Leap Gradient Algorithm. The Fastest Method for Univariate Polynomial Extrema.

Working Paper · April 2014

DOI: 10.13140/RG.2.1.1545.2244

1 author:

S. Nikitin
Arizona State University
63 PUBLICATIONS **188** CITATIONS

SEE PROFILE

Leap Gradient Algorithm The Fastest Method for Univariate Polynomial Extrema *

Sergey Nikitin

Received: 06-2014 / under revision (draft in work)

Abstract

Leap gradient algorithm is able to solve global univariate optimization problems. It requires neither convexity nor the knowledge of the Lipschitz constant for the target function. For a broad variety of target functions after performing (if necessary) several evolutionary leaps the algorithm naturally becomes the standard gradient descent (or ascent) procedure near the global extremum. Moreover, it leads us to an efficient recursive numerical procedure for calculating the global extrema of univariate polynomials. Numerical experiments presented in the paper show that the newly proposed leap gradient algorithm outperforms several well known methods of finding global extrema for univariate polynomials.

1 Introduction

The problem of finding global extrema (maxima and minima) for a univariate real function is important for variety of real world applications (see introduction in [13]). In many industrial applications the global optimization algorithm is expected to operate in real time while simultaneously, finding the global extremum of non-convex functions exhibiting large number of local sub-extrema.

The problem of efficiently finding a function's global extremum has been historically challenging. One of the first solutions, Zero Derivative Method (ZDM), was proposed by Pierre de Fermat (1601-1665). His main idea was to look for the global extremum among critical points: the points where the derivative of the target function is zero. Despite its theoretical significance, Fermat's proposed method (ZDM) is limited by the numerical difficulties imposed by finding critical points.

One of the leading global optimization approaches adopted by many industrial applications is a brute-force search or exhaustive search for the global extremum (Brute-Force Search (BFS)). It is simple to implement but its performance linearly depends on the complexity of the target function and the size of the search area.

*partially supported by ASU, CLAS Undergraduate Summer Enrichment (USE) award

A plethora of optimization methods have been developed for various types of target functions. Among them Piyavskii-Shubert Method (PSM) occupies a special place [10], [11], [12]. It is one of the few procedures that delivers the global extremum for a univariate function and at the same time it exhibits reasonable performance as long as the respective Lipschitz constant is of a modest value. On the other hand, the method is very sensitive to the size of the Lipschitz constant: its performance sharply diminishes for large Lipschitz constants. For this reason, accelerations and improvements of PSM were developed in the following papers [3], [8], [13], [14] . Numerical experiments presented in this paper show that Leap Gradient Algorithm (referred as LGA) significantly outperforms PSM together with its modifications and improvements (from [3], [13], [14]) when finding global extrema of polynomials.

The method of gradient descent (see, e.g. [5], [6], [9]) is widely used to solve various practical optimization problems. The main advantage of the gradient descent algorithm is its simplicity and applicability to a wide range of practical problems. On the other hand, gradient descent has 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 for a univariate function 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. LGA neither requires the knowledge of the Lipschitz constant nor convexity conditions that are often imposed on the target function. Moreover, LGA naturally becomes 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 efficient algorithm for calculating global extrema for univariate polynomials. LGA does not intend to locate any critical points (like ZDM) instead it follows gradient descent (ascent) until a local extremum is reached and then performs an evolutionary leap towards the next extremum. As far as performance is concerned, numerical experiments conducted for univariate polynomials show that LGA outperforms BFS, ZDM and PSM with all its modifications from [3], [13], [14].

The layout of this publication is as follows.

Section 1 is the introduction.

Section 2 introduces LGA for univariate functions.

Section 3 explores the recursive application of LGA.

Section 4 describes a recursive implementation of LGA for polynomials.

Section 5 reports on numerical experiments with LGA. It compares LGA with ZDM, BFS, PSM and accelerations of PSM presented in [3], [13], [14]. Polynomial roots for ZDM are calculated with the help of Laguerre's Method [1], [2].

Section 7 finalizes the publication with a snapshot of the working and thoroughly tested source code for LGA.

2 LGA: leaps towards the global extremum

Let $f(x)$ be known and well defined real function on $[a,\ b]$, a closed interval of real numbers. Consider the optimization problem

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

LGA can solve it with a given precision $h > 0$ if its solution exists.

Leap Gradient Algorithm (LGA).

STEP 0. Set

$$x_0 = a.$$

STEP 1. Iterate

$$x_{k+1} = x_k + h,$$

as long as

$$f(x_k + h) \leq f(x_k)$$

and

$$x_k < b.$$

If $x_k \geq b$ then **STOP** and take $(b, f(b))$ as an estimate of the argument and the value for the global minimum.

If

$$f(x_k + h) > f(x_k) \tag{1}$$

then proceed with **STEP 2**.

STEP 2. Let $x_k^\star$ be the solution of the following optimization problem

$$\frac{f(x) - f(x_k)}{x - x_k} \quad \rightarrow \quad \min_{x\in[x_k,\,b]} \tag{2}$$

If

$$x_k^\star \geq x_k \ \text{ and } \ f(x_k^\star) \geq f(x_k)$$

then **STOP** and $(x_k, f(x_k))$ is an estimate of the argument and the value for the global minimum.

If

$$x_k^\star > x_k \ \text{ and } \ f(x_k^\star) < f(x_k)$$

then $x_{k+1} = x_k^\star$ (LGA performs *an evolutionary leap*) and go to **STEP 1**.

LGA is illustrated in Fig. 1.

If the target function is twice continuously differentiable on $(a,\ b)$ then the number of inflection points from $(a,\ b)$ is related to the number of evolutionary leaps performed by LGA (see **STEP 2**).

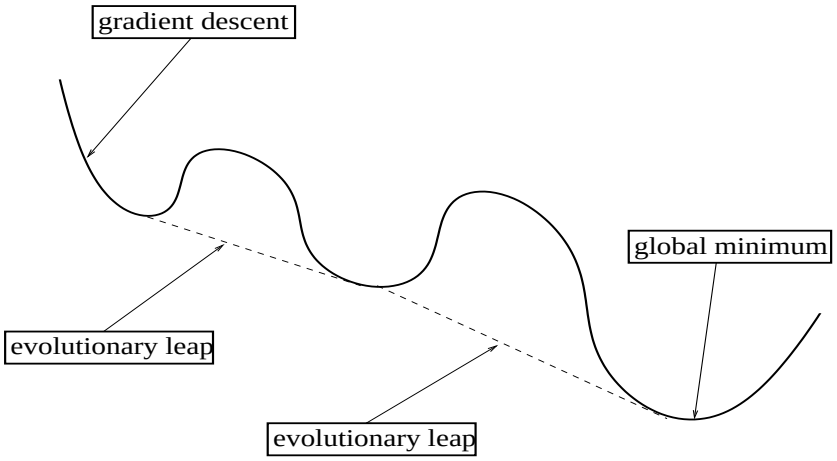

Figure 1: Leap Gradient Algorithm (LGA) for a univariate function

Theorem 1 *Let $f(x)$ be twice continuously differentiable on $(a,\ b)$. Let $x_k^\star$ be the value of the evolutionary leap (***STEP 2** *of LGA).*

If

$$a < x_k < x_k^\star < b$$

then $(x_{k-1},\ x_k^\star]$ contains at least two points where $(\frac{d}{dx})^2 f(x)$ is equal to zero.

Proof.

Evolutionary leaps from x_k (on $(a,\ b)$) are solutions of the following equation.

$$f(x) - f(x_k) = \frac{d}{dx} f(x) \cdot (x - x_k) \tag{3}$$

where $x \in\ [x_k,\ b)$.

Indeed, if $f(x)$ is continuously differentiable then the necessary condition for x to be the solution for (2) on $(x_k,\ b)$ is

$$\frac{d}{dx}\left(\frac{f(x) - f(x_k)}{x - x_k}\right) = 0.$$

Differentiating yields

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

After multiplying the equation with $(x - x_k)^2$ we obtain (3).

Since $x_k^\star$ is the solution of the problem (2) and $f(x)$ is twice differentiable we conclude that

$$\left(\frac{d}{dx}\right)^2 \left(\frac{f(x) - f(x_k)}{x - x_k}\right) \Big|_{x = x_k^\star} \geq 0$$

Differentiating yields

$$\frac{\left(\frac{d}{dx}\right)^2 f(x)}{(x-x_k)} - \frac{2}{(x-x_k)^3}\cdot\left(\left(\frac{d}{dx}f(x)\right)\cdot(x-x_k) - (f(x)-f(x_k))\right) \geq 0$$

when $x = x_k^\star$. Hence, making use of (3) we obtain

$$\left(\frac{d}{dx}\right)^2 f(x)\Big|_{x=x_k^\star} \geq 0. \tag{4}$$

According to **STEP 2** of LGA for the evolutionary leap $x_k^\star$ we have

$$f(x_k^\star) < f(x_k).$$

That together with (3) implies

$$\left(\frac{d}{dx}\right) f(x)\Big|_{x=x_k^\star} < 0. \tag{5}$$

On the other hand, (1) yields the existence of ξ such that

$$\left(\frac{d}{dx}\right) f(x)\Big|_{x=\xi} > 0 \quad \text{and } \xi \in (x_k,\, x_k+h)$$

That together with (5) yield the existence of $\tilde{x} \in (x_k, x_k^\star)$ where

$$\left(\frac{d}{dx}\right)^2 f(x)\Big|_{x=\tilde{x}} < 0. \tag{6}$$

Taking into account that $x_k > a$ is obtained after **STEP 1** of LGA and

$$f(x_k) - f(x_{k-1}) \leq 0, \qquad f(x_k+h) - f(x_k) > 0$$

we obtain the existence of $\bar{\bar{x}} \in (x_{k-1},\, x_k+h)$ where

$$\left(\frac{d}{dx}\right)^2 f(x)\Big|_{x=\bar{\bar{x}}} > 0. \tag{7}$$

Under the conditions of the theorem

$$\left(\frac{d}{dx}\right)^2 f(x)$$

is continuous on $(a,\, b)$ and the statement follows from (4), (6) and (7).

Q.E.D.

That means (in a generic situation) a non trivial evolutionary leap inside $(x_k,\, b)$ is only possible over two inflection points of a target function.

Example 1 *Consider the optimization problem*

$$f(x) = x^4 + a \cdot x^3 + b \cdot x^2 + c \cdot x + d \quad \rightarrow \quad \min_{x \in \mathbb{R}} \tag{8}$$

where a, b, c, and d are real numbers. If

$$3a^2 > 8b$$

then

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

has two different zeroes. Hence, by Theorem 1, LGA might need to perform a single evolutionary leap (over two inflection points) in order to solve the optimization problem.

Otherwise, $3a^2 \leq 8b$, LGA coincides with the standard gradient decent.

3 Recursive leap gradient procedure

LGA replaces the optimization problem

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

with

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

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.

$$g_0(x) = f(x) \tag{9}$$

and

$$g_m(x) = \frac{g_{m-1}(x) - g_{m-1}(x_k)}{x - x_k}. \tag{10}$$

In order to complete an iteration of LGA for $g_{m-1}(x)$ one needs to calculate

$$g_m(x) \quad \rightarrow \quad \min_{x \in [x_k .\, b]}$$

and LGA finds the minimum of $g_m(x)$. If finding the minimum for $g_m(x)$ is obvious, then the iteration of LGA is completed after m recursive steps.

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

Theorem 2 *If a real function $f(x)$ is at least n-times continuously differentiable on the segment $[a,\, b]$ and*

$$\frac{d^n}{dx^n} f(x) \geq 0 \quad \forall \; x \in \; [a,\, b]$$

then each iteration of LGA for

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

is completed after not more than $n-1$ *recursive steps.*

Proof.

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

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

for all $x\in[x_k,\,b]$. In accordance with notations (9) and (10) we have

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

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 on $[x_k,\,b]$ at x_k.

Q.E.D.

Recursive LGA is an efficient numerical method for finding global extrema of univariate polynomials.

Theorem 3 *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 $[a,\,b]$ *recursive LGA 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 a 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 (9), (10)

$$\begin{aligned} g_0(x) &= p_0 + p_1 \cdot x + p_2 \cdot x^2 \\ g_1(x) &= p_1 + 2p_2 \cdot a + p_2 \cdot (x-a). \end{aligned}$$

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 ad $x = a$ 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 required precision of LGA. 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 recursive LGA. Indeed, suppose that recursive LGA delivers, in a finite number of steps, the global minimum for any polynomial of degree less than n. However, following notations (9) and (10), 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.

4 LGA: numerical recursive procedure for polynomial extrema.

Horner's algorithm (see, e.g., [4], [7]) plays the central role in implementation of recursive LGA 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

- $q_{n-1} = p_n$
- $q_{j-1} = x_k \cdot q_j + p_j, \quad \text{where} \quad j = n-1,\ n-2,\ \ldots,\ 1$

Recursive LGA (9), (10) described in section 3 is reduced to a finite number of iterations for Horner's Algorithm until the resulted polynomial is either a linear or a quadratic function. Then the solution of the optimization problem is trivial and therefore the recursive procedure delivers the global extremum.

Performing an evolutionary leap is a numerically expensive operation. Theorem 1 shows that each LGA leap inside the interval is a jump over two zeroes of the second derivative of the target function. That allows to improve the performance of LGA for polynomials by limitting the number of evolutionary jumps by at most $n-2$, where n is the degree of the target polynomial. The respective modification of LGA for

$$p_0 + p_1 \cdot x + p_2 \cdot x^2 + \cdots + p_n \cdot x^n \;\rightarrow \min_{x\in[a,\,b]}$$

is as follows.

LGA for polynomials

STEP 0. If the degree of the polynomial is 1, then return

$$(a,\ p_0 + p_1 \cdot a) \quad \text{if} \quad p_0 + p_1 \cdot a \leq\ p_0 + p_1 \cdot b$$

Otherwise, return

$$(b,\ p_0 + p_1 \cdot b)$$

as the argument, value pair for the global minimum.

If the degree is equal to 2,

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

If $p_2 > 0$ then return $(b,\ P(b))$ for

$$-\frac{p_1}{2 \cdot p_2} \geq b,$$

and return $(a,\ P(a))$ when

$$-\frac{p_1}{2 \cdot p_2} \leq b.$$

Otherwise, return

$$(c,\ P(c)),$$

where $c = -\frac{p_1}{2 \cdot p_2}$.

If the degree of the polynomial is larger than 2, then set

$$\textbf{Number\_of\_jumps} = 0,$$

$$x_0 = a.$$

STEP 1. Iterate

$$x_{k+1} = x_k + h,$$

as long as

$$q_{k0} + q_{k1} \cdot (x_k + h) + q_{k2} \cdot (x_k + h)^2 + \cdots + q_{kn-1} \cdot (x_k + h)^{(n-1)} \leq 0$$

and

$$x_k < b,$$

where $(q_{k0},\ q_{k1},\ \ldots\ q_{kn-1})$ are calculated with Horner's algorithm.

- $q_{kn-1} = p_n$
- $q_{ki-1} = x_k \cdot q_{ki} + p_i \quad \text{for} \quad i = 1,\ 2,\ \ldots n-1$

If $x_k \geq b$ then **STOP** and return b as the argument and

$$p_0 + p_1 \cdot b + p_2 \cdot b^2 + \cdots + p_n \cdot b^n$$

as the value for the estimate of the global minimum.

If

$$q_{k0} + q_{k1} \cdot (x_k + h) + q_{k2} \cdot (x_k + h)^2 + \cdots + q_{kn-1} \cdot (x_k + h)^{(n-1)} > 0$$

then proceed with **STEP 2**.

STEP 2. If LGA already performed $n-2$ evolutionary jumps, **Number_of_jumps** $\geq n-2$, then **STOP** and return x_k and

$$p_0 + p_1 \cdot x_k + p_2 \cdot x_k^2 + \cdots + p_n \cdot x_k^n$$

as the argument, value pair for the estimate of the global minimum.

Otherwise, recursively apply LGA to

$$q_{k0} + q_{k1} \cdot x + q_{k2} \cdot x^2 + \cdots + q_{kn-1} \cdot x^{(n-1)} \quad \rightarrow \quad \min_{x \in [x_k,\, b]} \tag{11}$$

Let $x_k^\star$ be the solution of (11). If

$$x_k^\star \geq x_k$$

and

$$q_{k0} + q_{k1} \cdot (x_k^\star) + q_{k2} \cdot (x_k^\star)^2 + \cdots + q_{kn-1} \cdot (x_k^\star)^{(n-1)} \geq 0$$

then **STOP** and return x_k,

$$p_0 + p_1 \cdot x_k + p_2 \cdot x_k^2 + \cdots + p_n \cdot x_k^n$$

as an argument, value estimate of the global minimum.

If

$$x_k^\star > x_k$$

and

$$q_{k0} + q_{k1} \cdot (x_k^\star) + q_{k2} \cdot (x_k^\star)^2 + \cdots + q_{kn-1} \cdot (x_k^\star)^{(n-1)} < 0$$

then, by Theorem 1, increment **Number_of_jumps** by one if x_k equals to a otherwise by two. After that LGA performs *an evolutionary leap* by setting

$$x_{k+1} = x_k^\star.$$

and proceding with **STEP 1**.

A polynomial

$$p_0 + p_1 \cdot x_k + p_2 \cdot x_k^2 + \cdots + p_n \cdot x_k^n$$

in LGA is evaluated with Horner's algorithm as follows.

Set

$$v = p_n.$$

Then iterate

$$v = v \cdot x + p_j \quad \text{for} \quad j = n-1,\, n-2,\, \ldots\, 1,\, 0$$

and v is the value of the polynomial at x.

Interested reader will find a snapshot of the working and thoroughly tested source code of LGA in Appendix of this paper.

5 Numerical experiments

This section presents the results of numerical experiments conducted in order to compare the performance of LGA with Brute-Force Search (BFS), Zero Derivative Method (ZDM), modifications of Piyavskii-Shubert Method (PSM) discussed in [3], [13], [14]. All numerical experiments follow the same scenario:

- Repeat 500 times **Step 1** and **Step 2**.

Step 1. Randomly generate a real polynomial $p(x)$ of degree n with roots uniformly distributed on $[-1,\ b]\times[-1,\ 1]$, where b is a real number between -1 and 1 which remains fixed across all 500 trials.

Step 2. Use LGA and its competitor to solve the optimization problem

$$p(x)\ \rightarrow\ \min_{[-1,\,1]}$$

with precision 0.0001 for x-argument of the global minimum on $[-1,\ 1]$. Record the processing time for LGA and its competitor.

- After repeating 500 times **Step1** and **Step2** calculate T_ℓ and T_c, the average processing time for LGA and its competitor respectively.

- Update the file with the experimental records by adding a new line $(n,\ T_\ell,\ T_c)$, where n is the degree of the polynomial.

The final result is presented in the form of two curves (average time spent versus polynomial degree), one for LGA and the other for its competitor.

Total time spent T_c includes all necessary supplementary steps that are needed in order to successfully implement the tested algorithm. For example, ZDM total time covers calculation of critical points with Laguerre's method and the subsequent search for the minimum among critical values. PSM time includes calculation of the Lipschitz constant or its counterparts.

5.1 BFS

BFS attacks

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

by transforming it into

$$f(a+(b-a)\cdot\frac{j}{N})\ \rightarrow\ \min_{0\le j\le N}$$

and then taking the smallest value in $\{f(a+(b-a)\cdot\frac{j}{N})\}_{j=0}^{N}$ and its respective x-argument as an estimate for the solution of the optimization problem.

LGA outperforms BFS for polynomials with roots uniformly distributed on $[-1,\ b]\times[-1,\ 1]$ where $-1<\ b<1$. LGA considerably speeds up as the value of the parameter b decreases (Fig. 2, Fig. 3).

If $b=1$ then LGA works exactly so well as BFS (Fig. 4, Fig. 5). A generic application corresponds to the situation with $b<1$. Therefore employing LGA instead of BFS will improve the performance of your application.

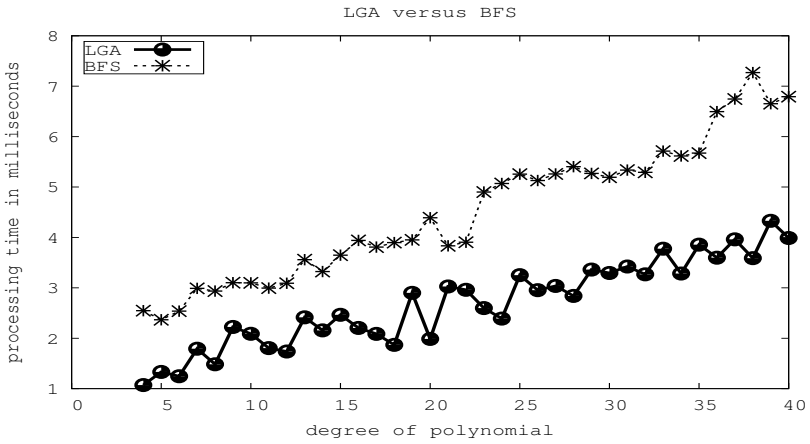

Figure 2: LGA versus Brute-Force Search, where $b = 0$.

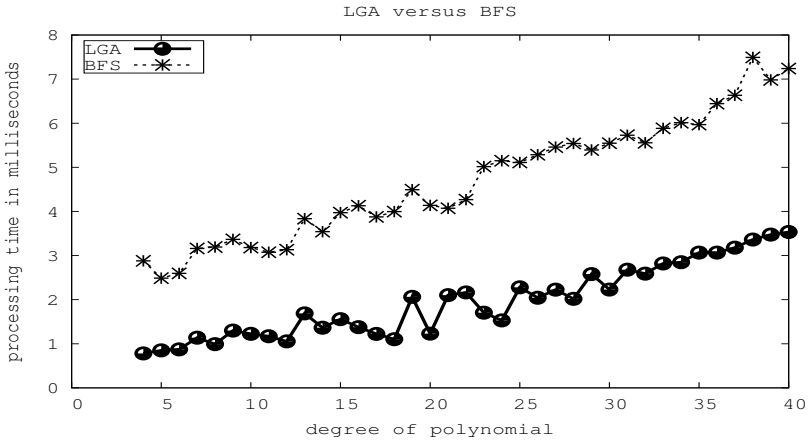

Figure 3: LGA versus Brute-Force Search, where $b = -0.5$.

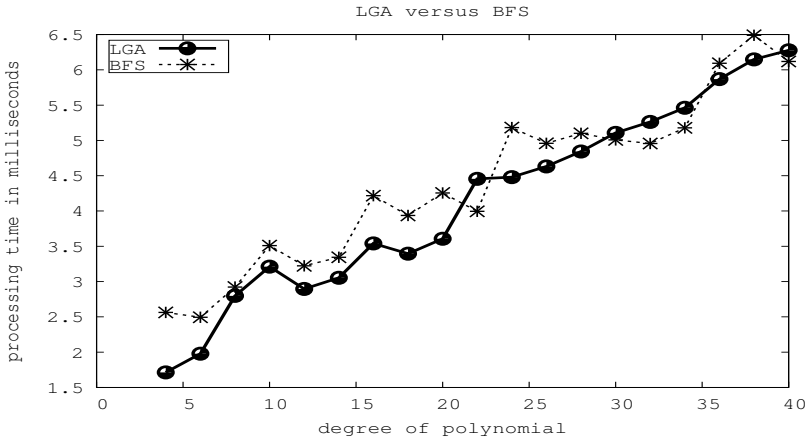

Figure 4: LGA versus Brute-Force Search for polynomials of even degrees and where $b = 1$.

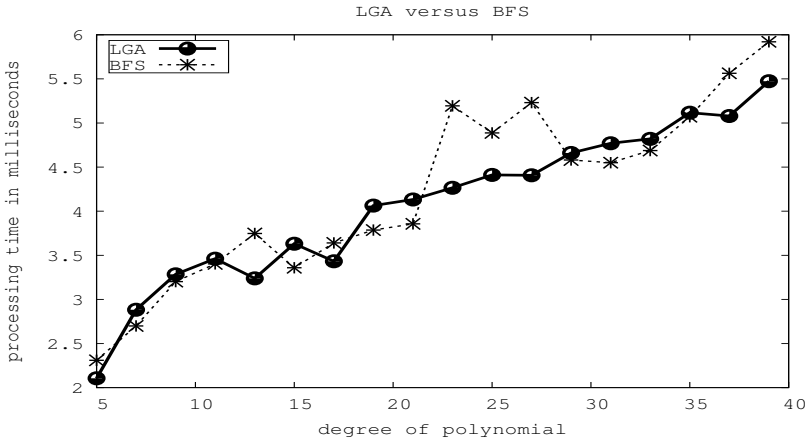

Figure 5: LGA versus Brute-Force Search for polynomials with odd degrees and where $b = 1$.

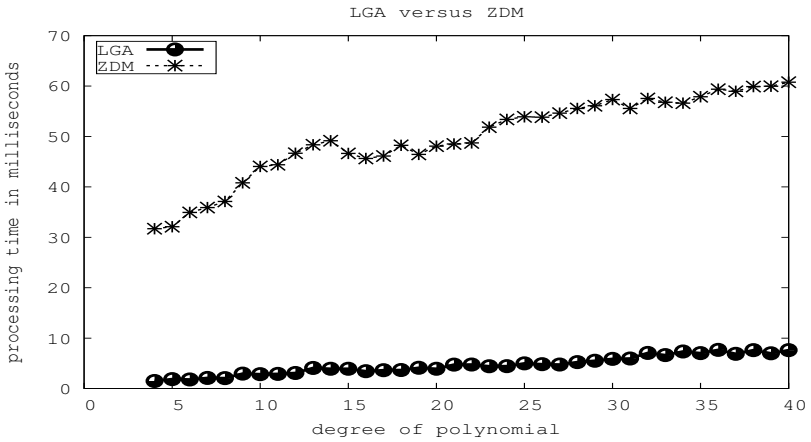

Figure 6: LGA versus ZDM and $b = 1$.

5.2 ZDM

ZDM finds the global minimum

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

by calculating zeroes $\{x_j\}_{j=1}^{M}$ of the derivative

$$\frac{d}{dx} f(x) = 0 \ \text{ for } \ x \ \in \ [a,\, b]$$

and then finding the smallest value in $\{f(x_j)\}_{j=1}^{M}$. If it is $f(x_j)$ then ZDM returns $(x_j,\ f(x_j))$ as an estimate for the solution of the optimization problem. In all numerical experiments reported in this paper the polynomial roots for ZDM were calculated with Laguerre's Method [1], [2]. The ZDM processing time includes the invocation of Laguerre's Method. LGA notably faster than ZDM (Fig. 6).

5.3 PSM and its accelerations

Piyavskii's type algorithms tackle the optimization problem

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

by constructing at each iteration either a piecewise linear ($f(x)$ is Lipschitz [10], [11], [13], [12]) or a piecewise quadratic ($\frac{d}{dx} f(x)$ is Lipschitz [3], [8], [13], [14]) auxiliary function $\Phi_n(x)$ such that

$$\Phi_n(x) \leq f(x) \ \ \forall\, x \in [a,\, b]$$

Then the original optimization problem is replaced with

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

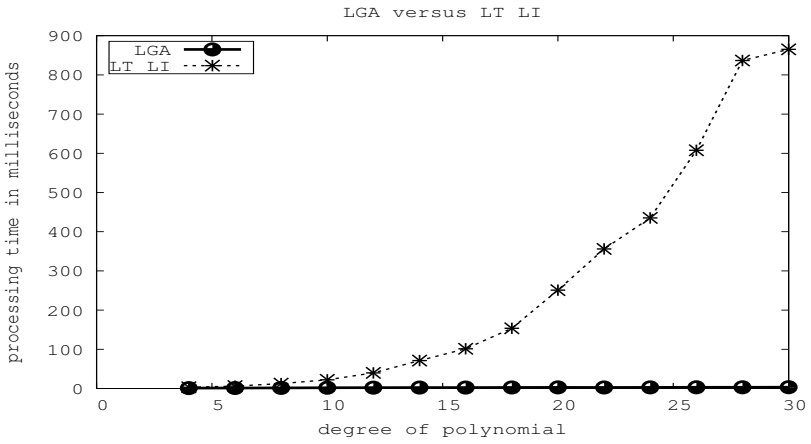

Figure 7: LGA versus LT LI for polynomials with even degrees and $b = 1$.

Based on its solution the algorithm either terminates or proceeds to the next step with the new refined auxiliary function $\Phi_{n+1}(x)$.

5.3.1 Modifications of PSM with local tuning of piecewise linear auxiliary functions

LGA is compared against PSM with tuning of the local Lipschitz constants (referred as LT) and its enhancement LT_LI presented in [13]. In view of the numerical simulations from [13] LT and LT_LI appear to be the fastest among Pyavskii's type algorithms (discussed in [13]) with piecewise linear auxiliary functions. LGA is faster than LT LI (Fig. 7).

5.3.2 Modifications of PSM with local tuning of piecewise quadratic auxiliary functions

Modifications of PSM with smooth piecewise quadratic auxiliary functions are discussed in [13]. Fig. 8 presents the comparison results between LGA and PSM enriched by local tuning of the Lipschitz constant for $\frac{d}{dx}f(x)$ (referred as DLT). LGA outperforms DLT.

DLT with some local improvement technique [13] is addressed as DLT_LI. Its comparison with LGA is presented by Fig.9. LGA is faster than DLT_LI.

5.3.3 Modifications of PSM with piecewise quadratic auxiliary functions

The paper [3] introduces the modification of PSM based on piecewise quadratic auxiliary functions that are not necessary smooth. The algorithm from [3] is referred in Fig.10 as EEK. LGA is faster than EEK.

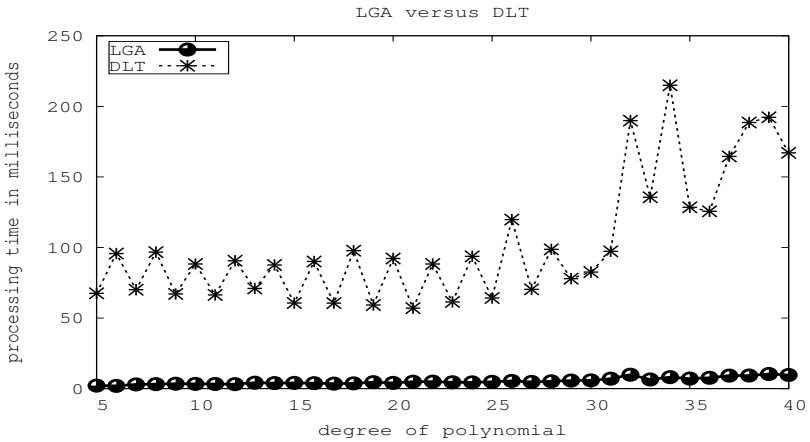

Figure 8: LGA versus DLT, $b = 1$.

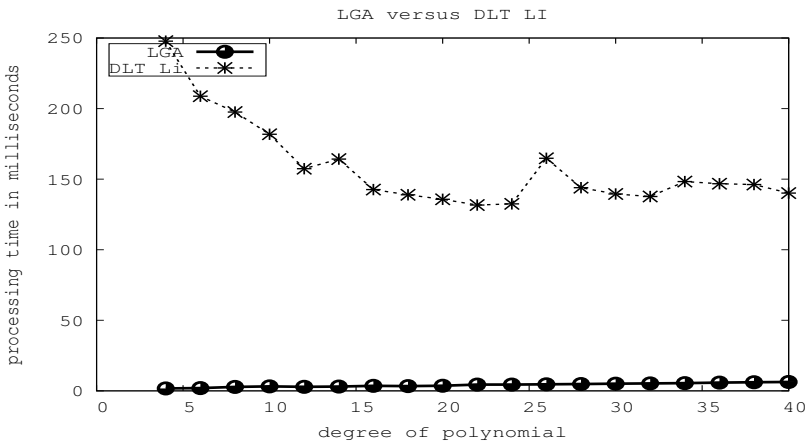

Figure 9: LGA versus DLT LI for polynomials with even degrees, $b = 1$.

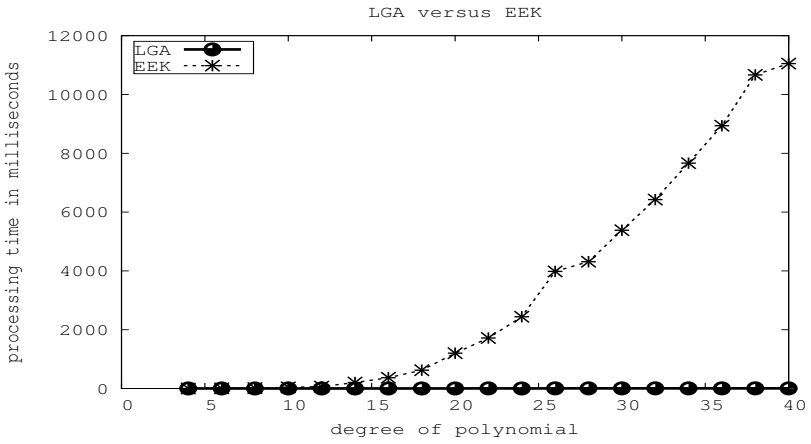

Figure 10: LGA versus EEK for polynomials with even degrees, $b = 1$.

6 Acknowledgment

The author is grateful to anonymous referees for comments and suggestions that helped to focus and improve the original manuscript. In particular, the comparison between LGA and Piyavskii-Shubert method was added upon a referee remark. The author is thankful to Blaiklen Marx for the help with preparing the manuscript for publication and testing the java implementations of algorithms in i-oblako.org framework.

7 Appendix

The program is looking for the global minimum of a polynomial on the interval $[a, b]$. The polynomial is represented as an array

```
double polynom = new double[degree+1];
```

A snapshot of the source code fragment essential for LGA is as follows.

```
public double getMin(double[] polynom,
                     double a, double b,
                            double step){
double rt=a;
double[] pl = null;
double rt_prev=a;

if(a >= b)
   return b;

if(b-a <= step)
   return a;
```

```
while(polynom[polynom.length - 1] == 0 && polynom.length > 1){
      pl= new double[polynom.length - 1];
      for(int i =0; i < polynom.length - 1;i++)
            pl[i]=polynom[i];
    polynom=pl;
}
if(polynom.length == 1)
  return a;
if(polynom.length == 2){
  if(polynom[1]>= 0 )
        return a;
  else
        return b;
}

rt=a;
rt_prev=rt;
int Njumps = 0;
    do{
          rt_prev = GradientDescent(polynom,rt,b,step);
            if(rt_prev >= b)
                  return b;

           if(Njumps >= (polynom.length - 3))
                  return rt_prev;

            pl = Horner(polynom,rt_prev);
            rt = getMin(pl,rt_prev,b,step);
        if(rt - rt_prev<= step)
             return rt_prev;
        else{
        if(rt_prev == a)
          Njumps++;
        else
          Njumps=Njumps + 2;
         }

          if(HornerEval(pl,rt)>=0)
                return rt_prev;

        rt_prev=rt;
      }
  while(rt < b-step);
}
```

and gradient descent is implemented as follows.

```
public double EvalDerivative(double[] polynom,double x){
          double ret = 0;

                for(int i = polynom.length -1; i>0; i--)
                      ret=ret*x + i*polynom[i];
                return ret;
}

public double GradientDescent(double[] polynom,
                                double a, double b,
                                      double step){

         double  rt=a;

           while(EvalDerivative(polynom,rt)<0 && rt < b )
                                rt=rt+step;
              return rt;
}
```

Java implementations of the algorithms discussed in the paper are available upon request.

References

[1] D.A. Adams, "A Stopping Criterion for Polynomial Root Finding." Comm. ACM,10, 655-658, (1967).

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

[3] R. Ellaia, M. Z. Es-Sadek, H. Kasbioui, Modified Piyavskii's Global One-Dimensional Optimization of a Differentiable Function, Applied Mathematics, 3, 1306-1320 (2012)

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

[5] O. Güler, Foundations of Optimizations, Springer, New York (2010)

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

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

[8] D.E. Kvasov, Y.D. Sergeyev, A Univariate Global Search Working With a Set of Lipschitz Constants for the First Derivative, Optim. Lett., 3, 303-318 (2009)

[9] Yu. Nesterov, Introductory Lectures on Convex Optimization, Applied Optimization, 87, Kluwer Academic Publishers, Boston (2004)

[10] S. Piyavskii, An algorithm for finding the absolute minimum of a function, Theory of Optimal Solutions, IK Akad. Nauk USSR, Kiev, 2, 13-24, (1967)

[11] S. Piyavskii, An algorithm for finding the absolute extremum of a function, USSR Comput. Math.Math. Phys., 12, 57-67, (1972)

[12] B. Shubert, A sequential method seeking the global maximum of a function, SIAM J. Numer. Anal., 9, 379-388 (1972)

[13] D. Lera and Y. D. Sergeyev, Acceleration of Univariate Global Optimization Algorithms Working With Lipschitz Functions and Lipschitz First Derivatives, SIAM J. Optim., 23, 1, 508-529 (2013)

[14] Y. D. Sergeyev, Global one-dimensional optimization using smooth auxiliary functions, Math. Programming, 81, 127-146 (1998)