEvaluate(P, k, s)
  q = Horner\textunderscore LGA(Q, [prev_1, b_1], \ldots, [prev_n, b_n], step\_size)
  for i = 1, 2, \ldots, k - 1
do
    s_i = q_i
done
  for i = k, 2, \ldots, n - 1
do
    s_{i+1} = q_i
done
done
return s
\end{verbatim}

Due to constraints imposed by the size of a journal publication we omit various tuning details related to the outlined LGA and their optimizations for various practical problems. For example, when applying LGA to polynomials with many variables the initial guess for the optimal solution should not coincide with a critical point. Moreover, in LGA applications it is not always warranted to perform evolutionary leap search at each iteration of the algorithm as it might appear for the reader of this paper. In practice, one can use standard gradient descent until it either converges or becomes
unacceptably slow and only then perform the recursive evolutionary leap search furnished by LGA. One can also use the information about inflection points in order to tune the search procedure for evolutionary leaps. LGA also can be employed for the construction of topping functions discussed in [2]. A topping function could be constructed recursively with the help of LGA.

6 Conclusion

LGA delivers an efficient recursive algorithm of calculating extrema for polynomials (java implementation of the algorithms discussed in the paper available upon request). The recursive LGA algorithm also is able to calculate extrema for multivariable polynomials on a compact set. Since any continuous function on a compact can be approximated by a polynomial LGA can be employed to deliver global extrema for any continuous function.

References

[1] L. Armijo, Minimization of functions having Lipschitz continuous first partial derivatives, Pacific J. Math., 16, 1 - 3 (1966)

[2] G.E. Backus, The topping algorithm for calculating extrema of polynomials on subsets of \( \mathbb{R}^n \), SIAM J. Numer. Anal., 17, 479 - 511 (1980)

[3] Y.H. Dai, J.Y. Yuan, and Y. Yuan, Modified two-point step-size gradient methods for unconstrained optimization, Computational Optimization and Applications, 22, 103-109 (2002).

[4] N.J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM, Philadelphia (2002).

[5] A. A. Goldstein, Constructive real analysis, Harper & Row Publishers, New York (1967)

[6] O. G"uler, Foundations of Optimizations, Springer, New York (2010)

[7] Z. Guangxing, X. Shuijing, Global minimization of multivariate polynomials using nonstandard methods, J Glob Optim., 53, 391-415 (2012)

[8] L. Kantorovich and G. Akilov, Functional Analysis in Normed Spaces, Fizmatgiz, Moscow (1959), translated by D. Brown and A. Robertson, Pergamon Press, Oxford (1964)

[9] D.E. Knuth, Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed., MA: Addison-Wesley, (1998)

[10] B. Mishra, Algorithmic Algebra, Texts and Monographs in Computer Science. Springer, New York (1993)
[11] Yu. Nesterov, Introductory Lectures on Convex Optimization, Applied Optimization, 87, Kluwer Academic Publishers, Boston (2004)

[12] E. Polak, Optimization. Algorithms and Consistent Approximations, Applied Mathematical Sciences, 124, Springer, New York (1997)

[13] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal Royal Statistical Society B, 58, 267-288, (1996)

[14] J.V. Uspensky, Theory of equations, McGraw-Hill, New York (1948).

[15] Ph. Wolfe, Convergence conditions for ascent methods, SIAM Review, 11, 226-235 (1969)

[16] Ph. Wolfe, Convergence conditions for ascent methods II: Some corrections, SIAM Review, 13, 185-188 (1971)

[17] B.L. van der Waerden, Algebra I, Springer Verlag, New York (1971)

[18] X. Zhou, D.Y. Gao, C. Yang, Canonical primaldual algorithm for solving fourth-order polynomial minimization problems, Applied Mathematics and Computation, 227, 246-255 (2014)

[19] Zhikun She, Zhiming Zheng, Condition number based complexity estimate for computing local extrema, Journal of Computational and Applied Mathematics 230, 233242 (2009)