IJPAS ISSN: 2149-0910 | INTERNATIONAL JOURNAL PURE AND APPLIED SCIENCES

Vol.10(1); 209-227 (2024)
DOI: 10.29132/ijpas.1467644

Research Article

Exponential-Quadratic-Logarithmic Composite Function Optimization In Positive Domains: Leveraging Multiplicative Calculus In Gradient Descent Algorithms

Erkan KIYMIK*, Ali Emre ÖZTÜRK

Hasan Kalyoncu University, Faculty of Engineering, Department of Electrical and Electronics Engineering, Gaziantep, Türkiye; erkan.kiymik@hku.edu.tr; emre.ozturk@hku.edu.tr
*****Correspondencing Author

Received: 12. 04. 2024
Accepted: 16. 06. 2024

Abstract: This work investigates the integration of multiplicative calculus into gradient descent algorithms, including Adaptive Gradient algorithm (AdaGrad), Root Mean Squared Propagation (RMSProp), Nesterov Accelerated Gradient (NAG), and Momentum, to optimize exponential-quadratic-logarithmic composite functions with the positivity constrained. This research, conducted across five scenarios within the Constrained and Unconstrained Testing Environment (CUTEst), compares these multiplicative methods with their classical counterparts under a variety of constraints environments such as bounded, quadratic, and other types, and unconstrained environments. The results demonstrate the significant superiority of multiplicative-based algorithms, especially in unconstrained and bounded constrained scenarios, and demonstrate their potential for complex optimization tasks. Statistical analysis supports the observed performance advantages, indicating significant opportunities for optimization strategies in positive domains.

Keywords: Gradient descent; optimization; machine learning; multiplicative calculus; composite functions

Pozitif Alanlarda Üstel-İkinci Dereceden Logaritmik Bileşik Fonksiyon Op-timizasyonu: Gradyan İniş Algoritmalarında Çarpımsal Kalkülüsten Yararlanma

Özet: Bu çalışma, pozitiflik kısıtlamalı üstel-logaritmik bileşik fonksiyonları optimize etmek için, Uyarlanabilir Gradyan algoritması (AdaGrad), Kök Ortalama Kare Yayılımı (RMSProp), Nesterov Hızlandırılmış Gradyan (NAG) ve Momentum gibi gradyan iniş algoritmalarına Çarpımsal Kalkülüsün entegrasyonunu araştırmaktadır. Kısıtlamalı ve Kısıtlamasız Test Ortamı (CUTEst) içinde beş senaryoda yürütülen bu araştırma, Çarpımsal Kalkülüs ile revize edilen gradyan iniş algoritmalarıyla, sınırlı, karesel ve diğer türler gibi çeşitli kısıtlama ortamlarının yanı sıra kısıtlamasız ortamlarda klasik kalkülüsteki gradyan iniş algoritmalarıyla karşılaştırmaktadır. Sonuçlar, özellikle kısıtlamasız ve sınırlı kısıtlamalı senaryolarda çarpan tabanlı algoritmaların belirgin üstünlüğünü ve karmaşık optimizasyon görevleri için potansiyellerini göstermektedir. İstatistiksel analiz, gözlemlenen performans avantajlarını destekleyerek, pozitif alanlarda optimizasyon stratejileri için önemli fırsatlar olduğunu göstermektedir.

Anahtar Kelimeler: Dereceli alçalma; optimizasyon; makine öğrenme; çarpımsal kalkülüs; bileşik fonksiyonlar

1. Introduction

Setting the Context:

Gradient descent-based algorithms, which are at the core of the optimization field, have found a place in many fields such as machine learning and applied sciences due to their ease of use and effectiveness [1]. Variants of these algorithms, which use the principles of Newtonian calculus, have been researched and optimized many times over the years. However, as real-world problems constantly evolve and create new situations that need to be optimized, these basic methods require updating and improvement.

Identifying the Problem:

One such avenue for innovation occurs in the optimization of functions restricted to positive domains, especially functions containing exponential, quadratic, and logarithmic properties. Classical gradient descent methods, although versatile, often face limitations in these contexts, mainly due to their inherent linear additive nature and the challenges posed by non-negative constraints [2].

Introducing the Novel Approach:

This study deals with integrating Multiplicative Calculus (MC) principles into gradient descent variants and making them more suitable for exponential-quadratic-logarithmic composite functions that are limited to positive areas. It brings a new perspective to optimization for positively bounded areas. An alternative to Newtonian Calculus, Multiplicative Calculus offers a different perspective by focusing on multiplication and ratios rather than addition and subtraction. This inherently makes it suitable for positive fields and multiplicative growth processes.

Rationale and Methodology:

To exploit this potential of MC, we adapted several classical gradient descent algorithms into the multiplicative calculus framework, including RMSProp, AdaGrad, Momentum, and NAG [3–5]. To apply the methods comparatively, exponential-quadratic-logarithmic composite functions were created using the CUTEst test set [6]. We developed Exponential-Quadratic-Logarithmic Composite Functions as our objective functions, $g(x_n)$, by transforming the quadratic objective functions provided by CUTEst's testing suite. The transformation process involves taking a function $f(x_n)$ from CUTEst and applying the transformation $g(x_n)=e^{(f(\log(xn)))}$ to it.

Objective and Scope:

Through extensive testing in the CUTEst optimization environment, this research aims to systematically evaluate the performance of these multiplicative-based methods relative to their classical counterparts in various scenarios characterized by different iterations and learning rates. 5 scenarios were used: 1- 1000 iterations, 0.001 learning rate, 2- 10 iterations, 0.1 learning rate, 3- 10 iterations, 0.01 learning rate, 4- 10 iterations, 0.001 learning rate, 5- 10 iterations, 0.0001 learning rate. Comparisons were made by categorizing the composite function used as an unconstrained, bounded constraint, quadratic constrained, and other type of constrained. In the results section, we present comparison results between alternative methods and their counterparts. Additionally, we include performance comparisons of these methods in both constrained and unconstrained scenarios. We also provide detailed statistical analysis, including t-statistics and p-values, to further validate the benchmark.

Significance of the Work:

The results of this research can go beyond theoretical knowledge and offer a new perspective from which optimization problems can be viewed and solved. By demonstrating the effectiveness of methods based on multiplicative analysis, especially in the context of exponential-quadratic-logarithmic composite functions, this study opens the door to innovative optimization strategies that can significantly improve problem-solving in areas where positive domain constraints are common.

Structure of the Paper:

The paper's structure is designed for clarity and simplicity, organized as follows:

- Introduction: Sets the stage, outlining the research question and its significance.
- Related Works: Reviews existing literature to contextualize the study within the broader field.
- Definitions of Multiplicative Calculus: Explains key concepts and mathematical foundations of multiplicative calculus.
- Revised Methods: Describes the adaptations of classical algorithms using multiplicative calculus principles.
- Results: Presents findings from testing the revised methods on various functions and scenarios.

- Discussion: Analyze the implications of the results, comparing the performance of revised and classical methods.
- Conclusion: Summarizes the study's contributions, and limitations, and suggests directions for future research.

2. Related Works

2.1. Gradient Descent Methods

Since the gradient descent algorithm is at the core of optimization, it has been and is being studied a lot. Thanks to these extensive studies, various variants such as AdaGrad, RMSProp, Momentum, and NAG have emerged [7]. These variants are obtained by integrating innovations such as adaptive learning rate or momentum to increase convergence rate and stability. Momentum includes a velocity component that incorporates past gradients to accelerate convergence in relevant directions while reducing oscillations and proves particularly effective in navigating valleys and plateaus in the target landscape [8]. AdaGrad improves performance on sparse gradient problems by providing adaptive learning rates for each parameter, allowing individualized parameter updates, thus facilitating a more refined optimization process [9]. NAG improves the concept of momentum by integrating a forward-looking mechanism that predicts future gradients to make more informed update decisions, thus improving convergence rates. RMSProp, attributed to Geoffrey Hinton, modifies AdaGrad's aggressive diminishing learning rates by maintaining a moving average of recent squared gradients, ensuring sustained learning progress across iterations.

2.2. Multiplicative Calculus

Multiplicative calculus, developed as an alternative to Newtonian calculus, was introduced by Grossman and Katz in the early 1970s. While classical calculus, that is, Newtonian calculus, is based on addition and subtraction, in Multiplicative calculus, derivative and integral are defined by multiplication and division operations. This framework offers a unique perspective for problems where growth processes are multiplicative rather than additive in nature.

The theoretical foundations of multiplicative calculus were further developed by Bashirov and others, who extended the concepts of multiplicative differentiation and integration, demonstrated their applicability to a variety of mathematical functions, and established the fundamental theorems of multiplicative calculus [10, 11]. These works laid the foundation for the field's mathematical rigor and consistency.

Multiplicative calculus has found use in areas where the examined phenomenon grows or decays exponentially or geometrically and has been shown to provide advantages over Newtonian calculus [12–15]. For example, in the field of biomedical sciences, multiplicative analysis has been applied to model the growth of biological tissues and populations [16]. It offers a new approach to modeling compound interest and exponential growth processes by providing more accurate and intuitive explanations than traditional analyses in economics [17].

Although the application of multiplicative calculus in optimization is relatively new, its potential is significant. The inherently multiplicative nature of many optimization problems, especially those involving exponential growth or decay, suggests that multiplicative calculus could offer more natural and efficient optimization algorithms. Investigating the principles of multiplicative calculus within gradient descent algorithms represents a new research direction that may provide insights into more effective optimization strategies.

Recent studies have begun to explore the integration of multiplicative calculus with computational algorithms, including its potential in numerical analysis and algorithmic design [13]. A study highlights the utility of multiplicative calculus in biomedical imaging, particularly its ability to preserve positivity in images and matrix fields, a crucial feature for applications such as diffusion tensor imaging [18]. Another research study presents a multiplicative calculus adaptation of the Runge-Kutta method, demonstrating its effectiveness and broader applicability in solving problems involving positive-valued functions, complemented by extensive error and stability analyses [19, 20]. In addition, a review of wave propagation models reveals a significant advance in simulation methodologies by revealing how multiplicative calculus, with its different approach to exponentials, can increase solution efficiency and overcome traditional sampling limitations [21].

The adoption of MC in the broader mathematical and scientific communities faces challenges. First of all, it differs from traditional mathematical principles and needs further theoretical development and practical verification. Future research will focus on introducing algorithms, empirical testing in optimization, and exploring uses in machine learning and data analysis.

Multiplicative Calculus is a field that is open to exploration and has the potential to provide advantages in different fields. As scientists' attention is drawn to this field, it may provide new insights, methodologies, and applications that push and expand the boundaries of multiplicative calculus.

2.3. Optimization in Positive Domains

Optimizing functions constrained to positive domains presents unique challenges, often requiring transformations or adaptations of standard methods to maintain feasibility and convergence [22]. Exponential-quadratic-logarithmic composite functions, characterized by their strictly positive output and exponential growth behavior, are typical examples of such difficulties. In the literature, positive bounded problems have been discussed, but this situation inherent in Multiplicative calculus has not been investigated in the literature.

2.4. Our Contribution

Building on the foundational work in both gradient descent methodologies and multiplicative calculus, our research introduces a novel integration of these domains. By adapting classical gradient descent algorithms to the multiplicative calculus framework and applying them to exponential-quadratic-logarithmic composite functions, we present a unique approach to optimization in positive domains. Our work extends the current understanding of gradient descent's applicability, providing empirical evidence of the advantages offered by a multiplicative-based approach, particularly in handling functions with inherent exponential growth characteristics within bounded constraints.

3. Definitions of Multiplicative Calculus

This section introduces the definitions of MC used in this paper [11].

Let $\mathbb{R}_* = (0, \infty)$

Definition 3.1 Multiplicative addition operation is defined in the following manner.

$$a +_* b = a \cdot b. \tag{3.1}$$

Definition 3.2 Multiplicative multiplication is shown as $\cdot_*$ and the operation is performed in the following manner.

$$a \cdot_* b = e^{\log(a) \cdot \log(b)}. \tag{3.2}$$

Definition 3.3 Multiplicative zero and multiplicative one are changed to *1* and *e* respectively. Some of the properties of multiplicative summation and calculation is given below:

- Commutativity of MC Addition: Let $x, y \in \mathbb{R}_*$ then

$$x +_* y = y +_* x. \tag{3.3}$$

- Associativity of MC addition: Let $x, y, z \in \mathbb{R}_*$ then

$$x +_* (y +_* z) = (x +_* y) +_* z. \tag{3.4}$$

- Identity element of MC addition: Let $x \in \mathbb{R}_*$ then

$$x +_* 1 = x. \tag{3.5}$$

- Inverse elements of MC addition: Let $x \in \mathbb{R}_*$ then

$$-_* x = \frac{1}{x}. \tag{3.6}$$

- Identity elements of MC multiplication: Let $x \in \mathbb{R}_*$ then

$$x \cdot_* e = e^{\log(e)\cdot\log(b)} = x. \tag{3.7}$$

- Inverse elements of MC multiplication: Let $x \in \mathbb{R}_*$ then

$$x^{-1_*} = e. \tag{3.8}$$

- Distributivity.: Let $x, y, z \in \mathbb{R}_*$ then

$$(x +_* y) \cdot_* z = (x \cdot_* z) + (y \cdot_* z). \tag{3.9}$$

Definition 3.4 Let $x, y \in \mathbb{R}_*$, MC subtraction is shown as $-_*$, operation is defined in the following manner;

$$x -_* y = x +_* (-_* y) = x +_* \left(\frac{1}{y}\right) = \frac{x}{y}. \tag{3.10}$$

Definition 3.5 Let $x, y \in \mathbb{R}_*$, MC division is shown as $/_*$ and operation is defined in the following manner;

$$x /_* y = x \cdot_* (y^{-1_*}) = x \cdot_* \left(e^{\frac{1}{\log(y)}}\right) = e^{\log(x)\cdot\log\left(e^{\frac{1}{\log(y)}}\right)}, \tag{3.11a}$$

$$= e^{\frac{\log(x)}{\log(y)}}. \tag{3.11b}$$

Definition 3.6 Let $n \in N$, MC factorial process is shown as $!_*$ and operation is defined in following manner;

$$n!_* = e^{n!}. \tag{3.12}$$

Definition 3.7 Let $k \in N$ and $x \in \mathbb{R}_*$, MC power operation is shown below.

$$x^{k_*} = e^{(log(x))^k}. \tag{3.13}$$

Definition 3.8 Let $A \subseteq \mathbb{R}_*$ and first MC derivative is defined of f at $x \in A$, shown as $f^*(x)$, derivative operation is following manner;

$$f^*(x) = \lim_{h \to 0^*} (f(x +_* h) -_* f(x)) /_* h,$$

$$= \lim_{h\to 0^*} (f(x\cdot h) -_* f(x)) /_* h,$$

$$= \lim_{h\to 1} (f(x\cdot h) -_* f(x)) /_* h,$$

$$= \lim_{h\to 1} \left(\frac{f(x\cdot h)}{f(x)}\right) /_* h,$$

$$= \lim_{h\to 1} \left(e^{\frac{log\left(\frac{f(x\cdot h)}{f(x)}\right)}{log(h)}}\right),$$

$$= \lim_{h\to 1} \left(e^{\frac{x\cdot h\cdot f(x)\cdot f'(x\cdot h)}{f(x\cdot h)\cdot f(x)}}\right),$$

$$f^*(x) = e^{\frac{x\cdot f\prime(x)}{f(x)}} \quad , \quad x \in A. \tag{3.14}$$

Multiplicative calculus operates exclusively on positive numbers, ensuring that its derivative always falls within the range of 0 to positive infinity [11]. In the context of multiplicative calculus, it is imperative to consider three distinct cases of the derivative. The first case, when f(x)* > 1, leads to a decrease in f(x). The second case, where f(x)* = 1, signifies the identification of optimal points, mirroring the scenario in classical calculus where the derivative equals 0. Lastly, when f(x)* > 1, it results in an increase in f(x).

4. Revised Methods

4.1. Gradient Descent

In classical calculus, gradient descent algorithms adjust weights through a combination of learning rate and derivative. The derivative in classical calculus can take values from negative infinity to positive infinity. The Classical Gradient Descent (GD) formula for classical calculus is illustrated in Equation 4.1. Where θ is parameters that should be updated, α is the learning rate, $J'(\theta)$ is derivative of the cost function.

$$\theta_{new} = \theta_{old} - \alpha \cdot J'(\theta). \tag{4.1}$$

When Equation 4.1 is examined, it is seen that the old and new parameters are equal to each other when the derivative is equal to zero. However, in the context of generating a Multiplicative Gradient Descent algorithm, it is imperative to acknowledge that the multiplicative derivative is bounded within the interval (0, ∞). This means that the multiplicative derivative cannot be directly integrated into Equation 4.1. For the use of the MC derivative, Equation 4.1 needs to be revised with MC principles. When the GD algorithm is adapted to include multiplicative calculus, it results in the formulation of Equation 4.2.

$$\theta_{new} = \frac{\theta_{old}}{\alpha^{\log_e(J^*(\theta))}} \;. \tag{4.2}$$

When Equation 4.2 is examined, it is seen that when the multiplicative derivative is equal to 1, the new and old parameters are equal to each other. The second crucial observation arises when the multiplicative derivative exceeds 1 or falls below 1. In cases where the learning rate is less than 1, the update process does not yield accurate results. Hence, it becomes imperative to set the update magnitude greater than 1 for proper convergence. For the comparative analysis between classical calculus gradient descent and multiplicative gradient descent, the learning rate for classical gradient descent is denoted as 'α', while the learning rate for multiplicative gradient descent is set as $' \propto = e^{\alpha}'$. After implementing the requisite corrections, the Multiplicative Gradient Descent algorithm is presented in Equation 4.3b.

$$\theta_{new} = \frac{\theta_{old}}{\propto^{\log_e e^{(\frac{\theta \cdot J\prime(\theta)}{J(\theta)})}}}, \tag{4.3a}$$

$$\theta_{new} = \frac{\theta_{old}}{\propto^{(\frac{\theta \cdot J\prime(\theta)}{J(\theta)})}} . \tag{4.3b}$$

4.2. Momentum

The formula for the Momentum update rule in gradient descent is as follows:

$$\vartheta_{t+1} = \beta \cdot \vartheta_t + \alpha \cdot J'(\theta_t), \tag{4.4}$$

$$\theta_{t+1} = \theta_t - \vartheta_{t+1}. \tag{4.5}$$

Where:

- v_t is the momentum term at time step *t*.
- β is a hyperparameter that controls the strength of the momentum.
- *J'(θ)* is the gradient of the loss function at time step *t*.
- θ_t is the parameter vector at time step *t*.
- α is the learning rate.

This update rule introduces a momentum term, which is a fraction of the previous update, to smooth out oscillations in the updates and accelerate convergence in relevant directions.

The momentum update is aligned with the principles of MC, as illustrated in Equations 4.6b and 4.7 below.

$$\vartheta_{t+1} = \beta^{\log_e \vartheta_t} \cdot \alpha^{\log_e J^*(\theta_t)}, \tag{4.6a}$$

$$\vartheta_{t+1} = \beta^{\log_e \vartheta_t} \cdot \alpha^{\log_e e^{\frac{\theta \cdot J\prime(\theta)}{J(\theta)}}}, \tag{4.6b}$$

$$\theta_{t+1} = \theta_t / \vartheta_{t+1}. \tag{4.7}$$

In transitioning from VGD to the MC domain, it becomes apparent that the learning rate requires an adjustment to $\propto = e^{\alpha}$. In classical calculus, the momentum parameter typically falls within the range of 0 to 1. Consequently, to effectively apply momentum in the MC context, it is imperative to utilize $\psi = e^{\beta}$.

$$\propto = e^{\alpha}, \quad \psi = e^{\beta}, \tag{4.8a,4.8b}$$

$$\vartheta_{t+1} = \psi^{\log_e \vartheta_t} \cdot \propto^{\frac{\theta \cdot J\prime(\theta)}{J(\theta)}}, \tag{4.9}$$

$$\theta_{t+1} = \theta_t / \vartheta_{t+1}. \tag{4.10}$$

In the experiments, the initialization of ϑ differs based on the calculus approach. In classical calculus, ϑ is set to 0, whereas in the context of MC, ϑ is initialized to 1.

4.3 NAG

One notable distinction between NAG and standard Momentum lies in its computation of the gradient, projected ahead in the direction of momentum. This feature substantially mitigates oscillations and bolsters convergence, particularly in situations characterized by high curvature. The NAG update protocol is delineated by Equations 4.11, 4.12 and 4.13.

$$\tau = \theta_t - \beta \cdot \vartheta_t, \tag{4.11}$$

$$\vartheta_{t+1} = \beta \cdot \vartheta_t + \alpha \cdot J'(\tau), \tag{4.12}$$

$$\theta_{t+1} = \theta_t - \vartheta_{t+1}. \tag{4.13}$$

Where:

- v_t is the momentum term at time step t.
- β is a hyperparameter that controls the strength of the momentum.
- $J'(\theta)$ is the gradient of the loss function at the parameter vector θ.
- θ_t is the parameter vector at time step t.
- α is the learning rate.

Applying the formulations outlined in Equations 4.8a and 4.8b, along with the definitions delineated in Section 2, we derive the expressions for MC-NAG, which are subsequently presented in Equations 4.15c and 4.16.

$$\tau = \frac{\theta_t}{\psi^{\log_e \vartheta_t}}, \tag{4.14}$$

$$\vartheta_{t+1} = \psi^{\log_e \vartheta_t} \cdot \propto^{\log_e J^*(\tau)}, \tag{4.15a}$$

$$= \psi^{\log_e \vartheta_t} \cdot \propto^{\log_e e^{\frac{\tau \cdot J'(\tau)}{J(\tau)}}}, \tag{4.15b}$$

$$= \psi^{\log_e \vartheta_t} \cdot \propto^{\frac{\tau \cdot J'(\tau)}{J(\tau)}}, \tag{4.15c}$$

$$\theta_{t+1} = \frac{\theta_t}{\vartheta_{t+1}} . \tag{4.16}$$

In classical calculus, ϑ is set to 0, whereas in the context of MC, ϑ is initialized to 1.

4.4. AdaGrad

AdaGrad's fundamental concept lies in adjusting learning rates according to the historical gradients associated with each parameter. This adaptive approach proves beneficial in scenarios where certain parameters necessitate more conservative updates while others warrant more substantial adjustments. The update process for Classical AdaGrad is detailed below. The accumulated squared gradient G_t for a parameter θ at time step t is updated based on the gradient of the loss function with respect to that parameter $J(\theta_t)$. The accumulated squared gradient G_t is shown in Equation 4.17.

$$G_t = G_{t-1} + (J'(\theta_t))^2. \tag{4.17}$$

This means that at each time step, the squared gradient is added to the previous accumulated squared gradient. For the learning rate, at each time step t, the learning rate α_t is determined by dividing the initial learning rate α_0 by the square root of the accumulated squared gradients G_t plus ϵ.

$$\alpha_t = \alpha_0 / \sqrt{G_t + \epsilon}. \tag{4.18}$$

This means that the learning rate is reduced for parameters that have received large updates in the past and increased for parameters that have received smaller updates. The update for the parameter θ at time step *t* is then performed using the calculated learning rate:

$$\theta_t = \theta_{t-1} + \alpha_t (J'(\theta_t)) . \tag{4.19}$$

In the initial step, the Accumulated Gradient undergoes an update. In the classical calculus framework, the summation operation for the Accumulated Squared Gradient is transformed into a multiplication operation, thereby optimizing the Accumulated Gradient.

$$G_t = G_{t-1}\, e^{(\ln J^*(\theta_t))^2}. \tag{4.20}$$

In the classical AdaGrad approach, the gradient is initialized with a value of 0 for accumulation. In MC-AdaGrad, a distinctive initialization of 1 is employed, reflecting the multiplicative nature of the method. Subsequently, the employment of multiplicative derivatives is imperative in the formulation. Upon the incorporation of the MC derivative and subsequent necessary simplifications, we arrive at Equation 4.21.

$$G_t = G_{t-1} \cdot e^{ln\left(\frac{\theta \cdot J'(\theta)}{J(\theta)}\right)^2}. \tag{4.21}$$

The subsequent crucial step involves the refinement of the adaptive learning rate equation. In MC gradient descent, conventional learning rates within the range of 0 to 1 are not suitable, given the unique nature of MC, which corresponds to negative values in classical calculus. Consequently, the exponential function e^a is employed to define the learning rate within the MC calculus framework. Following this adjustment, the equation is aligned with the definition provided in Section 2, resulting in the derivation of Equation 4.23.

$$\propto = e^{\alpha}, \tag{4.22}$$

$$\propto_t = \propto_0{}^{\log_e(1/e^{(\log_e G_t \cdot \epsilon)^{0.5}})}. \tag{4.23}$$

The final step in the MC-AdaGrad methodology involves the parameter update, succinctly represented by Equation 4.24 below.

$$\theta_t = \theta_{t-1} \cdot \propto^{ln(\frac{\theta \cdot J'(\theta)}{J(\theta)})} . \tag{4.24}$$

4.5. RMSProp

RMSProp is an optimization algorithm designed to dynamically adjust learning rates for individual parameters. It tackles the diminishing learning rates challenge observed in AdaGrad by incorporating a moving average of squared gradients. The moving average employed in Newtonian calculus needs to be adapted to the geometric mean for its effective integration into MC. The update formula for parameter θ at time step *t* in RMSProp is expressed as:

$$G_t = \beta G_{t-1} + (1 - \beta) \cdot (J'(\theta_t))^2, \tag{4.25}$$

$$\theta_t = \theta_{t-1} - (\alpha \cdot (J'(\theta_t))/\sqrt{G_t + \epsilon}. \tag{4.26}$$

- α is the learning rate.
- $J'(\theta_t)$is the gradient of the loss function with respect to θ at time step *t*.
- $G_t$is the moving average of squared gradients for parameter θ at time step *t*.
- β is a hyperparameter that controls the exponential decay of the moving average.

- ϵ is a small constant.

Adapting RMSProp to the principles of MC is essential. Initially, parameters such as learning rate and decay should be initialized as the exponent of 'e'. Subsequently, operations should be updated in accordance with the definitions outlined in Section 2 and by using Equations 4.8a and 4.8b.

$$J^*(\theta_t) = e^{\frac{\theta \cdot J\prime(\theta)}{J(\theta)}}, \tag{4.27}$$

$$G_t = \psi^{ln\, G_{t-1}} \cdot (1/\psi)^{ln\, e^{(ln((J^*(\theta_t)))^2}}, \tag{4.28}$$

$$\theta_t = \theta_{t-1} /(\propto^{ln(J^*(\theta_t))^{ln(1/(e^{(ln(G_t \cdot \epsilon))^{0.5}}))}}). \tag{4.29}$$

5. Results

Classical gradient descent algorithm, NAG, Momentum, AdaGrad, and RMSProp algorithms, along with their multiplicative analysis-based counterparts, were evaluated in five different scenarios on exponential-quadratic-logarithmic composite functions subject to various constraints: bounded, quadratic, others, and unbounded. The scenarios have been deliberately chosen to cover a wide range of optimization conditions:

Scenario 1: Iteration Count: 1000, Learning Rate: 0.001

- Reason for Iteration Count: The high number of iterations allows the algorithm to comprehensively explore the optimization environment and find a global or near-global optimum. This is especially useful for complex functions with many local minima.
- Reason for Learning Rate: The reason why the learning coefficient is chosen small is to test the algorithms in high iterations in a controlled manner without exceeding the optimum value.

Scenario 2: Iteration Count: 10, Learning Rate: 0.1

- Reason for Iteration Count: A low number of iterations tests the ability of the algorithm to converge quickly. This scenario is designed to evaluate the algorithm's performance in situations where computational resources or time are limited.
- Reason for Learning Rate: A high learning rate forces larger parameter updates, testing the algorithm's ability to make significant progress with each iteration.

Scenario 3: Iteration Count: 10, Learning Rate: 0.01

- Reason for Iteration Count: Similar to Scenario 2, the low number of iterations tests the performance of the algorithm with minimal computational effort.
- Reason for Learning Rate: A moderate level of learning was chosen to test the behavior of the algorithms when they need both significant updates and stability at the same time.

Scenario 4: Iteration Count: 10, Learning Rate: 0.001

- Reason for Iteration Count: Again low number of iterations to test fast convergence capabilities.
- Reason for Learning Rate: A small learning rate similar to Scenario 1, but with fewer iterations to see how well the algorithm performs with minimal updates over a short period.

Scenario 5: Iteration Count: 10, Learning Rate: 0.0001

- Reason for Iteration Count: Low iteration count for rapid convergence testing.

- Reason for Learning Rate: An extremely small learning rate tests the performance of the algorithm with very careful updates.

Collectively, these scenarios were selected to conduct a comprehensive investigation to understand how algorithms behave at high iteration counts, aggressive-rapid changes, sensitive updates, and the need for stability. This comprehensive approach ensures that the study's findings are applicable to a wide range of practical optimization problems and highlights the versatility and potential advantages of MC-based algorithms. Table 1 displays the conducted scenarios.

Table 1. *Scenarios of Experiments.*

Scenarios	Iteration count	Learning rate
1	1000	0.001
2	10	0.1
3	10	0.01
4	10	0.001
5	10	0.0001

In the experimental setup, the parameters of the classical and alternative methods were set as follows:

- β = 0.9 for Classical Momentum.
- β = 0.9 for Classical NAG.
- $\epsilon = 10^{-8}$ for Classical AdaGrad.
- $\epsilon = 10^{-8}$ for Classical RMSProp.
- β = 0.9 for Classical RMSProp.
- $\beta = e^{0.9}$ for MC-Momentum.
- $\beta = e^{0.9}$ for MC-NAG.
- $\epsilon = e^{10^{-8}}$ for MC-AdaGrad.
- $\epsilon = e^{10^{-8}}$ for MC- RMSProp.
- $\beta = e^{0.9}$ for MC-RMSProp.

In the experimental design, starting points were established using the exponential of CUTEst's default starting values, aligning with the inherently positive domain of the composite functions under study. This adjustment ensured that both algorithms commenced from equivalent starting points, providing a consistent and stable foundation for comparison within the experimental framework.

Classical Gradient Descent:

In this analysis, classical gradient descent was evaluated across 405 instances. Of these, the multiplicative gradient descent approach remained convergent in 278 cases, while its classical counterpart did so in 258 instances. Among the scenarios where both methods achieved convergence, the alternative multiplicative method yielded superior results in 124 cases. Conversely, the classical approach outperformed the alternative in 17 instances, and both methods arrived at identical solutions in 74 cases. Statistical analysis further substantiated these findings, with t-statistics and p-values recorded at 3.0469 and 0.0026, respectively, indicating the significance of the observed differences.

Table 2. *Comparison of Classical Gradient descent and MC-Gradient Descent*

	Classical Gradient Descent	MC-Gradient Descent
Stable Convergence	258 cases	278 cases
Better Convergence	17 cases	124 cases
Convergence to same point	74 cases	
T-statistics	3.0469	
P-value	0.0026	

Momentum:

The Momentum Gradient Descent method underwent testing in 405 scenarios. In these tests, the Multiplicative Momentum Gradient Descent approach maintained convergence in 269 instances, while the Classical Momentum method did so in 263 cases. Within the subset where both methods achieved convergence, the multiplicative variant outperformed the classical one in 118 cases. In contrast, the classical approach yielded superior outcomes in 19 cases, and both methods reached identical solutions in 73 cases. The significance of the performance differential between the two methods was confirmed through statistical analysis, which yielded t-statistics and p-values of 2.9701 and 0.0033, respectively.

Table 3. *Comparison of Momentum and MC-Momentum*

	Momentum	MC-Momentum
Stable Convergence	263 cases	269 cases
Better Convergence	19 cases	118 cases
Convergence to same point	73 cases	
T-statistics	2.9701	
P-value	0.0033	

NAG:

The NAG method was evaluated across 405 scenarios. The Multiplicative NAG approach remained convergent in 267 of these cases, whereas the Classical NAG method did so in 255 instances. In situations where both approaches achieved convergence, the multiplicative variant demonstrated superior performance in 118 cases. On the other hand, the classical version outperformed the multiplicative one in 18 instances, and both methods arrived at identical outcomes in 73 cases. Statistical analysis further underscored these results, revealing t-statistics and p-values of 3.0475 and 0.0026, respectively, thereby highlighting the significant differences in performance between the two approaches.

Table 4. *Comparison of NAG and MC-NAG*

	NAG	MC-NAG
Stable Convergence	255 cases	267 cases
Better Convergence	18 cases	118 cases
Convergence to same point	73 cases	
T-statistics	3.0475	
P-value	0.0026	

AdaGrad:

The AdaGrad algorithm was subjected to examination in 405 distinct scenarios. The Multiplicative AdaGrad variant remained convergent in 257 of these cases, mirroring the Classical AdaGrad's convergence in an equal number of instances. Among the scenarios where convergence was achieved by both methods, the alternative multiplicative AdaGrad yielded more favorable outcomes in 115 cases, while the classical AdaGrad delivered superior performance in 108 cases, and both methods reached identical solutions in 33 cases. A thorough statistical analysis was conducted to assess these findings, yielding t-statistics and p-values of 4.0893 and $5.8033*10^{-5}$, respectively, which underscore the significant performance differences between the two methodologies.

Table 5. *Comparison of AdaGrad and MC-AdaGrad*

	AdaGrad	MC-AdaGrad
Stable Convergence	257 cases	257 cases
Better Convergence	108 cases	115 cases
Convergence to same point	33 cases	
T-statistics	4.0893	
P-value	$5.8033*10^{-5}$	

RMSProp:

The RMSProp optimization technique was rigorously tested across 405 scenarios. In this series of tests, the Multiplicative RMSProp variant maintained convergence in 262 instances, closely followed by the Classical RMSProp, which did so in 261 instances. Within the subset where both versions converged, the multiplicative adaptation of RMSProp achieved more favorable outcomes in 119 cases. Conversely, the classical version of RMSProp outperformed the multiplicative variant in 110 cases, with both methods arriving at identical outcomes in 30 cases. Comprehensive statistical analysis was performed to evaluate these outcomes, resulting in t-statistics and p-values of 4.2504 and $2.9842*10^{-5}$, respectively, thereby highlighting the significant performance distinctions between the two RMSProp adaptations.

Table 6. *Comparison of RMSProp and MC- RMSProp*

	RMSProp	MC-RMSProp
Stable Convergence	261 cases	262 cases
Better Convergence	110 cases	119 cases
Convergence to same point	30 cases	
T-statistics	4.2504	
P-value	$2.9842*10^{-5}$	

To discern the strengths and limitations of the algorithms within various constrained environments, a meticulous evaluation was conducted, focusing on the impact of different constraint types on algorithm performance.

Bounded Constraints:

In the context of composite functions with bounded constraints, the alternative methods demonstrated superior performance in 379 instances, while classical methods excelled in 199 cases. Statistical analysis underscored these findings, yielding a t-statistic of 5.3609 and a highly significant p-value of $1.19*10^{-7}$.

Table 7. *Comparison at Bounded Constraints*

	Classical Methods	MC-Methods
Better Convergence	119 cases	379 cases
Convergence to same point	0 cases	
T-statistics	5.3609	
P-value	$1.19*10^{-7}$	

Quadratic Constraints:

When analyzing composite functions subject to quadratic constraints, alternative methods surpassed classical methods in 62 instances, whereas classical methods were superior in 45 cases. Both methods converged to the same value in 141 scenarios. Statistical validation of these outcomes was provided by a t-statistic of 3.5332 and a p-value of 0.0004, indicating significant performance differences.

Table 8. *Comparison at Quadratic Constraints*

	Classical Methods	MC-Methods
Better Convergence	45 cases	62 cases
Convergence to same point	141 cases	
T-statistics	3.5332	
P-value	0.0004	

Other type of Constraints:

In scenarios involving composite functions with other types of constraints, the alternative methods outshined classical approaches in 125 instances, while the latter prevailed in 25 cases. Additionally, both methodologies achieved identical outcomes in 142 instances. The distinction in performance between the two methods was statistically significant, as evidenced by a t-statistic of 3.3093 and a p-value of 0.0010.

Table 9. *Comparison at Other types of Constraints*

	Classical Methods	MC-Methods
Better Convergence	25 cases	125 cases
Convergence to same point	142 cases	
T-statistics	3.3093	
P-value	0.0010	

Unconstrained:

Within the realm of unconstrained composite functions, alternative methodologies demonstrated superior performance in 26 instances, in contrast to classical approaches, which excelled in 3 cases. The significant disparity in efficacy between the two methodologies is corroborated by a t-statistic of 3.9010 and a p-value of 0.0005, indicating a statistically significant difference.

Table 10. *Comparison at Unconstrained functions*

	Classical Methods	MC-Methods
Better Convergence	3 cases	26 cases
Convergence to same point	0 cases	
T-statistics	3.9010	
P-value	0.0005	

6. Discussion

This study has attempted a comprehensive review of classical and multiplicative analysis-based optimization algorithms across a diverse set of scenarios and constraints, using exponential-quadratic-logarithmic composite functions. Findings from this research provide insight into the relative strengths and potential limitations of each approach under varying conditions.

Algorithm Performance Across Scenarios:

The classical gradient descent algorithm and its variations, including NAG, Momentum, AdaGrad, and RMSProp, were analyzed along with their multiplicative counterparts. In our tests across five scenarios, we consistently saw that multiplicative methods were especially adaptable and responsive, especially in tests with many iterations and different learning rates. This adaptability highlights how multiplicative computing can improve how algorithms combine and perform, especially in complex optimization cases.

Convergence and Superiority:

An important aspect of our findings concerns the convergence rates of classical and multiplicative methods. While both approaches exhibited commendable convergence abilities, multiplicative methods frequently outperformed their classical counterparts, providing smaller values in most cases where convergence was achieved. This superiority was seen most in the Bounded constraints case, and while alternative methods gave better results for 379 cases, only 199 methods gave better results for classical methods. The Unconstrained case was the second most successful, as the number of scenarios was relatively small compared to the Bounded scenario, Classical algorithms only succeeded in converging in %10 of the cases. While the Classical algorithms had difficulty converging, the alternative methods were successful in more scenarios and performed better in 90 percent of comparable situations. Regarding other types of constraints, alternative methods outperformed classical methods in 125 cases, while classical methods showed superior performance in 25 cases; this indicated a significant advantage for alternative approaches in these specific conditions.

Statistical Significance:

Statistical analysis, including t-statistics and p-values, played an important role in validating the performance differences observed between classical and multiplicative methods. The significant p-values obtained across various algorithms and types of constraints strengthen the robustness of approaches based on multiplicative analysis and shows a significant improvement over traditional methods in certain optimization scenarios.

Constraint-Specific Observations:

Evaluation based on types of constraints further illuminated the strengths and weaknesses of the algorithms. There were examples of the superiority of both classical and alternative methods, especially in scenario with quadratic; This suggested that the choice of optimization method may depend on the specific nature of the constraint. However, in unconstrained environments, alternative methods have demonstrated their versatility and distinct advantage in broader optimization applications.

Multiplicative Methods' Superiority:

The superior performance of multiplicative methods in certain scenarios can be attributed to their natural fit with the exponential growth or decay models common to the exponential-quadratic-logarithmic composite functions tested. Thanks to this adaptability, it enables effective research in an optimization environment where classical algorithms may have difficulty in unconstrained situations. Future studies can create new models by blending the best features of multiplicative analysis-based methods with the features of traditional methods.

Limitations:

A limitation of this work is the computational efficiency and RAM usage of methods based on multiplicative calculus. This should be taken into consideration when using multiplicative calculus-based optimization algorithms in large-scale projects and algorithms that require high processing speed and RAM capacity, such as deep learning.

Implications and Future Directions:

The analyzes obtained from this study have the potential to provide advantages in practical application in various fields beyond academic interest. It is thought that the proven superiority of methods based on multiplicative analysis in certain scenarios will direct scientists to further research on optimization and applications in other fields. Future studies may focus on improving these methods and analyzing the performance of real-world problems.

This study opens a different innovative way to complex optimization problems by presenting a comparative analysis on optimization based on classical and multiplicative analysis. This performance increase obtained on the composite function with multiplicative calculus also paves the way for the evaluation of other alternative calculus methods.

7. Conclusion

Our investigation into the application of multiplicative calculus within optimization algorithms has revealed a promising avenue for addressing complex optimization problems, especially in positive domains. The multiplicative methods exhibited exceptional performance in unconstrained and bounded scenarios, outperforming classical algorithms in a significant majority of cases. These findings underscore the potential of multiplicative calculus-based methods for enhancing optimization techniques. However, the computational efficiency and RAM usage of these methods pose challenges that warrant further exploration.

Zou and colleagues examined the convergence properties of adaptive methods such as RMSProp and Adam, finding them to be effective in a variety of scenarios [3]. Our study supports these results and shows that multiplicative adaptations not only preserve but also strengthen these traits. In particular, our statistical analysis shows that multiplicative methods provide faster and more stable convergence in various optimization environments. Boyd and Vandenberghe’s work on convex optimization provides a solid foundation for understanding optimization in constrained environments [22]. Our research builds on these principles by applying multiplicative computation to gradient descent algorithms, demonstrating superior performance in constrained and unconstrained scenarios that Boyd and Vandenberghe describe as challenging for classical methods. Wilson's work addresses several limitations of classical gradient descent methods, including handling of positivity constraints, convergence issues, and performance in non-convex optimization [4]. Multiplicative methods inherently preserve positivity constraints and provide non-negative variables throughout the optimization process. Multiplicative methods show stronger convergence properties, increasing the probability of reaching optimal solutions, especially in convex scenarios. Additionally, Multiplicative based methods exhibited greater robustness to local minima, improving performance in complex optimization environments. The strengths of adaptive methods such as AdaGrad and RMSProp are emphasized by Ruder, faster convergence and stability are achieved by integrating the MC calculation in our work[1]. The non-linear structure of MC based methods enables faster convergence.

Future research could focus on developing hybrid models that combine the strengths of both classical and multiplicative approaches, potentially mitigating limitations and expanding the applicability of these advanced optimization strategies.

Conflict Of Interest

The Author reports no conflict of interest relevant to this article.

Research And Publication Ethics Statement

The author declares that this study complies with research and publication ethics.

References

[1] Ruder, S. (2016). An overview of gradient descent optimization algorithms. 1–14, [Online]. Available: http://arxiv.org/abs/1609.04747.

[2] Baldi, P. (1995). Gradient Descent Learning Algorithm Overview: A General Dynamical Systems Perspective. *IEEE Trans. Neural Networks*, 6(1), 182–195.

[3] Zou, F., Shen, L., Jie, Z., Zhang, W. and Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2019(1), 11119–11127.

[4] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *Adv. Neural Inf. Process. Syst.*, 2017, 4149–4159.

[5] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1), 145–151.

[6] Gould, N. I. M., Orban, D. and Toint, P. L. (2015). CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. *Comput. Optim. Appl.*, 60(3),545–557.

[7] Zhang, J. (2019). Gradient Descent based Optimization Algorithms for Deep Learning Models Training. [Online]. Available: http://arxiv.org/abs/1903.03614.

[8] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods.

USSR Comput. Math. Math. Phys., 4(5), 1–17.

[9] Duchi, J. C., Bartlett, P. L.and Wainwright, M. J. (2012). Randomized smoothing for (parallel) stochastic optimization. *Proc. IEEE Conf. Decis. Control*, 12, 5442–5444.

[10] Bashirov, A. E., Kurpinar, E. M. and Özyapici, A. (2008). Multiplicative calculus and its applications. *J. Math. Anal. Appl.*, 337(1), 36–48.

[11] Georgiev S. G. and Zennir, K. (2022). *Multiplicative Differential Calculus*, 1st ed., vol. 1, no. 1. Taylor & Francis.

[12] Gürefe Y. and Misirli, D. D. E.(2011). Product Calculi And Its Applications. *J. Phys. A Math. Theor.*, 44(8), 1–22.

[13] Uzer, A. (2010). Multiplicative type complex calculus as an alternative to the classical calculus. *Comput. Math. with Appl.*, 60(10), 2725–2737.

[14] Stanley, D. (1999). A multiplicative calculus. *Primus*, 9(4), 310–326.

[15] Özyapici A. and Misirli, A. P. D. E. (2009). Multiplicative Calculus And Its Aplications,” Ege University.

[16] Özyapıcı, A., Riza, M., Bilgehan, B. and Bashirov, A. E. (2014). On multiplicative and Volterra minimization methods. *Numer. Algorithms*, 67(3), 623–636.

[17] Filip, D., Piatecki, C. and Andrada Filip, D. (2014). A non-newtonian examination of the theory of exogenous economic growth. *Work. Pap.*, X(XX), 2014, [Online]. Available: https://hal.archives-ouvertes.fr/hal-00945781.

[18] Florack L.and Van Assen, H. (2012). Multiplicative calculus in biomedical image analysis. *J. Math. Imaging Vis.*, 42(1), 64–75.

[19] Riza M.and Aktöre, H. (2015). The Runge-Kutta method in geometric multiplicative calculus. *LMS J. Comput. Math.*, 18(1), 539–554.

[20] Riza M. and Eminağa, B. (2014). Bigeometric Calculus and Runge Kutta Method. 1–19, [Online]. Available: http://arxiv.org/abs/1402.2877.

[21] Cubillos, M. (2018). Modelling wave propagation without sampling restrictions using the multiplicative calculus I: Theoretical considerations. 1–18, [Online]. Available: http://arxiv.org/abs/1801.03402.

[22] Stephen Boyd Lieven Vandenberghe, (2013). *Convex Optimization 中文影印*. Cambridge University Press.