

École polytechnique de Louvain

Tight Convergence Analysis of Exact and Inexact Gradient Methods with Constant and Silver Schedules

Author: **Pierre VERNIMMEN**
Supervisor: **François GLINEUR**
Readers: **Geovani GRAPIGLIA, Julien HENDRICKX**
Academic year 2023–2024
Master [120] in Mathematical Engineering

Abstract

This work explores optimization and first-order algorithms, beginning with an introduction to these concepts in Section 1, followed by an explanation of the recently developed Performance Estimation framework. This framework treats the worst-case convergence analysis of an algorithm on a given class of functions as an optimization problem itself. Next, we discuss the concept of *relative* inexact gradient, where first-order optimization algorithms are applied using *inexact* gradient information. Taking account of imprecision makes perfect sense in today's world. In many of today's applications, we want to optimise processes and all we have is inexact information, for example in machine learning.

In Section 2, we theoretically analyze constant step size first-order optimization algorithms, considering both exact and inexact gradients. We derive theoretical bounds on the rate of convergence σ_n (for example with the setup $\|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f_*)$, where $\{x_k\}_{k \geq 0}$ is the sequence of iterates generated by the optimization method). We investigate the effects of inexactness on these methods, compare our findings with numerical results, and propose conjectures on the best step sizes to use based on the number of iterations and the level of inexactness.

Section 3 introduces and experimentally analyzes a new step size schedule, the *silver step size schedule*, designed to accelerate the gradient method without requiring memory storage. Using the Performance Estimation methodology, we obtain tight bounds not presented in the original paper. We also analyze this method when dealing with inexact gradient and develop new algorithms to tackle the challenges posed by inexactness.

Finally, Section 4 compares the different methods from both theoretical and practical perspectives, including a linear regression experiment. This last section also includes a discussion of the optimal method to choose, depending on the properties of the problem being addressed.

Keywords Convex optimization · Gradient descent · Gradient with relative inexactness · Worst-case complexity analysis · Performance estimation problem · Silver step size schedule

Acknowledgments

I would like to sincerely thank Professor François Glineur for his exceptional advice and unfailing support throughout my thesis. His availability, his always prompt responses and his determination to explain complex concepts were priceless. His mentorship and dedication have played an essential role in the good realization of this work.

I would also like to thank Professors Geovani Grapiglia and Julien Hendrickx for agreeing to be part of the jury for my thesis. Their willingness to put their expertise and time at the disposal of the assessment of my work is very much appreciated.

Finally, I would like to thank my family and friends for their support and encouragement throughout my academic journey. Their love, understanding, and encouragement have always been a source of strength and motivation. I'm deeply grateful to them for believing in me and supporting me every step of the way.

Contents

Abstract	1
Acknowledgments	2
Contents	3
1 Introduction	4
1.1 Optimization and First-Order Algorithms	4
1.1.1 Introduction to Optimization	4
1.1.2 Non-Linear Optimization and First-Order Algorithms	5
1.1.3 Key Function Properties in Optimization	6
1.1.4 The Concept of Oracle in Optimization	9
1.2 Performance Estimation	10
1.2.1 Classical Performance Estimation	10
1.2.2 Automated Performance Estimation	13
1.3 Inexact gradient	17
2 Constant Step Size Gradient Method	20
2.1 Exact Gradient	20
2.1.1 Single Step of Exact Gradient	20
2.1.2 Multiple Steps of Exact Gradient	25
2.2 Inexact Gradient	31
2.2.1 Single Step of Inexact Gradient	31
2.2.2 Multiple Steps of Inexact Gradient	36
3 Silver Step Schedule	43
3.1 Introduction to Silver Step Schedule	43
3.2 Convergence Rate Analysis and Comparison with the Classical Gradient Descent	46
3.2.1 Comparison with Classical Gradient Descent	46
3.2.2 Improving the Convergence Rate from Altschuler and Parrilo	48
3.3 Inexact Gradient and Modified Silver Schedule	50
3.3.1 Introduction to Modified Silver Schedule	51
3.3.2 Modified Silver Schedule with Inexact Gradient	53
3.4 Conclusion on the Different Silver Schedules	58
4 Comparison of Algorithms	60
4.1 Theoretical Comparison	60
4.2 Experiment Analysis and Application to Linear Regression	65
5 Conclusion	73
Bibliography	75

1 Introduction

1.1 Optimization and First-Order Algorithms

1.1.1 Introduction to Optimization

Optimization, is a powerful field in mathematics playing a fundamental role in decision-making, problem-solving, and the advancement of various theoretical and applied disciplines. The goal of this speciality is to identify the optimal solution to a problem within a set of feasible alternatives.

In the context of optimization, a problem is typically defined by an objective function, which is either to be maximized or minimized, subject to a set of constraints. The objective function represents the quantity that is to be optimized, such as profit, efficiency, or utility. Constraints, however, reflect the limitations or requirements imposed on the decision variables. The objective is to identify the values of the decision variables resulting in the optimal outcome in accordance with the defined criteria.

Optimization problems are common in many fields, including engineering, economics, finance, operations research, machine learning, and many others. They provide a systematic framework for addressing complex decision-making challenges, enabling the efficient allocation of resources, the design of optimal systems, and the improvement of overall performance.

In this work, we will use the same mathematical terminology as in [Nes03] to formulate our general optimization problem. We will then be able to work with several variants of this general problem.

Let $x = (x^{(1)}, \dots, x^{(d)}) \in \mathbb{R}^d$ be the d -dimensional vector of variables, $S \subseteq \mathbb{R}^d$ be a constraint set and $f(x), f_1(x), \dots, f_m(x)$ be real-valued functions of x . The general optimization problem can thus be written as¹:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_j(x) \ \& \ 0, j = 1, \dots, m \\ & x \in S, \end{aligned} \tag{1.1}$$

where $\&$ can either be $\leq$, $\geq$ or $=$.

¹In this work, d will represent the dimension and n the number of iterations of the methods.

The function $f(x)$ is called the *objective function* of the problem, while $Q = \{x \in S \mid f_j(x) \leq 0, j = 1, \dots, m\}$ is called its *feasible set*. If $Q \equiv \mathbb{R}^d$, then the problem is said to be unconstrained. If all $f_j(x)$ are differentiable, the problem is smooth.

The global optimal solution $x_* \in Q$ to problem (1.1) is such that $f(x_*) \leq f(x) \forall x \in Q$. We should note that the minimization in (1.1) is a convention, but it's equivalent to consider the maximization of $-f(x)$.

1.1.2 Non-Linear Optimization and First-Order Algorithms

All along this work on optimization, we will mainly talk about *nonlinear* optimization. This is a general scope where we assume that functions $f_i(x)$ are nonlinear, without loss of generality. In some cases, some of them may be linear, but we won't use this assumption and will treat them as if they were not.

Without loss of generality, our typical non-linear optimization problems will be unconstrained, meaning that our goal is to minimize a function $f(x)$ on the set $Q = \mathbb{R}^d$. A constrained optimization problem can be rewritten as an unconstrained one in several ways, but it won't be discussed in this work.

It has been proven in [Nes03] that optimization problems are in general mathematically impossible to solve exactly (this strong claim comes from the limitations of algorithms for high-dimensional problems, where the number of calls of the Oracle, which will be defined in section 1.1.4, grows exponentially with the dimension of the problem), and we thus have to rely on some algorithms to find some ϵ -approximate solution.

Definition 1.1 (ϵ -solution). *Let x_* be the optimal solution, and x be a candidate solution. Solution x is said to be an ϵ -solution with respect to some metric $M_f(\cdot)$ if $M_f(x) \leq \epsilon$. Thus, the lower the metric, the better the solution, and x_* is the exact solution if and only if $M_f(x_*) = 0$. The metric $M_f(\cdot)$ can be the difference between function values $f(x) - f(x_*)$, the norm of its gradient $\|\nabla f(x)\|^2$, etc*

It has been shown that uniform grid method (which consists in evaluating the function at different places on a grid) is bad numerically speaking (this concept of numerical performance will be discussed in Section 1.2) and that we should use more information, if available, to find an ϵ -solution to our optimization problem. This extra information on the function is, for example, its gradient $\nabla f(x)$. Remembering that the gradient gives the direction of steepest slope of a function and that we want to minimize the objective function, the following algorithm comes naturally to mind :

Algorithm 1 Gradient Descent with Stopping Criteria according to Metric $M_f(\cdot)$ and Step Size α

```

1: Set arbitrary  $\mathbf{x}$ 
2: while  $M_f(\mathbf{x}) > \epsilon$  do
3:    $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$ 
4: end while
5: Return  $\mathbf{x}$ .

```

This algorithm starts with an arbitrary point and then minimizes the function value by going down in the opposite direction of the gradient by some factor α , called the step size of the method, until an ϵ -solution is found. This algorithm is called the *Gradient Descent* and is the most typical example of a first-order optimization method.

Definition 1.2 (First-order optimization algorithm). *An optimization algorithm that only uses first order information on the function $f(x)$ (i.e, function values and gradient $\nabla f(\cdot)$) is said to be first-order.*

1.1.3 Key Function Properties in Optimization

Some crucial mathematical properties of functions need to be introduced before going any further into the world of optimization. Those properties will allow us to design better optimization algorithms on some particular classes of functions.

The first three properties will be L -smoothness, convexity and μ -strong convexity, which are the most important to design appropriate optimization algorithms to minimize the objective function.

Definition 1.3 (L -smooth function). *A differentiable function $f(\cdot)$ is said to be L -smooth if $\forall x \in \mathbb{R}^d$ the following inequality holds:*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2, \quad \forall y \in \mathbb{R}^d,$$

where $\langle \cdot, \cdot \rangle$ is the usual scalar product.

Graphically, it means that the function $f(\cdot)$ is upper bounded by some quadratic function of parameter $L \forall x \in \mathbb{R}^d$. This property can be visualized on an example, along with the other function properties defined in this section, on Figure 1.1.

Definition 1.4 (Convex function). *A differentiable function $f(\cdot)$ is said to be convex if and only if $\forall x \in \mathbb{R}^d$ the following inequality holds:*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle, \quad \forall y \in \mathbb{R}^d.$$

Graphically, it means that the graph of the function $f(\cdot)$ lies above all of its tangents. It can also be seen on Figure 1.1.

Definition 1.5 (μ -strongly convex function). *For a given $\mu > 0$, a differentiable function $f(\cdot)$ is said to be μ -strongly convex if and only if $\forall x \in \mathbb{R}^d$ the following inequality holds:*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2, \quad \forall y \in \mathbb{R}^d.$$

In this case, this property can be seen graphically in the sense that the function is lower bounded by some quadratic function of parameter $\mu \forall x \in \mathbb{R}^d$. This phenomenon can also be seen in Figure 1.1. We can notice that a μ -strongly convex function is necessarily convex, but that the converse is not always true. A simple way to find both constants L and μ exists if the function is twice differentiable. This is shown hereunder.

Theorem 1.1 (Condition on the Hessian for L -smoothness). *If a function $f(x)$ is twice differentiable and the following inequality holds:*

$$\nabla^2 f(x) \preceq LI,$$

then $f(x)$ is L -smooth.

Proof. Considering the Taylor expansion of $f(y)$ around x up to the second order:

$$f(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} (y - x)^T \nabla^2 f(\xi) (y - x), \quad (1.2)$$

where ξ is some point between x and y .

Using the condition $\nabla^2 f(x) - LI \preceq 0$, we have:

$$\frac{1}{2} (y - x)^T \nabla^2 f(x) (y - x) - \frac{1}{2} L (y - x)^T (y - x) \leq 0 \quad (1.3)$$

Using (1.2) and (1.3), we get that:

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2,$$

which is the definition of an L -smooth function. $\square$

Theorem 1.2 (Condition on the Hessian for strong convexity). *If a function $f(x)$ is twice differentiable and the following inequality holds:*

$$\mu I \preceq \nabla^2 f(x)$$

then $f(x)$ is μ -strongly convex.

Proof. The proof is similar to the one above. $\square$

Example 1.1 We will now show graphically the concepts defined above on the 1-dimensional example $f(x) = x^2 + \sqrt{x^2 + 1}$. Its Hessian, which is simply its second derivative, as the function is 1-dimensional, is $f''(x) = 2 + \frac{1}{(x^2+1)^{3/2}}$. Using Theorems 1.1 and 1.2, we find that $f(x)$ is L -smooth with $L = 3$ and μ -strongly convex with $\mu = 2$ (because $0 \leq \frac{1}{(x^2+1)^{3/2}} \leq 1, \forall x$). We can indeed graphically see on Figure 1.1 that $f(x)$ is upper bounded by a quadratic function with parameter $L = 3$ and lower bounded by a quadratic function with parameter $\mu = 2$. As the function is μ -strongly convex, it is also convex and thus lower bounded by a linear function. We can see this graphically on the plot below.

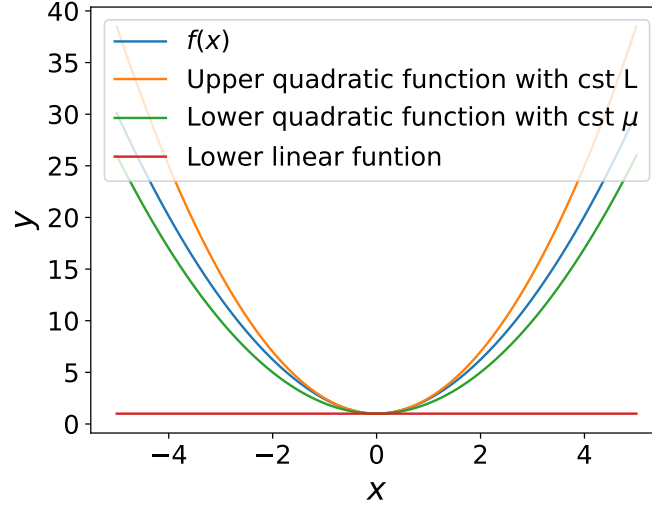


Figure 1.1: Visual representation of the L -smooth and μ -strongly convex function $f(x) = x^2 + \sqrt{x^2 + 1}$.

The ratio $\kappa = \frac{L}{\mu}$ can be interpreted as a condition number of $f(\cdot)$, generalizing the condition number of a linear system $Ax = b$. This quantity will play a prominent role in the convergence analysis (introduced in Section 1.2) of optimization algorithms for strongly convex functions.

To end this section properly, it's worth adding the following definition of a concept closely related to the concept of L -smoothness.

Definition 1.6 (Function with Lipschitz continuous gradient). *A differentiable function $f(\cdot)$ is said to have a Lipschitz continuous gradient with constant L if and only if $\forall x, y \in \mathbb{R}^d$ the following inequality holds:*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

If the function $f(\cdot)$ is also convex, then the concepts of L -smooth function and function with Lipschitz continuous gradient are equivalent. However, if the function is non-convex, those properties are not necessarily equivalent. In this work, the non-convex framework won't be covered and the concepts of L -smooth functions and functions with Lipschitz-continuous gradient can be seen as equivalent.

1.1.4 The Concept of Oracle in Optimization

For an optimization problem stated in the standard form as referenced in (1.1), there is still a question about how the objective and constraint functions are defined.

We could have access to their analytical forms, but it's not always possible, depending on the application in which the optimization algorithm is used ([BV04]). That's why, in general, the concept of Oracle is used.

Definition 1.7 (Oracle $\mathcal{O}_f$). *In optimization, an Oracle $\mathcal{O}_f$ is a computational entity or subroutine that provides information about the objective function or its derivatives at specific points in the domain.*

In addition to providing information about the objective function and its derivatives, Oracles offer several advantages in optimization:

- **Flexibility in Function Representation:** Oracles allow for the representation of objective functions and their derivatives in a flexible manner. This flexibility enables optimization algorithms to work with functions that may not have explicit analytical forms but can be evaluated or approximated at specific points in the domain.
- **Black-Box Optimization:** Oracles facilitate black-box optimization, where the internal structure of the objective function is unknown or too complex to be explicitly defined. Optimization algorithms can interact with the Oracle without needing to know the details of how the function is implemented.

In general, in addition to the oracle, we are also given additional information on the function, such as differentiability or convexity. Notice that we can easily go from the analytic form to an oracle form, while the opposite is not true. That's why optimization algorithms are often based on this oracle concept.

1.2 Performance Estimation

1.2.1 Classical Performance Estimation

For now, we've seen that optimization problems are impossible to be solved exactly and that we have to rely on algorithms to find an ϵ -approximate solution to the real problem. Those optimization algorithms will play a crucial role in finding those approximate solutions. It is therefore very important to understand how they behave and converge (or not, if they are bad algorithms) towards an approximate solution. In addition to the convergence towards an ϵ -approximate solution, the most important aspect of this algorithmic analysis is the (approximate) number of iterations required to converge towards this ϵ -approximate solution.

Definition 1.8 (Performance of algorithm $\mathcal{A}$ on problem $\mathcal{P}$). *The performance of algorithm $\mathcal{A}$ on problem $\mathcal{P}$ is the total amount of computational efforts required by algorithm $\mathcal{A}$ to find an ϵ -approximate solution to problem $\mathcal{P}$.*

However, it would not be smart to compute the exact number of iterations needed on a defined problem. We will in fact analyze the performance on a *class* of problems $\mathcal{F}$. A class of problems may for example be: the class of convex and L -smooth functions. We will thus analyze the *worst-case complexity* of a algorithm $\mathcal{A}$ on the problem class $\mathcal{F}$.

Definition 1.9 (Performance of algorithm $\mathcal{A}$ on class of functions $\mathcal{F}$). *The performance of algorithm $\mathcal{A}$ on class of functions $\mathcal{F}$ is defined as the performance of algorithm $\mathcal{A}$ on the worst-case problem $\mathcal{P}$ within the class $\mathcal{F}$.*

This *worst-case complexity* analysis could also be extended to other complexity analysis, such as a *average-case complexity*, but this won't be covered in this work.

Generally, to find the performance of an optimization algorithm, inequalities are combined to find its worst case complexity.

Before going into an example, let's give a name to the most common first-order optimization algorithm.

Definition 1.10 (Classical gradient descent). *The classical gradient descent is a gradient descent (such as Algorithm 1) with constant step size $\alpha = \frac{1}{L}$, L being the L -smooth constant of the function.*

Example 1.2 Let's now consider the classical gradient descent and the metric $M_f(x) = \|\nabla f(\cdot)\|$. We will first show that, applying this algorithm, the function value will always decrease and then prove the number of iterations needed to reach an ϵ -approximate solution. We assume that $f(\cdot)$ is L -smooth, but we assume nothing about its convexity.

Lemma 1.1 (Descent Lemma). *Suppose that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is L -smooth, and let $\{x_k\}_{k \geq 0}$ be a sequence generated by the classical gradient descent method applied to $f(\cdot)$. Then,*

$$f(x_k) - f(x_{k+1}) \geq \frac{1}{2L} \|\nabla f(x_k)\|^2 \quad \forall k \geq 0.$$

Proof. Using the definition of an L -smooth function (Definition 1.3) and the classical

gradient descent linking the iterates together by the relation $x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k)$, we get:

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) + \langle \nabla f(x_k), x_{k+1} - x_k \rangle + \frac{L}{2} \|x_{k+1} - x_k\|^2 \\ &= f(x_k) + \langle \nabla f(x_k), -\frac{1}{L}\nabla f(x_k) \rangle + \frac{L}{2} \left\| -\frac{1}{L}\nabla f(x_k) \right\|^2 \\ &= f(x_k) - \frac{1}{L} \|\nabla f(x_k)\|^2 + \frac{1}{2L} \|\nabla f(x_k)\|^2 \\ &= f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2, \end{aligned}$$

which is the desired result. $\square$

Theorem 1.3 (Number of iterations to get an ϵ -approximate solution for classical gradient descent on L-smooth function). *Suppose that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is L-smooth, bounded from below by f_{low} , and let $\{x_k\}_{k \geq 0}$ be a sequence generated by the classical gradient descent applied to $f(\cdot)$. Given $\epsilon > 0$, if T is the first iteration number such that $\|\nabla f(x_T)\| \leq \epsilon$, then*

$$T < 2L(f(x_0) - f_{low})\epsilon^{-2}.$$

Proof. By the definition of T , we have:

$$\|\nabla f(x_k)\| > \epsilon, \quad \forall k \in \{0, \dots, T-1\}$$

From Lemma 1.1, it follows that:

$$f(x_k) - f(x_{k+1}) \geq \frac{1}{2L} \|\nabla f(x_k)\|^2 > \frac{1}{2L} \epsilon^2 \quad k = 0, \dots, T-1 \quad (1.4)$$

Summing up inequalities in (1.4) and using the lower bound f_{low} , we get:

$$\begin{aligned} f(x_0) - f_{low} &\geq f(x_0) - f(x_T) \\ &= \sum_{k=0}^{T-1} f(x_k) - f(x_{k+1}) \\ &> \sum_{k=0}^{T-1} \frac{1}{2L} \epsilon^2 \\ &= T \frac{1}{2L} \epsilon^2, \end{aligned}$$

and we obtain the desired result. $\square$

With this example coming from [GG23], we can see that finding the (approximate) number of iterations of a given algorithm on a given function class is already a bit long using one of the simplest possible algorithms on a simple function class. We thus expect that the proofs are in general much harder than the one from example 1.2. That is one of the reasons of the creation of the automated method that will be studied in Section 1.2.2.

1.2.2 Automated Performance Estimation

The Automated Performance Estimation methodology, called *Performance Estimation* methodology in the rest of this document, introduced in [DT14] allows to compute exact worst-case performance guarantees for a very large class of first-order optimization algorithms using automated methods. The approach relies on the observation that by definition, the worst-case behaviour of a first-order black-box optimization algorithm is by itself an optimization problem which consists of finding the maximal absolute inaccuracy over all possible inputs to the algorithm. Thus, with x_n being the output of the algorithm $\mathcal{A}$ after making n calls to the oracle $\mathcal{O}_f$, we look at the solution of the following Performance Estimation Problem (PEP) for the metric $M_f(\cdot) = f(x_n) - f(x_*)$ [Tay17]²:

$$\begin{aligned} \max_{f, x_0, \dots, x_n, x_*} \quad & f(x_n) - f(x_*) \\ \text{s.t.} \quad & f \in \mathcal{F} \\ & x_1, \dots, x_n \text{ are generated from } x_0 \text{ by algorithm } \mathcal{A} \text{ with } \mathcal{O}_f \quad (\text{PEP}) \\ & x_* \text{ is optimal for } f, \|x_* - x_0\|_2 \leq R \\ & x_0, \dots, x_n, x_* \in \mathbb{R}^d \end{aligned}$$

This means that solving the optimization problem (PEP) will give the tight worst-case performance of a first-order algorithm $\mathcal{A}$ on the class of functions $\mathcal{F}$ ³ (the performance is said to be tight as no better rate can be found). Parameter R is introduced to bound the distance between the initial point x_0 and the optimal solution x_* . Indeed, it is well-known that in most situations, performance of a first-order method cannot be sensibly assessed without such a constraint.

The hardest constraint to tackle is the functional constraint $f \in \mathcal{F}$, because the arising problem becomes an abstract hard optimization problem in infinite

²This problem can be extended to other standard metrics.

³This methodology could also be extended to higher-order optimization algorithms, but it won't be covered in this work.

dimensions. To overcome this difficulty, the approach taken in [DT14, Tay17] is to relax this constraint so that the problem can be reduced and formulated as an explicit finite-dimensional problem that can eventually be adequately analyzed. If we restrict ourselves to the set $\mathcal{F}$ of convex continuously differentiable functions with Lipschitz continuous gradient, the problem (PEP) can be reformulated as $(f_i = f(x_i), g_i = \nabla f(x_i))$ and $I = \{0, 1, \dots, N, *\}$:

$$\begin{aligned}
 \max_{\{x_i, f_i, g_i\}_{i \in I}} \quad & f_N - f_* \\
 \text{s.t.} \quad & \exists f \in \mathcal{F} \text{ s.t. } \mathcal{O}_f(x_i) = \{f_i, g_i\} \quad \forall i \in I \\
 & x_1, \dots, x_n \text{ are generated from } x_0 \text{ by algorithm } \mathcal{A} \text{ using } \{f_i, g_i\}_{i \in \{0, \dots, N-1\}} \\
 & g_* = 0, \|x_* - x_0\|_2 \leq R \\
 & x_0, \dots, x_n, x_* \in \mathbb{R}^d
 \end{aligned} \tag{1.5}$$

The crucial part of this reformulation is the first constraint, which can be understood as requiring that the set of triplets $\{x_i, f_i, g_i\}_{i \in I}$ can be interpolated by a function belonging to the class $\mathcal{F}$. This optimization problem is strictly equivalent to the original (PEP) in terms of optimal value, since every solution to (1.5) can be interpolated by a solution of (PEP) and, reciprocally, every solution of (PEP) can be discretized to provide a solution to (1.5) [THG17].

It has been shown in [THG17] that the problem (1.5) can be rewritten as a semi-definite program if the first-order algorithm $\mathcal{A}$ relies on *fixed step sizes*. It means that if the step sizes are fixed in advance (i.e. they don't change dynamically during the algorithmic procedure), then for any class of function $\mathcal{F}_{\mu, L}$ (the class of μ -strongly convex continuously differentiable functions with Lipschitz continuous gradient), with $0 \leq \mu \leq L \leq \infty$, the problem (PEP) can be solved as a semi-definite program. Furthermore, this problem is convex.

It's somehow intuitive that the problem of finding the worst-possible function within a given class for a given algorithm is equivalent to finding the best possible *proof* of convergence for the exact same problem. This equivalence is very common in optimization and those problems are called *strongly dual* to each other, i.e. the optimization problem on the proofs is strongly dual to the optimization problem (1.5) on the functions. This result has been proven in [THG17] using Slater point condition. It also means that every feasible point of the dual problem (the one on the proofs) is an upper bound on the optimal value of the (PEP) problem. This observation can be useful in the case where an upper bound is easier to find analytically than the exact (tight) bound.

As (PEP) can be obtained using a semi-definite program, numerical results can be solved easily using an SDP-solver. The full python package PEPit, that includes this numerical resolution, has been done in [GMG+22] and those numerical results can be a great help in the goal of finding analytical proofs of exact worst-case rate of

convergence. Furthermore, as the results from the SDP-solver give *exact* worst-case convergence rate and instance, if an analytical result exactly follows the numerical one given by the resolution of (PEP), then with very high numerical evidence, it can be assumed that the analytical result is tight (i.e. cannot be improved). In Section 2.1, those numerical results will be shown and the parallelism with analytical rates found with more classical mathematical proofs will be made.

General idea for proofs of rate of convergence. Knowing the class of functions $\mathcal{F}$, we can derive inequalities between function values. The class treated here and along this work is the class of L -smooth convex functions. The following relation, called interpolation inequalities, thus holds between any points (x_i, f_i) and (x_j, f_j) , $(i, j) \in \{0, 1, \dots, N, *\}$:

$$Q_{ij} : f_i \geq f_j + \langle g_j, x_i - x_j \rangle + \frac{1}{2L} \|g_j - g_i\|^2 \quad (1.6)$$

The general idea to prove the rate of convergence of a given method will be to use the relations between iterates (e.g. $x_1 = x_0 - \frac{1}{L}g_0$) and to combine the interpolation inequalities (1.6) with multipliers λ_{ij} associated with inequalities Q_{ij} to isolate the desired terms. This combination of inequalities and the desire to find the mathematically tight rate is equivalent to solving the equivalent SDP problem (as done numerically in the PEPit toolbox). Such an example of proof can be found in Section 2.1.

Three traditional setups will be mainly used in this paper to analyze the rate of convergence of different optimization methods:

$$\begin{aligned} \frac{1}{L} \|\nabla f(x_n)\|^2 &\leq \tau_n(f(x_0) - f(x_n)), \\ \frac{1}{L} \|\nabla f(x_n)\|^2 &\leq \tau_n(f(x_0) - f(x_*)), \\ \frac{1}{L} (f(x_n) - f(x_*)) &\leq \tau_n \|x_0 - x_*\|^2. \end{aligned}$$

One should note that evident relations exist between those different setups (the link between the two first ones is shown in Theorem 2.5, while the link with the third one is explained in the beginning of Section 2.1). Some theoretical results are also found in [KOPR23]. Throughout this work, it should be noted that the general behaviour of the convergence rates remains equivalent for all setups (e.g., if the rate of one setup increases by a certain factor between two iterates, then the rate of another setup will also increase by a similar factor).

All results in this paper will be proved for $L = 1$. Theorem 1.4 demonstrates why this can be done without loss of generality.

Theorem 1.4. *All results in this paper can be proved for $L = 1$ without loss of generality.*

Proof. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a L -smooth convex function. Suppose f is scaled by a factor $c > 0$, defining a new function $\tilde{f}(x) = cf(x)$.

The gradient of $\tilde{f}$ is given by:

$$\nabla \tilde{f}(x) = c \nabla f(x).$$

The smoothness constant of $\tilde{f}$ is cL . This follows because for all $x, y \in \mathbb{R}^d$,

$$\|\nabla \tilde{f}(x) - \nabla \tilde{f}(y)\| = c \|\nabla f(x) - \nabla f(y)\| \leq cL \|x - y\|,$$

as the function has a Lipschitz continuous gradient, because it is L -smooth and convex (see the discussion at the end of Section 1.1.3).

Using a normalized step size $\frac{h}{L}$, the update rule for the gradient descent algorithm becomes:

$$x_{k+1} = x_k - \frac{h}{L} \nabla f(x_k).$$

For the scaled function $\tilde{f}$, using the same normalized step size, we have:

$$x_{k+1} = x_k - \frac{h}{cL} \nabla \tilde{f}(x_k) = x_k - \frac{h}{cL} c \nabla f(x_k) = x_k - \frac{h}{L} \nabla f(x_k).$$

Thus, the iterates x_k remain the same.

Considering the first setup:

$$\frac{1}{L} \|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f(x_n)).$$

For the scaled function $\tilde{f}$, we have:

$$\frac{1}{cL} \|\nabla \tilde{f}(x_n)\|^2 = \frac{1}{cL} c^2 \|\nabla f(x_n)\|^2 = \frac{c}{L} \|\nabla f(x_n)\|^2.$$

The inequality becomes:

$$\frac{c}{L} \|\nabla f(x_n)\|^2 \leq \sigma_n c (f(x_0) - f(x_n)),$$

which simplifies to:

$$\frac{1}{L} \|\nabla f(x_n)\|^2 \leq \sigma_n (f(x_0) - f(x_n)).$$

The rate of convergence thus remains the same for any $c > 0$. Setting $c = L$ ensures that all results can be proven using $L = 1$. The results can then be generalized by a factor $\frac{1}{L}$.

The same reasoning can be applied to the two other setups, and it therefore shows that proving the results for $L = 1$ generalizes to any $L > 0$ without loss of generality. $\square$

1.3 Inexact gradient

The goal of Sections 2.2 and 3.3.2 of this work is to study how the convergence of the gradient method is affected when the gradient that is used at each iteration is computed inexactly. This situation is encountered in a large variety of situations, such as the use of floating point computations with limited accuracy, dependence on data that is only known approximately, or more generally the time-accuracy tradeoff that is almost always present when the objective function (and its gradient) is obtained through another iterative procedure (e.g. a simulation or another optimization process)⁴.

First-order methods relying on an inexact gradient have been studied before, using several distinct notions of inexactness. The approximate gradient introduced in [d'A08] is assumed to differ from the true gradient by some error whose norm is bounded. In [DGN14], another notion of inexact gradient is developed, based on the maximal error incurred by the corresponding quadratic upper bound. In these two cases, the approximation error is not directly related to the scale of the gradient, i.e. the error is measured in an *absolute* manner.

In this work, we focus on a *relative* notion of inexactness, where the norm of the difference between the true gradient and its approximate value is bounded by a certain fraction of the gradient norm ($\|d_k - \nabla f(x_k)\|^2 \leq \delta^2 \|\nabla f(x_k)\|^2$, with δ being the size of inexactness). This concept can be easily illustrated graphically. If we assume that the gradient $\nabla f(x_k) = (1, \dots, 1) \in \mathbb{R}^d$, then d_0 can be any vector starting at the origin and ending within a hyper-sphere of radius $\sqrt{d}\delta$, centered at

⁴Stochasticity may be viewed as another source of inexactness, but we limit the scope of this work to deterministic methods

$(1, 1, \dots, 1)$. A visual representation of this inequality is shown on Figure 1.2 for $d = 2$.

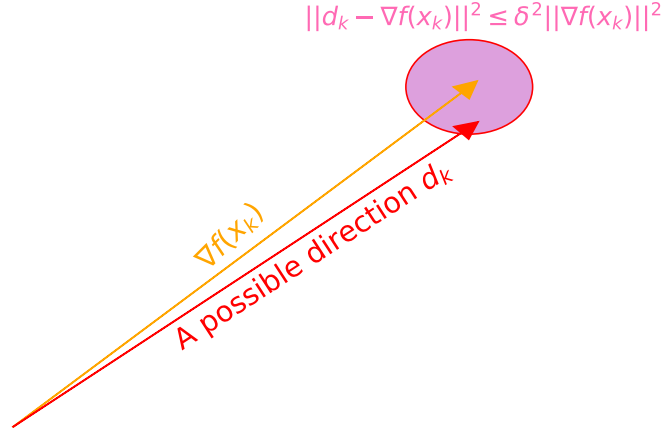


Figure 1.2: Visual representation of the inequality $\|d_k - \nabla f(x_k)\|^2 \leq \delta^2 \|\nabla f(x_k)\|^2$. d_k should end in a hyper-sphere of radius $\sqrt{d}\delta$ around the end of $\nabla f(x_k)$.

This notion was introduced in [DKGT20], and used to analyze the effect of inexactness when dealing with smooth, strongly convex functions. In this work, we extend this analysis to smooth convex functions in the absence of strong convexity. More specifically, we will provide a tight convergence analysis of the following inexact gradient method:

Algorithm 2 Inexact Gradient Descent with Constant (normalized) Step Size h

- 1: Given an L -smooth convex function f , a starting iterate x_0 , a constant step size h and a relative inexactness δ
- 2: $k \leftarrow 0$
- 3: **while** $k < \text{max_iter}$ **do**
- 4: Let d_k be an approximate gradient with δ relative inexactness, i.e.

$$\|d_k - \nabla f(x_k)\|^2 \leq \delta^2 \|\nabla f(x_k)\|^2 \quad (1.7)$$

- 5: Compute the next iterate as $x_{k+1} = x_k - \frac{h}{L} d_k$
 - 6: $k \leftarrow k + 1$
 - 7: **end while**
-

In the case of strongly convex functions, it was proved in [DKGT20, Theorem 5.3] that, when applied to an L -smooth and μ -strongly convex objective (with $\mu > 0$), the above gradient method with constant step size h and inexactness $\delta < \frac{2\mu}{L+\mu}$ converges linearly according to

$$\|\nabla f(x_1)\| \leq \left(1 - (1 - \delta)\frac{\mu}{L}\right) \|\nabla f(x_0)\|,$$

hence with

$$\|\nabla f(x_n)\| \leq \left(1 - (1 - \delta)\frac{\mu}{L}h\right)^n \|\nabla f(x_0)\|,$$

for $\delta \in [0; \frac{2\mu}{L+\mu}]$ and $h \in [0; \frac{2\mu-\delta(L+\mu)}{(1-\delta)\mu(L+\mu)}]$, and that this rate is tight. However, when the function becomes merely convex ($\mu \rightarrow 0$), this rate becomes effective only for $\delta = 0$ (as $\frac{2\mu}{L+\mu} \rightarrow 0$, when $\mu \rightarrow 0$), which is not very interesting considering inexact gradient! One must instead consider a different measure for the initial iterate x_0 , such as its accuracy. In the Section 2.2, we will prove results of the following type:

$$\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \tau_n(f(x_0) - f(x_*)) \Leftrightarrow \frac{1/L\|\nabla f(x_n)\|^2}{f(x_0) - f(x_*)} \leq \tau_n,$$

where constant τ_n describes the rate of convergence after n iterations. In the exact case ($\delta = 0$), it is well-known that the convergence of τ_n for smooth convex functions is of order $\mathcal{O}(\frac{1}{n})$. This result will be proven in Section 2.1 for any step size $h \in [0; \frac{3}{2}]$.

The concept of a gradient with δ relative inexactness can be added to (PEP) by adding (1.7), with quantities d_i s as additional variables. The Performance Estimation Problem with constant step size inexact gradient descent thus reads:

$$\begin{aligned} & \max_{\{x_i, f_i, g_i\}_{i \in I}} && f_N - f_* \\ & \text{s.t.} && \exists f \in \mathcal{F} \text{ s.t. } \mathcal{O}_f(x_i) = \{f_i, g_i\} \quad \forall i \in I \\ & && x_1, \dots, x_n \text{ are generated from } x_0 \text{ such that } x_{k+1} = x_k - \frac{h}{L}d_k \\ & && g_* = 0 \\ & && \|x_* - x_0\|^2 \leq R^2 \\ & && x_0, \dots, x_n, x_* \in \mathbb{R}^n \\ & && \|d_i - g_i\|^2 \leq \delta^2 \|g_i\|^2 \quad \forall i \in I \end{aligned}$$

The numerical resolution of this problem by the PEPit toolbox will help us find the analytic rates, before being able to mathematically prove them. The value $R = 1$ has been used for all analysis in this work.

2 Constant Step Size Gradient Method

In this section, we will analyze the constant step size gradient method. First, we will examine the method with an exact gradient, before turning our attention to the method with a relatively inexact gradient (see Section 1.3 for more information). In both cases, we will numerically derive bounds depending on the step size h before proving them rigorously using interpolation inequalities (1.6). Additionally, analysis will also be conducted for both exact and inexact frameworks.

Throughout this work, when we refer to the step size h , we mean the normalized step size $\frac{h}{L}$, where L is the L -smooth constant of the function. This normalization is used in the algorithm (2) for both the exact case ($\delta = 0$) and the inexact case ($\delta > 0$). As proved at the end of Section 1.2.2, this normalization is applied without loss of generality, as the analysis assumes $L = 1$.

It's worth noting that all theorems and proofs found in this section are new and cannot be found in any existing paper.

2.1 Exact Gradient

In both this section and the next one (Section 2.2), an initial analysis of one iteration of the algorithm will be performed before being expanded to the scenario involving multiple iterations.

2.1.1 Single Step of Exact Gradient

The first case to be analyzed in gradient descent is a single step of the algorithm. In this section, we will first derive numerical bounds on the rates of convergence using the PEPit toolbox. These bounds will then be rigorously analyzed through mathematical proofs using interpolation inequalities between the points (x_0, f_0) , (x_1, f_1) and (x_*, f_*) with associated gradients g_0 , g_1 and g_* .

The rates of convergence τ_1 and σ_1 used in this first analysis are defined such that $\frac{1}{L}(f(x_1) - f_*) \leq \tau_1 \|x_0 - x_*\|^2$ and $\frac{1}{L} \|\nabla f(x_1)\|^2 \leq \sigma_1 (f(x_0) - f_*)$. Clearly, the smaller those rates are, the more effective the method is, as it indicates a greater reduction in the function value or the gradient norm, which an optimization method aims at.

Figure 2.1 shows the numerical results provided by the PEPit toolbox derived

from solving the optimization problems (PEP) associated with each of those *different* gradient descents⁵.

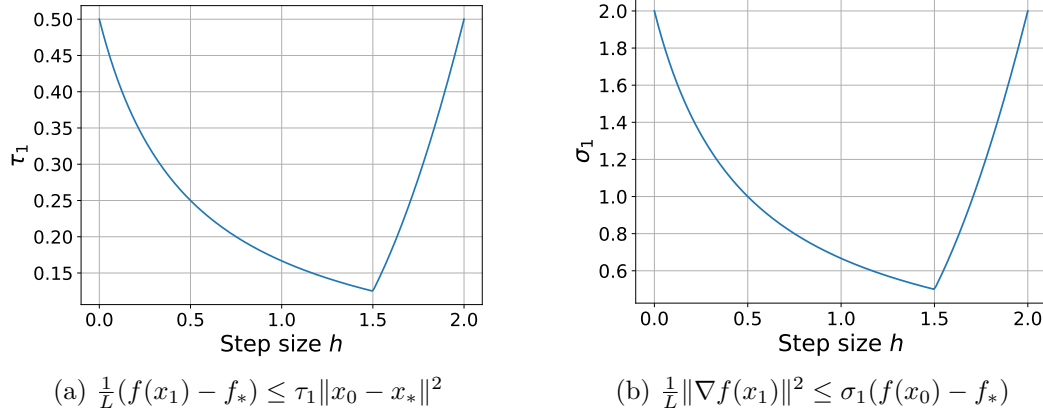


Figure 2.1: Numerical rates τ_1 and σ_1 as a function of the step size h for one step of the gradient method.

One can easily deduce from the left plot in Figure 2.1 that a step size longer than 2.0 will result in a worse (greater) function value. This can be demonstrated easily by taking the following test function:

$$f(x) = \frac{L}{2}x^2,$$

with $x_0 = 1$ (and thus $f_0 = \frac{L}{2}$ and $g_0 = L$). By applying one step of the gradient method, we get:

$$\begin{aligned} x_1 &= x_0 - \frac{h}{L}g_0 \\ &= 1 - \frac{h}{L}L \\ &= 1 - h, \end{aligned}$$

and thus:

$$\begin{aligned} f_1 &= \frac{L}{2}(1 - h)^2 \\ &= f_0(1 - h)^2 \\ &\geq f_0, \quad \text{when } h \geq 2. \end{aligned}$$

⁵They differ only in their step sizes.

This shows that applying one step of the gradient method on this particular function with a step size $h > 2$ results in a greater (worse, since we aim to minimize it) function value. This implies that the same will hold for the worst-case function.

In the same way, a step size $h > 2$ will also result in a greater (thus worse) norm of the gradient norm.

Furthermore, we can see an obvious link between the two plots above, which gives us some insight on the relationships between those two different setups. The relationship between those rates is simply a multiplicative factor 4 (i.e. $\sigma_n = 4\tau_n, \forall n$). The result won't be proven, but we can assume as true, because supported with strong numerical evidence. This multiplicative factor between the rates is always the same, even for several steps of constant (both exact and inexact, see Section 2.2) gradient descent. More theoretical results like this can be found in [KOPR23]. Writing down the above result neatly gives:

$$\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2 \iff \|\nabla f(x_n)\|^2 \leq 4\tau_n (f(x_0) - f(x_*)) \quad (2.1)$$

Additionally, the plots in Figure 2.1 allows us to deduce the analytical expression of the rate of convergence of one iteration of the gradient descent, depending on the step size h . This rate was thus initially identified through numerical results using the PEPit toolbox. Subsequently, it was rigorously proven through mathematical analysis. The left rate in (2.1) has been shown in [DT14], while the second currently doesn't exist in the literature. The current section of this works aims to state and prove the rate under the right setup in (2.1).

Theorem 2.1 (Rate of convergence of one iteration of gradient descent with step size $h \leq \frac{3}{2}$). *Suppose that $x_1 = x_0 - \frac{h}{L}\nabla f(x_0)$ with $h \in [0; \frac{3}{2}]$. If $f(\cdot)$ is an L -smooth convex function, then*

$$\frac{1}{L}\|\nabla f(x_1)\|^2 \leq \frac{2}{2h+1}(f(x_0) - f(x_*)).$$

Proof. The following three interpolation inequalities are guaranteed by smoothness and convexity (see equation (1.6)):

$$\begin{aligned} Q_{01} : f_0 &\geq f_1 + \langle g_1, x_0 - x_1 \rangle + \frac{1}{2L}\|g_1 - g_0\|_2^2 \\ Q_{10} : f_1 &\geq f_0 + \langle g_0, x_1 - x_0 \rangle + \frac{1}{2L}\|g_0 - g_1\|_2^2 \\ Q_{1*} : f_1 &\geq f_* + \langle g_*, x_1 - x_* \rangle + \frac{1}{2L}\|g_* - g_1\|_2^2. \end{aligned}$$

By noting the necessary first-order optimality condition $g_* = 0$, assuming that $L = 1$ (without loss of generality, as proved in Theorem 1.4), the relation $x_1 = x_0 - hg_0$, and combining the above inequalities with the corresponding multipliers $\lambda_{01} = 2$, $\lambda_{10} = 1$ and $\lambda_{1*} = 1$, we get:

$$f_0 \geq f_* + 2h\langle g_0, g_1 \rangle + \frac{3}{2}\|g_0 - g_1\|^2 - h\|g_0\|^2 + \frac{1}{2}\|g_1\|^2.$$

Using the relation $\|g_0 - g_1\|^2 = \|g_0\|^2 + \|g_1\|^2 - 2\langle g_0, g_1 \rangle$, we get:

$$\begin{aligned} f_0 &\geq f_* + (2h - 3)\langle g_0, g_1 \rangle + \left(\frac{3}{2} - h\right)\|g_0\|^2 + 2\|g_1\|^2 \\ &= f_* + (2h - 3)\langle g_0, g_1 \rangle + \left(\frac{3}{2} - h\right)\|g_0\|^2 + \left(\frac{3}{2} - h\right)\|g_1\|^2 + \left(\frac{1}{2} + h\right)\|g_1\|^2 \quad (2.2) \\ &= f_* + \left(\frac{3}{2} - h\right)\|g_1 - g_0\|^2 + \left(\frac{1}{2} + h\right)\|g_1\|^2, \end{aligned}$$

and as soon as $\frac{3}{2} - h \geq 0$, which is guaranteed by hypothesis, the second term in the right-hand side is non-negative, and this leads to:

$$f_0 \geq f_* + \left(\frac{1}{2} + h\right)\|g_1\|^2,$$

which can be rewritten under the following form:

$$\|g_1\|^2 \leq \frac{2}{2h + 1}(f_0 - f_*)$$

□

This bound can also be expressed in another (equivalent) way, which will be easier to handle to analyze several steps of gradient descent in Section 2.1.2.

Theorem 2.2. *Suppose that $x_1 = x_0 - \frac{h}{L}\nabla f(x_0)$ with $h \in [0; \frac{3}{2}]$. If $f(\cdot)$ is an L -smooth convex function, then*

$$\frac{1}{L}\|\nabla f(x_1)\|^2 \leq \frac{1}{h}(f(x_0) - f(x_1))$$

Proof. The proof is very similar to the one of Theorem 2.1, using multipliers $\lambda_{01} = 2$, $\lambda_{10} = 1$ and $\lambda_{1*} = 0$. □

We have thus shown an upper bound on the rate of convergence of the gradient method with constant step size $h \in [0; \frac{3}{2}]$. This bound was first found through numerical results using the PEPit toolbox but has now been rigorously proven through mathematical proofs using interpolation inequalities. We can also show that this bound is tight and cannot be improved. To do so, we will demonstrate that a particular function reaches the above bounds.

Theorem 2.3. *Both rates proven in Theorem 2.1 and 2.2 are tight.*

Proof. Again, we consider the case $L = 1$ without loss of generality. The following univariate function, often called Huber function, is 1-smooth and convex:

$$f(x) = \begin{cases} \frac{1}{\sqrt{h}}|x| - \frac{1}{2h}, & \text{when } |x| \geq \frac{1}{\sqrt{h}} \\ \frac{1}{2}x^2, & \text{when } |x| < \frac{1}{\sqrt{h}} \end{cases}$$

Assuming that $x_0 \geq \frac{1}{\sqrt{h}}$, we have that $\nabla f(x_0) = \frac{1}{\sqrt{h}}$. The iterate x_1 is given by

$$x_1 = x_0 - h \frac{1}{\sqrt{h}} = x_0 - \sqrt{h}.$$

Choosing x_0 such that $x_1 = \frac{1}{\sqrt{h}}$, i.e. $x_0 = \sqrt{h} + \frac{1}{\sqrt{h}}$, it is straightforward to check that $f(x_0) - f(x_1) = 1$. Thus we have shown :

$$\begin{aligned} \|g_1\|^2 &= \frac{1}{h} \\ &= \frac{f(x_0) - f(x_1)}{h}. \end{aligned}$$

As those quantities are *exactly equal*, we have proven that this particular function reaches *exactly* the rate given in Theorem 2.2. This bound is thus tight.

The rate of Theorem 2.1 can be checked to be tight in a similar way, using the same f , x_0 , d_0 , x_1 and noticing that $f_* = 0$ for this function.

□

An analytical form for the rate of convergence representing the right regimes in Figure 2.1 (corresponding to $h \in [\frac{3}{2}; 2]$) has been found. Unfortunately, no general mathematical proof has been found for this regime. However, given a specific step size h , a proof using the numerical values for λ_{ij} can still be derived. As no general proof will be presented in this work, the following result is stated as a conjecture. It should however be noted that this conjecture relies on strong numerical evidence given by the *exact* solution of the (PEP) problem for given step sizes h .

Conjecture 2.1. Suppose that $x_1 = x_0 - \frac{h}{L}\nabla f(x_0)$ with $h \in [\frac{3}{2}; 2]$. If $f(\cdot)$ is an L -smooth convex function, then

$$\frac{1}{L}\|\nabla f(x_1)\|^2 \leq 2(1-h)^2(f(x_0) - f(x_*))$$

The results discussed above and the plots in Figure 2.1 leads us to the following conjecture:

Conjecture 2.2 (Best step size for one iteration of gradient descent). *The best step size for one iteration of gradient descent on a L -smooth convex function $f(\cdot)$ is $h_{opt} = \frac{3}{2L}$.*

This conjecture follows directly from the proof of Theorem 2.1, which becomes incorrect when $h > \frac{3}{2}$. As soon as $h > \frac{3}{2}$, we have (coming from Equation (2.2)):

$$\|g_1\|^2 \leq \frac{2}{2h+1}(f_0 - f_*) + \frac{2h-3}{2h+1}\|g_1 - g_0\|^2,$$

where the last term is non-negative and increases quickly with h . This observation, while not a rigorous demonstration, provides insight into the behaviour of the constant step size gradient descent method when $h > \frac{3}{2}$.

2.1.2 Multiple Steps of Exact Gradient

After the needed analysis on step of the gradient method, let's now analyze the more interesting case of several iterations of this method. Using the same setup as in Theorem 2.2, we can derive the rate of convergence of several steps of gradient descent. To do so, we will first need the following result:

Lemma 2.1 (Monotonicity of the gradient). Suppose that $x_1 = x_0 - \frac{h}{L}\nabla f(x_0)$ with $h \in [0; 2]$. If $f(\cdot)$ is an L -smooth convex function, then

$$\|\nabla f(x_1)\|^2 \leq \|\nabla f(x_0)\|^2.$$

Proof. Using the relation $x_0 - x_1 = \frac{h}{L}\nabla f(x_0)$, assuming without loss of generality that $L = 1$ and using the following interpolation inequalities:

$$\begin{aligned} Q_{01} : f_0 &\geq f_1 + \langle g_1, x_0 - x_1 \rangle + \frac{1}{2L} \|g_1 - g_0\|_2^2 \\ Q_{10} : f_1 &\geq f_0 + \langle g_0, x_1 - x_0 \rangle + \frac{1}{2L} \|g_0 - g_1\|_2^2 \end{aligned}$$

with the corresponding multipliers $\lambda_{01} = \lambda_{10} = 1$, we get:

$$\begin{aligned} 0 &\geq h \langle g_0, g_1 \rangle - h \|g_0\|^2 + \|g_0 - g_1\|^2 \\ &= (h - 2) \langle g_0, g_1 \rangle + (1 - h) \|g_0\|^2 + \|g_1\|^2 \\ &= \left(1 - \frac{h}{2}\right) \|g_0 - g_1\|^2 - \frac{h}{2} \|g_0\|^2 + \frac{h}{2} \|g_1\|^2 \end{aligned}$$

which leads to:

$$\left(1 - \frac{h}{2}\right) \|g_0 - g_1\|^2 \leq \frac{h}{2} \|g_0\|^2 - \frac{h}{2} \|g_1\|^2$$

and, as soon as $h \leq 2$, which is guaranteed by hypothesis, the left hand-side is non-negative, and we thus have:

$$0 \leq \|g_0\|^2 - \|g_1\|^2$$

□

Once the monotonicity of the gradient has been established in Lemma 2.1, it becomes possible to gain significant insight into the convergence behaviour of the gradient descent algorithm. This finding indicates that as the algorithm iterates, the norm of the gradient will either decrease or remain unchanged. It is crucial to comprehend this monotonic behaviour in order to analyze the convergence properties of gradient descent. This is because it ensures that the algorithm is consistently moving towards minimizing the objective function. With this assurance, we can proceed to examine the convergence rate for multiple iterations of gradient descent with confidence.

This leads us to the following theorem:

Theorem 2.4 (Rate of convergence of several iterations of gradient descent with step size $h \leq \frac{3}{2}$). *Starting with x_0 and supposing that $x_i = x_{i-1} - \frac{h}{L} \nabla f(x_{i-1})$ with $h \in [0; \frac{3}{2}]$. If $f(\cdot)$ is an L -smooth convex function, then*

$$\frac{1}{L} \|\nabla f(x_n)\|^2 \leq \frac{1}{nh} (f(x_0) - f(x_n)) \quad (2.3)$$

Proof. The proof will once again be performed without loss of generality for $L = 1$ and will be performed using an induction argument. Clearly, equation (2.3) is true for $n = 1$, as it is the one-step case, covered in Theorem 2.2. By induction, assume that

$$\|\nabla f(x_i)\|^2 \leq \frac{1}{ih}(f(x_0) - f(x_i)) \quad (2.4)$$

is true. Theorem 2.2 with initial point x_i gives:

$$\|\nabla f(x_{i+1})\|^2 \leq \frac{1}{h}(f(x_i) - f(x_{i+1})) \quad (2.5)$$

Combining inequalities (2.4) and (2.5) with multipliers 1 and $\frac{1}{i}$ respectively, we have:

$$\begin{aligned} \|\nabla f(x_i)\|^2 + \frac{1}{i}\|\nabla f(x_{i+1})\|^2 &\leq \frac{1}{ih}(f(x_0) - f(x_i)) + \frac{1}{ih}(f(x_i) - f(x_{i+1})) \\ &= \frac{1}{ih}(f(x_0) - f(x_{i+1})) \end{aligned}$$

Since Lemma 2.1 ensures that $\|\nabla f(x_{i+1})\|^2 \leq \|\nabla f(x_i)\|^2$, we obtain:

$$\left(1 + \frac{1}{i}\right) \|\nabla f(x_{i+1})\|^2 \leq \frac{1}{ih}(f(x_0) - f(x_{i+1}))$$

As $\left(1 + \frac{1}{i}\right) = \frac{i+1}{i}$, we get:

$$\|\nabla f(x_{i+1})\|^2 \leq \frac{1}{(i+1)h}(f(x_0) - f(x_{i+1})) \quad (2.6)$$

Thus, we have shown that equation (2.3) holds for $n = i + 1$. Since we've established the base case and demonstrated the inductive step, the proof is complete. $\square$

From this result, we can derive the proof with the other setup.

Theorem 2.5 (Rate of convergence of several iterations of gradient descent with step size $h \leq \frac{3}{2}$). *Starting with x_0 and supposing that $x_i = x_{i-1} - \frac{h}{L}\nabla f(x_{i-1})$ with $h \in [0; \frac{3}{2}]$. If $f(\cdot)$ is an L -smooth convex function, then*

$$\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \frac{2}{2nh + 1}(f(x_0) - f_*)$$

Proof. Starting from the result of the Theorem 2.4 (and again using $L = 1$ without loss of generality):

$$\|\nabla f(x_n)\|^2 \leq \frac{1}{nh}(f(x_0) - f(x_n)) \quad (2.7)$$

and the following interpolation inequality along with the first-order optimality condition:

$$\begin{aligned} Q_{n*} : f_n &\geq f_* + \langle g_*, x_n - x_* \rangle + \frac{1}{2} \|\nabla f(x_*) - \nabla f(x_n)\|^2 \\ &= f_* + \frac{1}{2} \|\nabla f(x_n)\|^2 \end{aligned}$$

Using multipliers Nh for (2.7) and $\lambda_{N*} = 1$, we get:

$$\begin{aligned} \left(nh + \frac{1}{2}\right) \|\nabla f(x_n)\|^2 &\leq (f(x_0) - f(x_n)) + (f(x_n) - f(x_*)) \\ &= f(x_0) - f(x_*) \end{aligned}$$

And thus:

$$\begin{aligned} \|\nabla f(x_n)\|^2 &\leq \frac{1}{nh + \frac{1}{2}}(f(x_0) - f(x_*)) \\ &= \frac{2}{2nh + 1}(f(x_0) - f(x_*)) \end{aligned}$$

□

The rates proved above can be shown to be tight using the same approach as in Theorem 2.3. Furthermore, it turns out that, in practice, Theorems 2.4 and 2.5 seem to hold even for larger values of h . This observation leads to the following conjecture, which is strongly supported by numerical evidence:

Conjecture 2.3. *Theorems 2.4 and 2.5 hold for step sizes $h \leq h_{\max}$, with $h_{\max}$ being the solution of the non-linear equation*

$$\frac{1}{2nh_{\max} + 1} = (1 - h_{\max})^{2n}$$

Furthermore, the second part of this equation comes from the observed rate of convergence of gradient method when $h \in [h_{\max}; 2]$, which is:

$$\frac{1}{L} \|\nabla f(x_n)\|^2 \leq 2(1-h)^{2n}(f(x_0) - f_*)$$

This conjecture is backed by strong numerical evidence obtained from solving (PEP) using the PEPit toolbox. The proposed rate of convergence aligns closely with the numerical results depicted in Figures 2.1 and 2.3. Although this rate has not been formally proven, the numerical results strongly suggest its validity. It's interesting to note that it was numerically found that the worst-case function for the constant gradient descent with $h \in [h_{max}; 2]$ is the same as for one step of the method (see Theorem 2.3). Also note that if the chosen h exceeds h_{max} , the convergence rate deteriorates rapidly, underscoring the importance of selecting an appropriate step size.

The value h_{max} depending on the performed number of iterations is illustrated in Figure 2.2.

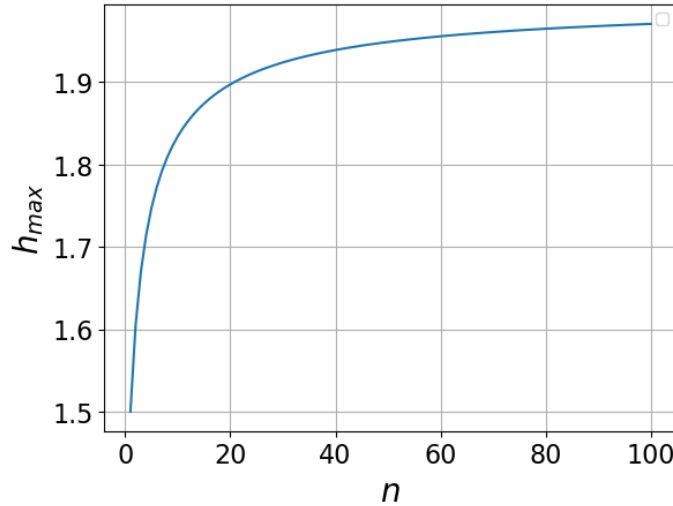


Figure 2.2: Numerical conjecture on the best step size h_{max} depending on the number of iterations n .

It can be observed that for $n = 1$, $h_{max} = 1.5$ and its value progressively increases with n , approaching $h_{max} \rightarrow 2$ as $n \rightarrow \infty$. This observation is particularly interesting, because, historically, the traditional choice of $h = 1$ (up to the usual factor $1/L$) was largely based on earlier theoretical work and empirical results. However, those findings, based on the recently developed Performance Estimation framework, suggest that a more nuanced approach to selecting the step size can yield significantly better performance. By optimizing h based on the number of iterations, we can achieve faster convergence and more efficient optimization processes. This has practical implications for fields such as machine learning, where large-scale optimization problems are common.

The significance of selecting an appropriate step size is further illustrated in Figure 2.3, which shows the rate of convergence as a function of the step size h for different numbers of iterations. Each minimum corresponds to the h_{max} for the given n , and can be compared with the corresponding value in Figure 2.2.

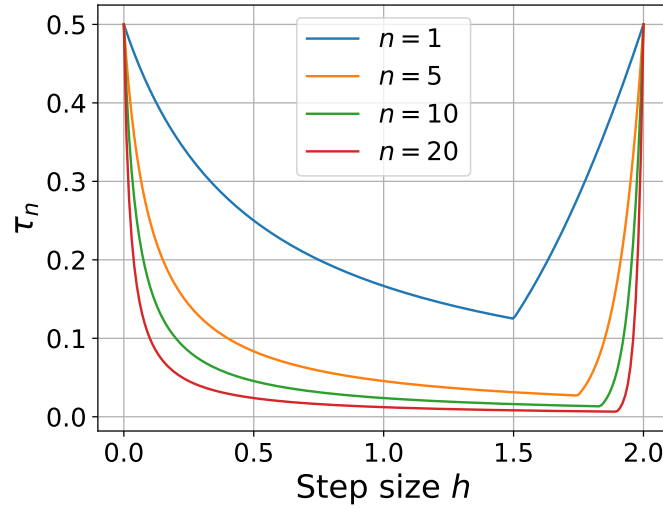


Figure 2.3: Numerical rate σ_n as a function of the step size h for one step of the gradient method and for different number of iterations. ($\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f_*)$)

On Figure 2.4, the numerical rate σ_n depending on n is shown for different choices of step sizes h .

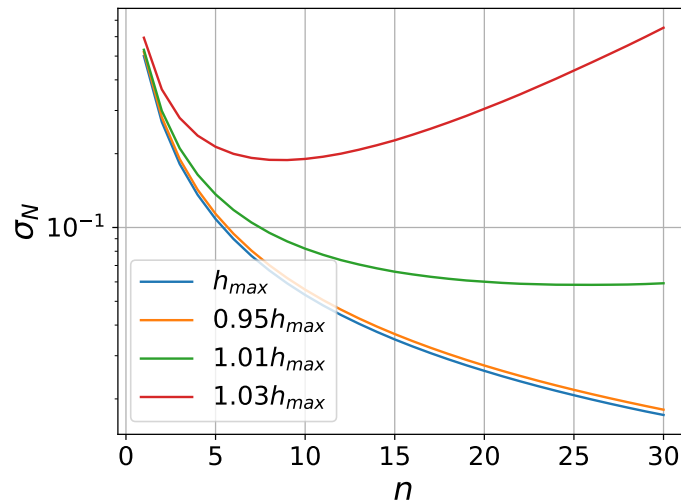


Figure 2.4: σ_n depending on n for different choices of h . h_{max} is defined above and is evaluated for each value of n . ($\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f_*)$)

We see that if h is chosen slightly shorter than h_{max} , the results don't vary

significantly. On the other hand, if h is poorly chosen as slightly longer than h_{max} , the results deteriorates quickly. This underlines the importance of choosing a slightly shorter step size rather than a slightly longer one.

A last remark on this best step size h_{max} is necessary. As h_{max} depends on the number of iterations, knowing the exact number of iterations in advance is crucial, which can be a limitation in practical scenarios. Nonetheless, using a slightly smaller step size than h_{max} is not problematic for the method's effectiveness. If more iterations than initially planned are performed, the method will still work reasonably well. However, performing fewer iterations than planned can lead to poor results, as the effective step size would then exceed the optimal h_{max} for the actual number of iterations performed, leading to degraded performance, as seen in Figure 2.4.

2.2 Inexact Gradient

We will now analyze inexact gradient descent introduced in Algorithm 2, with $\delta > 0$. As stated in the introduction on inexact gradient, the analysis of constant step size inexact gradient methods has already been done in [DKGT20] for the class of L -smooth μ -strongly convex functions, but cannot be extended to the class of functions analyzed here (see Section 1.3 for more information). The results set out and proven in this section are new and extend the work carried out previously. In this section, we will prove different rates of convergence for this method and conjecture a very good choice of step size when the size of inexactness δ is known. We will see that smaller step sizes are preferable when the gradient is inexact than when it is exact.

2.2.1 Single Step of Inexact Gradient

As in the exact case, let's start the analysis with one step of inexact gradient method.

Theorem 2.6 (Rate of convergence of one iteration of inexact gradient descent with step size $h \leq \frac{3}{2(1+\delta)}$ and $\delta < 1$). *Starting with x_0 and, suppose that $x_1 = x_0 - \frac{h}{L}d_0$ with $h \in]0; \frac{3}{2(1+\delta)}]$, $\delta \in]0; 1[$ and $\|d_0 - \nabla f(x_0)\|^2 \leq \delta^2 \|\nabla f(x_0)\|^2$. If $f(\cdot)$ is an L -smooth convex function, then*

$$\frac{1}{L} \|\nabla f(x_1)\|^2 \leq \frac{f(x_0) - f(x_1)}{h(1 - \delta)} \quad \text{and} \quad \frac{1}{L} \|\nabla f(x_1)\|^2 \leq \frac{f(x_0) - f(x_*)}{\frac{1}{2} + h(1 - \delta)}.$$

Proof. Using once again $L = 1$ without loss of generality, the interpolation inequalities

Q_{01} and Q_{10} with multipliers $\lambda_{01} = 2$ and $\lambda_{10} = 1$ and the inequality $\|d_0 - \nabla f(x_0)\|^2 \leq \delta^2 \|\nabla f(x_0)\|^2$ with multiplier $\frac{h}{2\delta}$, we get ⁶:

$$f_0 - f_1 \geq \left(\frac{3}{2} - h - \frac{h\delta}{2} \right) \|g_0\|^2 + \frac{3}{2} \|g_1\|^2 + \frac{h}{2\delta} \|D\|^2 + (2h-3)\langle g_0, g_1 \rangle + 2h\langle g_1, D \rangle - h\langle g_0, D \rangle,$$

which can be rewritten under matrix form:

$$f_0 - f_1 \geq \begin{pmatrix} g_0 & g_1 & D \end{pmatrix}^T \begin{pmatrix} (3/2 - h - h\delta/2) & (h - 3/2) & -h/2 \\ (h - 3/2) & 3/2 & h \\ -h/2 & h & h/2\delta \end{pmatrix} \begin{pmatrix} g_0 & g_1 & D \end{pmatrix}. \quad (2.8)$$

The central matrix can be rewritten as the following sum:

$$\begin{pmatrix} (3/2 - h - h\delta/2) & (h - 3/2) & -h/2 \\ (h - 3/2) & (3/2 - h(1 - \delta)) & h \\ -h/2 & h & h/2\delta \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & h(1 - \delta) & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

If the first matrix in the sum (called P) is positive semi-definite, then equation (2.8) will come down to a positive sum of squared norm, which will be non-negative. In this case, P could be removed from the inequality, and it will all come down to $f_0 - f_1 \geq h(1 - \delta)\|g_1\|^2$, which we want to prove.

To do so, we will use the following property of symmetric matrices (P is indeed symmetric). Suppose that a symmetric matrix is partitioned in four blocks as follows:

$$P = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}.$$

When the matrix C is invertible, the matrix $A - BC^{-1}B^T$ is called the *Schur complement of block C* of matrix P . Assuming that C is positive definite, it is well known that P is positive semi-definite if and only if its complement $A - BC^{-1}B^T$ is positive semi-definite.

In this case, we can decompose the P matrix as follows:

$$A = \begin{pmatrix} \frac{3}{2} - h - \frac{h\delta}{2} & -\frac{3}{2} + h \\ -\frac{3}{2} + h & \frac{3}{2} - h(1 + \delta) \end{pmatrix}, \quad B = \begin{pmatrix} -\frac{h}{2} \\ h \end{pmatrix}, \quad C = \frac{h}{2\delta}.$$

C is clearly positive definite and thus P is positive semi-definite if and only if $A - BC^{-1}B^T$ is. Let's write the Schur complement down:

⁶For compactness, the notation $D = d_0 - g_0$ has been used.

$$\begin{aligned}
 A - BC^{-1}B^T &= \begin{pmatrix} \frac{3}{2} - h - \frac{h\delta}{2} & -\frac{3}{2} + h \\ -\frac{3}{2} + h & \frac{3}{2} - h(1+\delta) \end{pmatrix} - \frac{2\delta}{h} \begin{pmatrix} \frac{h^2}{4} & -\frac{h^2}{2} \\ -\frac{h^2}{2} & h^2 \end{pmatrix} \\
 &= \begin{pmatrix} \frac{3}{2} - h(1+\delta) & -\frac{3}{2} + h(1+\delta) \\ -\frac{3}{2} + h(1+\delta) & \frac{3}{2} - h(1+\delta) \end{pmatrix} \\
 &= \left(\frac{3}{2} - h(1+\delta)\right) \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix},
 \end{aligned}$$

which is positive semi-definite if and only if $\frac{3}{2} - h(1+\delta) \geq 0$, which is guaranteed by the hypothesis $h \leq \frac{3}{2(1+\delta)}$. As the Schur complement of positive-definite matrix P is positive semi-definite, the matrix P is also positive semi-definite and thus $x^T Px \geq 0 \forall x$. We thus have that:

$$\begin{aligned}
 f_0 - f_1 &\geq \begin{pmatrix} g_0 & g_1 & D \end{pmatrix}^T \left(P + \begin{pmatrix} 0 & 0 & 0 \\ 0 & h(1-\delta) & 0 \\ 0 & 0 & 0 \end{pmatrix} \right) \begin{pmatrix} g_0 & g_1 & D \end{pmatrix} \\
 &\geq \begin{pmatrix} g_0 & g_1 & D \end{pmatrix}^T \begin{pmatrix} 0 & 0 & 0 \\ 0 & h(1-\delta) & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} g_0 & g_1 & D \end{pmatrix} \\
 &= h(1-\delta) \|g_1\|^2.
 \end{aligned}$$

To show the second rate of the theorem, we use another interpolation inequality (known as the descent lemma)

$$Q_{1*} : f_1 \geq f_* + \langle g_*, f_1 - f_* \rangle + \frac{1}{2} \|g_* - g_1\|^2 = f_* + \frac{1}{2} \|g_1\|^2,$$

and add it to $f_0 - f_1 \geq h(1-\delta) \|g_1\|^2$ to obtain the desired rate $f_0 - f_* \geq \left(\frac{1}{2} + h(1-\delta)\right) \|g_1\|^2$. $\square$

Let us now show that these rates cannot be improved.

Theorem 2.7. *Both rates proven in Theorem 2.6 are tight.*

Proof. Again we consider the case $L = 1$ without loss of generality. As in the proof of Theorem 2.3, consider the following Huber function (which is the same as before iff $\delta = 0$):

$$f(x) = \begin{cases} \frac{1}{\sqrt{h(1-\delta)}} |x| - \frac{1}{2h(1-\delta)}, & \text{when } |x| \geq \frac{1}{\sqrt{h(1-\delta)}} \\ \frac{1}{2} x^2, & \text{when } |x| < \frac{1}{\sqrt{h(1-\delta)}} \end{cases}$$

Assuming that $x_0 \geq \frac{1}{\sqrt{h(1-\delta)}}$, we have that $\nabla f(x_0) = \frac{1}{\sqrt{h(1-\delta)}}$. If we now choose $d_0 = (1 - \delta)\nabla f(x_0)$, which clearly satisfies the definition of δ inexactness, iterate x_1 is:

$$x_1 = x_0 - \sqrt{h(1 - \delta)}.$$

Now choosing x_0 such that $x_1 = \frac{1}{\sqrt{h(1-\delta)}}$, i.e. $x_0 = \sqrt{h(1 - \delta)} + \frac{1}{\sqrt{h(1-\delta)}}$, it is straightforward to check that $f(x_0) - f(x_1) = 1$ and we have shown that:

$$\begin{aligned} \|g_1\|^2 &= \frac{1}{h(1 - \delta)} \\ &= \frac{f(x_0) - f(x_1)}{h(1 - \delta)}. \end{aligned}$$

As those quantities are *equal* to each other, we have proven that this particular function with the particular acceptable direction $d_0 = (1 - \delta)\nabla f(x_0)$ achieves *exactly* the rate given in Theorem 2.6. This bound is thus tight.

The second rate can be checked to be tight in a similar way, using the same f , x_0 , d_0 , x_1 and the fact that $f_* = 0$.

□

Exploring Longer Step Sizes So far, we have shown the rates of convergence of one iteration of inexact gradient descent for $h \leq \frac{3}{2(1+\delta)}$. We will now also look at the numerical rates of convergence given by the PEPit toolbox [GMG⁺22] when $h > \frac{3}{2(1+\delta)}$. Although formal proofs for these scenarios are yet to be established, analyzing the numerical results provides valuable empirical understanding.

In Figure 2.5, we visualize the numerical convergence rate for a fixed $\delta = 0.1$, showcasing the rate σ_n as a function of the step size h .

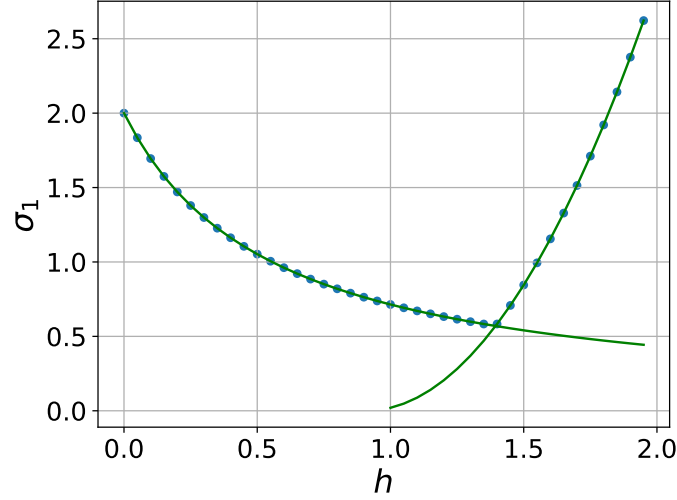


Figure 2.5: Rate σ_1 (blue dots) of one iteration of inexact gradient method ($\delta = 0.1$) as a function of the step size h . ($\frac{1}{L}\|\nabla f(x_1)\|^2 \leq \sigma_1(f(x_0) - f_*)$)

We can notice several things. First, for *small* step sizes $h \leq \frac{3}{2(1+\delta)}$, the rate aligns with the theoretical bounds established in Theorem 2.6 (left green line). However, beyond this threshold, the rate appears to follow the analytical rate $\|\nabla f(x_1)\|^2 \leq 2(1 - h(1 + \delta))^2(f(x_0) - f_*)$. Furthermore, similarly to the exact case, executing a too long gradient step tends to yield deteriorating outcomes compared to taking no action (i.e. $\sigma_1(h) > \sigma_1(0)$, when $h > \frac{2}{1+\delta}$).

Zooming in on the transitional zone between regimes, as depicted in Figure 2.6, reveals a surprising discovery: an intermediate regime exists between the two dominant regimes. This intermediate regime, situated between the left-regime (effective until $h = \frac{3}{2(1+\delta)}$) and the right-regime, presents a fascinating yet poorly understood phenomenon. It's also worth noticing that this intermediate-regime doesn't exist in the exact case ($\delta = 0$). A more precise analysis of this regime would bring us insights on how the method really works in this inexact context.

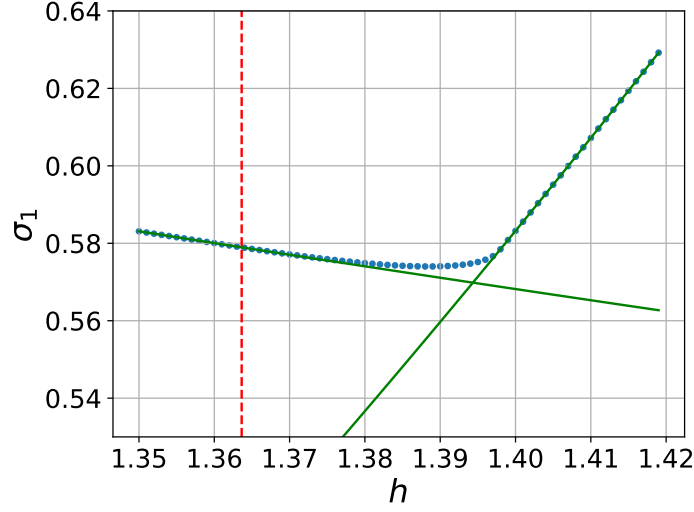


Figure 2.6: Zoom around change of regime zone of the rate σ_1 (blue dots) of one iteration of inexact gradient method ($\delta = 0.1$) as a function of the step size h . ($\frac{1}{L}\|\nabla f(x_1)\|^2 \leq \sigma_1(f(x_0) - f_*)$)

Given the current lack of comprehensive understanding surrounding this intermediate regime and recalling the principle emphasized in Section 2.1—that longer step sizes tend to yield inferior outcomes than shorter ones, a prudent choice for the step size of one iteration of inexact gradient descent is $h = \frac{3}{2(1+\delta)}$.

2.2.2 Multiple Steps of Inexact Gradient

Another rather surprising observation emerges when considering multiple steps of inexact gradient descent: the monotonicity of the gradient can no longer be guaranteed for $\delta > 0$, i.e. $\|g_{i+1}\|^2 \leq \|g_i\|^2$ does not hold in general for iterates generated by Algorithm 2. The following result given in [DKGT20, Theorem 5.3]:

$$\|\nabla f(x_1)\| \leq \left(1 - (1 - \delta)\frac{\mu}{L}h\right)\|\nabla f(x_0)\|$$

only holds for $\delta \in [0; \frac{2\mu}{L+\mu}]$. As we only consider the (not strongly) convex case here ($\mu = 0$), this result doesn't hold, except for the scenario where $\delta = 0$, which essentially corresponds to Theorem 2.1 in the exact case. Therefore, we will extend this analysis to the (not strongly) convex case, where the gradient monotonicity cannot be guaranteed in general.

Theorem 2.8. *Algorithm 2 applied to an L -smooth convex function with a constant step size $h \in]0; \frac{3}{2(1+\delta)}]$ and relative inexactness $\delta \in]0; 1[$, started from iterate x_0 ,*

generates iterates satisfying:

$$\|\nabla f(x_{i+1})\|^2 \leq \left(1 + \frac{h\delta^2}{2-h}\right) \|\nabla f(x_i)\|^2$$

Proof. Using $L = 1$ as always, and interpolation inequalities $Q_{i,i+1}$ and $Q_{i+1,i}$ with multipliers $\lambda_{i,i+1} = \lambda_{i+1,i} = 2$ and inexact-inequality $\|D_i\|^2 \leq \delta^2 \|g_i\|^2$ with multiplier $\frac{h^2}{2-h}$, we get⁷:

$$0 \geq \left(2-2h-\frac{h^2\delta^2}{2-h}\right) \|g_i\|^2 + 2\|g_{i+1}\|^2 + (2h-4)\langle g_i, g_{i+1} \rangle + 2h\langle g_{i+1}, D_i \rangle - 2h\langle g_i, D_i \rangle + \frac{h^2}{2-h} \|D_i\|^2.$$

As in the proof of Theorem 2.6, we can express the above inequality in matrix form:

$$0 \geq \begin{pmatrix} g_i & g_{i+1} & D_i \end{pmatrix}^T \begin{pmatrix} 2-2h-\frac{h^2\delta^2}{2-h} & h-2 & -h \\ h-2 & 2 & h \\ -h & h & \frac{h^2}{2-h} \end{pmatrix} \begin{pmatrix} g_i & g_{i+1} & D_i \end{pmatrix}.$$

The central matrix above can be rewritten as the following sum:

$$\begin{pmatrix} 2-h & h-2 & -h \\ h-2 & 2-h & h \\ -h & h & \frac{h^2}{2-h} \end{pmatrix} + \begin{pmatrix} -h-\frac{h^2\delta^2}{2-h} & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

If we can show that the first term, which we will call P , is positive semidefinite, then it will imply $0 \geq \left(-h-\frac{h^2\delta^2}{2-h}\right) \|g_0\|^2 + h\|g_1\|^2$, which is equivalent to the rate we want to prove.

As in the proof of Theorem 2.6, we will use the Schur complement of the matrix P to show it's positive semidefinite. We thus get the following submatrices:

$$A = \begin{pmatrix} 2-h & h-2 \\ h-2 & 2-h \end{pmatrix}, \quad B = \begin{pmatrix} -h \\ h \end{pmatrix}, \quad C = \frac{h^2}{2-h}.$$

The Schur complement $A - BC^{-1}B^T$ reads:

$$\begin{pmatrix} 2-h & h-2 \\ h-2 & 2-h \end{pmatrix} - \frac{2-h}{h^2} \begin{pmatrix} h^2 & -h^2 \\ -h^2 & h^2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

⁷Using again the notation $D_i = d_i - g_i$

which is clearly positive semi-definite, thus implying the positive semi-definiteness of P , since $C = \frac{h^2}{2-h} > 0$, when $h < 2$ (as ensured by the hypothesis). Consequently:

$$\begin{aligned} 0 &\geq \begin{pmatrix} g_i & g_{i+1} & D_i \end{pmatrix}^T \left(P + \begin{pmatrix} -h - \frac{h^2\delta^2}{2-h} & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & 0 \end{pmatrix} \right) \begin{pmatrix} g_i & g_{i+1} & D_i \end{pmatrix} \\ &\geq \begin{pmatrix} g_i & g_{i+1} & D_i \end{pmatrix}^T \begin{pmatrix} -h - \frac{h^2\delta^2}{2-h} & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} g_i & g_{i+1} & D_i \end{pmatrix} \\ &= \left(-h - \frac{h^2\delta^2}{2-h} \right) \|g_i\|^2 + h \|g_{i+1}\|^2, \end{aligned}$$

and dividing everything by h leads to:

$$\left(1 + \frac{h\delta^2}{2-h} \right) \|g_i\|^2 \geq \|g_{i+1}\|^2.$$

□

The rate shown in Theorem 2.8 aligns exactly with the outcomes provided by the PEPit toolbox, which is an exact solver. The high numerical evidence thus strongly supports the correctness, but also the tightness of the result shown in Theorem 2.8. Proving rigorously the tightness of this result seems difficult, as the worst-case function (needed to prove this result) appears to be high-dimensional and difficult to identify numerically.

The results given in Theorem 2.8, depending on both the step size h and the inexactness δ , can be seen on Figure 2.7.

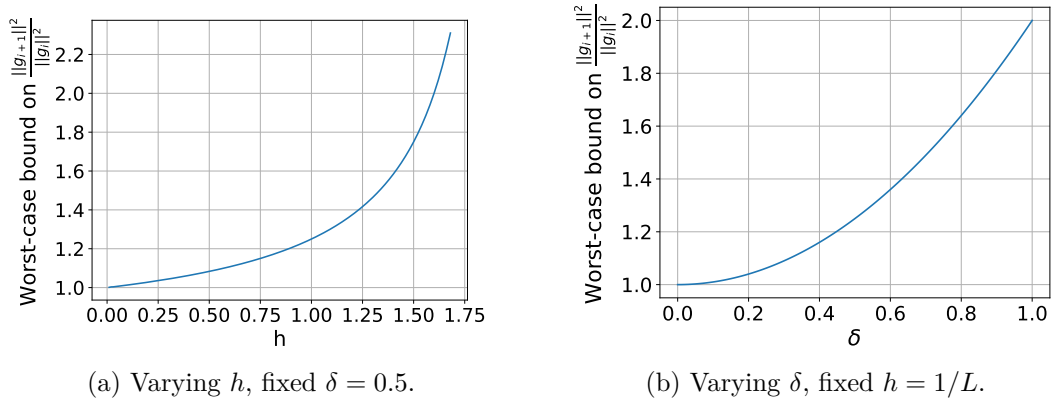


Figure 2.7: Worst-case bounds on $\frac{\|g_{i+1}\|^2}{\|g_i\|^2}$ depending on h and δ for inexact gradient method.

As expected, a higher value of δ will lead to a worsened worst-case ratio $\frac{\|g_{i+1}\|^2}{\|g_i\|^2}$. We can however also note that a higher value of step size h can lead to high values for the ratio. This observation implies that when the application demands a significant reduction in the gradient value at each iteration, opting for smaller step sizes h becomes prudent. Also note that the left plot was made with a high fixed value of δ . Such a great inexactness will obviously have huge impact on the method.

If we now look at several iterations of Algorithm 2, and using the above results, we can show that, in general, the rate for the last iterate of the method differs from the rate for the best among all iterates. For both rates, non-tight bounds will be given before making a graphical comparison between those two rates.

One might question the preference for selecting the last iterate over the best one. The rationale behind this decision is quite straightforward: in certain scenarios, preserving memory resources to store the current best iterate throughout the algorithm proves impractical. Consequently, resorting to the last iterate emerges as a practical solution to this challenge.

Theorem 2.9. *Algorithm 2 applied to an L -smooth convex function with a constant step size $h \in]0; \frac{3}{2(1+\delta)}]$ and relative inexactness $\delta \in]0; 1[$, started from iterate x_0 , generates iterates satisfying*

$$\frac{1}{L} \min_{k \in \{1, \dots, n\}} \|\nabla f(x_k)\|^2 \leq \frac{f(x_0) - f(x_*)}{\frac{1}{2} + nh(1 - \delta)}.$$

Proof. Considering once again the case $L = 1$, and writing the first rate in Theorem 2.6 on consecutive iterates (x_k, x_{k+1}) gives $h(1 - \delta)\|\nabla f(x_{k+1})\|^2 \leq f(x_k) - f(x_{k+1})$. Summing from $k = 0$ to $n - 1$, and adding the descent lemma $f(x_n) - f(x_*) \geq \frac{1}{2}\|\nabla f(x_n)\|^2$ as in the last step in the proof of Theorem 2.6 gives:

$$\begin{aligned} \frac{1}{2}\|\nabla f(x_n)\|^2 + h(1 - \delta) \sum_{k=0}^{n-1} \|\nabla f(x_{k+1})\|^2 &\leq \sum_{k=0}^{n-1} (f(x_k) - f(x_{k+1})) + (f(x_n) - f_*) \\ &= f(x_0) - f(x_*). \end{aligned} \tag{2.9}$$

Using the fact that every squared gradient in the left-hand side is greater than the one with the minimum value among them gives the sought inequality

$$\left(\frac{1}{2} + nh(1 - \delta)\right) \min_{k \in \{1, \dots, n\}} \|\nabla f(x_k)\|^2 \leq f(x_0) - f(x_*).$$

□

Hence, Algorithm 2 will converge for any $\delta \in]0; 1[$, with nearly the same sublinear convergence rate as in the exact case except for an extra $(1 - \delta)$ factor.

Now using naively the result given in Theorem 2.8, and starting from equation (2.9), we get that:

$$\frac{1}{2} \|\nabla f(x_n)\|^2 + h(1 - \delta) \sum_{k=0}^{n-1} \left(\frac{1}{\left(1 + \frac{h\delta^2}{2-h}\right)} \right)^{n-k-1} \|\nabla f(x_n)\|^2 \leq f(x_0) - f(x_*).$$

We see that:

$$\sum_{k=0}^{n-1} \left(\frac{1}{\left(1 + \frac{h\delta^2}{2-h}\right)} \right)^{n-k-1} = \sum_{k=0}^{n-1} \left(\frac{1}{\left(1 + \frac{h\delta^2}{2-h}\right)} \right)^k = \sum_{k=0}^{n-1} \left(\frac{2-h}{2-h+h\delta^2} \right)^k = \frac{1 - \left(\frac{2-h}{2-h+h\delta^2} \right)^n}{1 - \left(\frac{2-h}{2-h+h\delta^2} \right)},$$

where we used the properties of geometric series, to finally get:

$$\left(\frac{1}{2} + h(1 - \delta) \frac{1 - \left(\frac{2-h}{2-h+h\delta^2} \right)^n}{1 - \left(\frac{2-h}{2-h+h\delta^2} \right)} \right) \|\nabla f(x_n)\|^2 \leq f(x_0) - f(x_*)$$

The result above is plotted on Figure 2.8. We should note that both rate are valid, but *a priori* not tight. For the rate on the last iterate, we can hope for better results than the rate proven in Theorem 2.8 to the power n . Indeed, for example, with $h = 1$ and $\delta = 0.5$, then the theoretical rate for two iterations would give $\|g_2\|^2 \leq 1.25^2 \|g_0\|^2 = 1.5625 \|g_0\|^2$, the numerical rate given by PEPit is $\|g_2\|^2 \leq 1.3636 \|g_0\|^2$. However, we can know for sure that the last iterate can only be better (or equal) to the rate developed above and shown on the below plots:

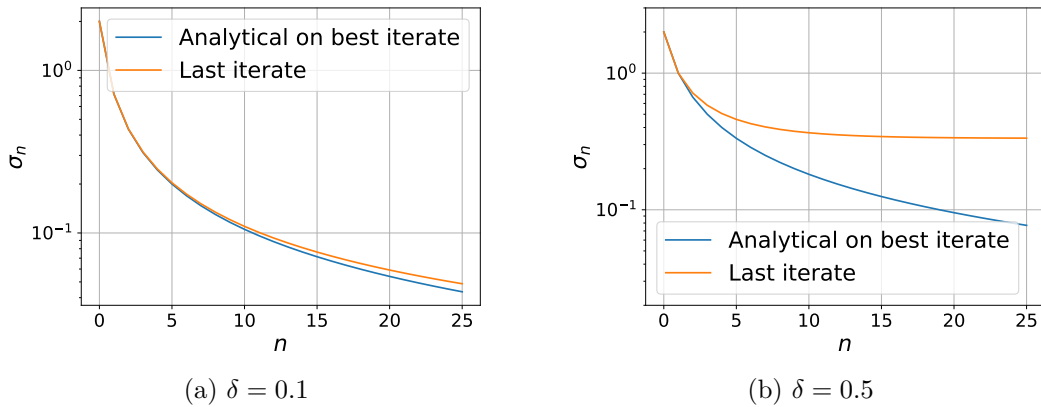


Figure 2.8: (Non-tight) rates for the best and last iterates based on the number of iterations of inexact gradient method ($h = 1$). $(\frac{1}{L} \|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f(x_*)))$

The behaviour observed doesn't vary significantly with the step size h . However, there is a notable impact from changes in the inexactness parameter δ , particularly evident in the behaviour of the last iterate. Based on the observations made here, when δ is not so big, the last iterate is a pretty good approximation of the best one, which is *a priori* not the case for a bigger δ . It's crucial to note once more that both rates are not tight and that better results can probably be found for them. The last iterate *could* however maybe also be a good approximation of the best one, even for larger values of δ .

Furthermore, if we look at the numerical tight rates for the last iterate given by the PEPit toolbox (shown on Figure 2.9), we observe a remarkable proximity between the (tight) numerical rate for the last iterate and the rate for the best iterate, which may not necessarily be tight. It once again suggests that the rate on the last iterate is a *very good* approximation for the rate on the best one.

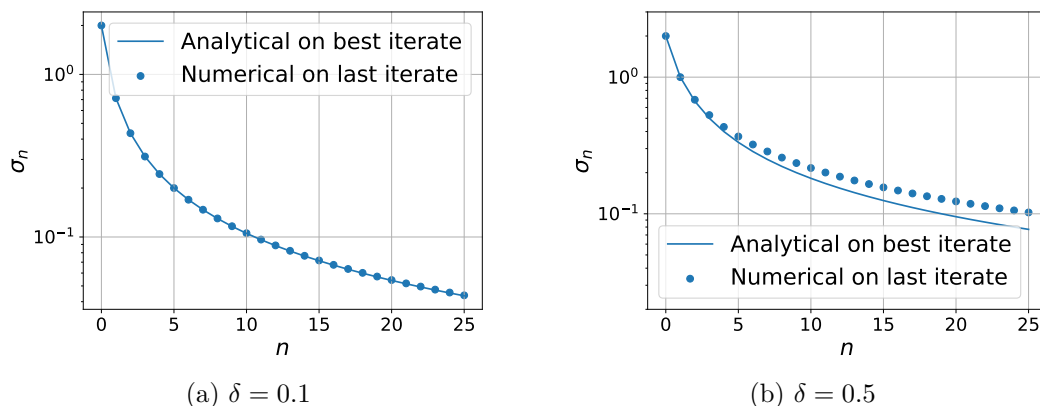


Figure 2.9: (Non-tight) analytical rate for the best iterate and (tight) numerical rate for the last iterate based on the number of iterations of inexact gradient method ($h = 1$). $(\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f(x_*)))$

Extension to longer step sizes We can also look at the behaviour of Algorithm 2 when $h > \frac{3}{2(1+\delta)}$. We analyze (tight) numerical rates provided by the PEPit toolbox, recognizing that while these rates are not mathematically proven, they offer valuable insights. As conjectured above, when δ is *small enough*, looking at the last iterate is a very good approximation of looking at the best iterate. We will thus look at the rate for the last iterate for different number of iterations, and depending on h (δ will here be fixed at 0.1, a value where the last iterate provides a reasonably accurate approximation of the best one).

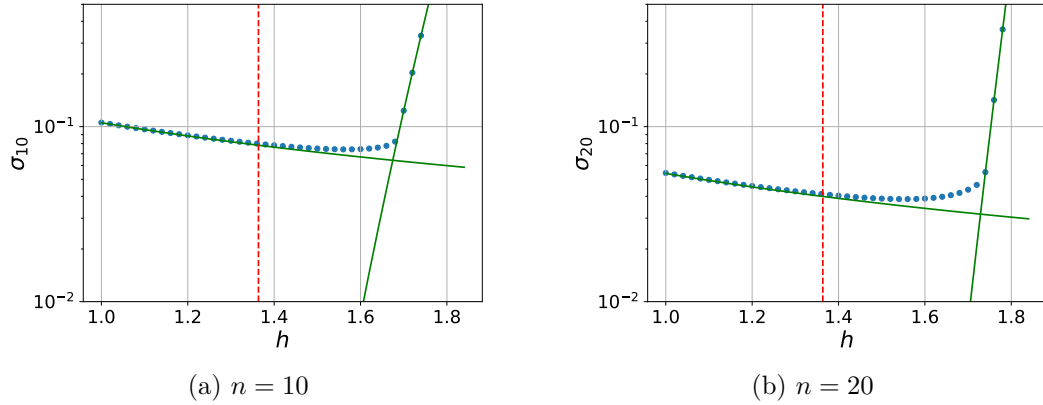


Figure 2.10: (Tight) numerical rate (blue dots) for the last iterate based on step size h . The vertical red dotted line corresponds to $h = \frac{3}{2(1+\delta)}$. $(\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f(x_*)))$

Interestingly, unlike the exact case, the threshold step size h at which the left-regime changes remains constant at $\frac{3}{2(1+\delta)}$ regardless of the number of iterations n . However, the right-regime well starts sooner for a smaller value of h . The intermediate-regime, though not thoroughly understood, extends with increasing n . However, as in the one-step case, the rate seems to appear relatively stable in this intermediate-regime. Consequently, in the inexact case, opting for the value $h = \frac{3}{2(1+\delta)}$ is a prudent choice for any number of iterations n . This result may however change with a better understanding of the intermediate-regime. This is a huge difference from the exact case, for which the choice for the best step size grows with n .

3 Silver Step Schedule

This section will be less theoretical than the previous one and will aim to experimentally analyze the silver schedule gradient based methods, introduced in [AP23], using the Performance Estimation Problem (PEP) methodology, with a particular focus on the PEPit toolbox. This step schedule aims at gradient-based methods while still maintaining the memoryless property (i.e. there is *a priori* no need to store the previous iterates), unlike the accelerated gradient ([Nes83]), for example. No theoretical result will be proved in this section, unlike in the previous one, but numerical and graphical analysis of the silver methods will be performed.

First, we will analyze the silver step schedule introduced in [AP23]. Subsequently, we will introduce and analyze new schemes, particularly those more robust to inexact gradient (see Section 1.3 for an introduction on inexact gradient).

As in Section 2, those results will be performed on the class of L -smooth convex functions. Furthermore, all step sizes will also always be normalized with the constant $1/L$.

3.1 Introduction to Silver Step Schedule

After analyzing (exact and inexact) gradient method with constant step size, we will now focus ourselves on another kind of algorithm, namely the so-called *fixed sequential step algorithms*.

Definition 3.1 (Fixed sequential step algorithm). *A fixed sequential step algorithm is an algorithm based on a fixed sequence of step sizes.*

Note that the constant step size gradient method is a particular case of fixed sequential step algorithms.

The search for the best possible fixed sequential step algorithm (i.e. the one which leads to the best rate of convergence), both for a given number of iterations or not, is still an open question, and several papers proposed such different sequences. The sequence which we will be focusing on in this work is the *silver step size schedule*, proposed in [AP23]. This sequence is generated recursively. The authors of this paper only propose sequences of lengths $n = 2^k - 1, k \in \mathbb{N}$. This is clearly a limitation, as we will only be able to correctly use, and analyze, the algorithm for the number of iterations respecting the above condition (i.e. such that $n = 2^k - 1, k \in \mathbb{N}$).

Nevertheless, we will see that this algorithm is promising for finding the optimal fixed sequence for the specified iteration counts, $n = 2^k - 1, k \in \mathbb{N}$. The silver step size schedule is recursively defined as follows (vector $h_{1:n}$ being the full list of step sizes for n iterations, and with, for example, h_1 being the first element of this list, i.e. the step size of the first iteration):

$$\begin{cases} h_1 &:= [\sqrt{2}] \\ h_{1:2n+1} &:= [h_{1:n}, 1 + \rho^{k-1}, h_{1:n}], \end{cases} \quad (3.1)$$

where $\rho := 1 + \sqrt{2}$ is referred to as the *silver ratio*.

A closer look at the silver schedule shows us that it is in fact fractal. It can be seen by observing it on Figure 3.1. This means that whatever the scale at which we look at the schedule, the same pattern will be observed. For example, the step size in the middle of the schedule will always be the biggest. We can also observe that this schedule is *symmetric* (recall that the schedule is only defined for lengths $n = 2^k - 1, k \in \mathbb{N}$).

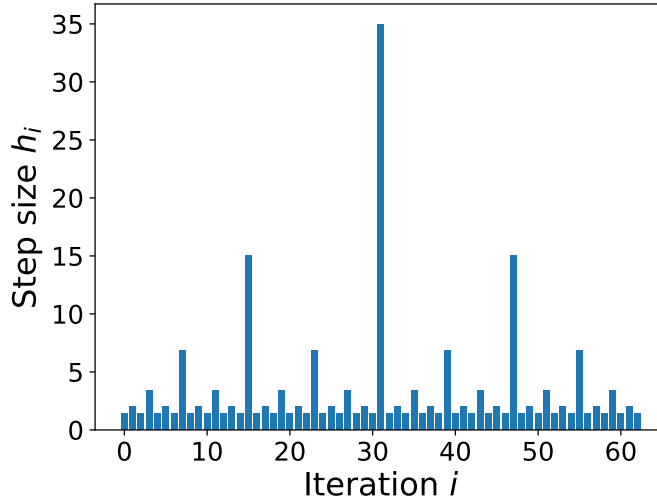


Figure 3.1: Visual representation of the silver schedule for $n = 2^6 - 1$ ($k = 6$).

This schedule can seem very strange at first glance. Indeed, we can see that very large step sizes are used (often repeatedly) in this pattern, while only small step sizes are usually used in optimization algorithms, especially when getting close to the solution (for example, in Section 2.1, we have shown that we can only guarantee a decrease in function value for $h < 2$). This strange phenomenon can be roughly explained: it uses a large step size to go far from the current iterate before using a well-chosen sequence of smaller step sizes⁸ to find a better feasible solution than

⁸Note that this sequence is chosen to perform well on the worst-case function in the considered function class, the class of L -smooth convex functions.

the one before the longer step. Before showing this phenomenon graphically, let's introduce the silver schedule algorithm:

Algorithm 3 Silver Schedule Gradient Descent

- 1: Given an L -smooth convex function f and a starting iterate x_0
- 2: Initialize the silver schedule h to have a length of at least `max_iter` such that:

$$\begin{cases} h_1 &:= [\sqrt{2}] \\ h_{1:2n+1} &:= [h_{1:n}, 1 + \rho^{k-1}, h_{1:n}] \end{cases}$$

- 3: $k \leftarrow 0$
 - 4: **while** $k < \text{max_iter}$ **do**
 - 5: Compute the next iterate as $x_{k+1} = x_k - \frac{h_{k+1}}{L} \nabla f(x_k)$
 - 6: $k \leftarrow k + 1$
 - 7: **end while**
-

Now that the silver schedule gradient descent is clearly defined, we can look at the phenomenon discussed above (i.e. the combination of long step with by a well-chosen sequence of smaller step sizes decreasing the function value) on a Huber objective function (recall that this was the worst-case function for constant step size gradient descent, see Theorems 2.3 and 2.7).

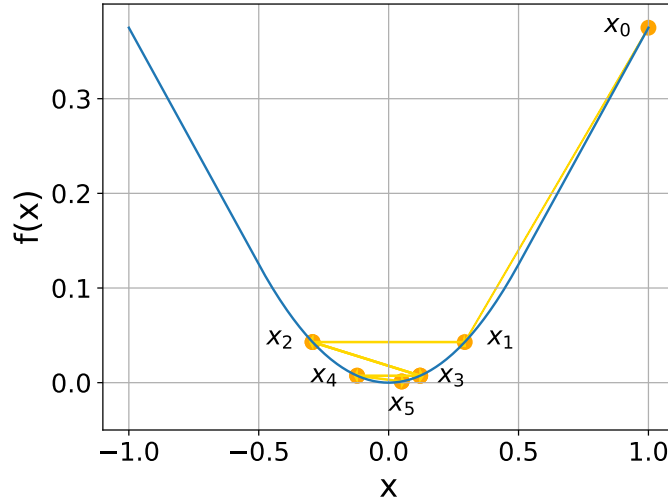


Figure 3.2: Visual representation of the silver schedule gradient descent on a Huber function. It can be seen that the method deliberately goes further from the minimum (going from x_3 to x_4) to get a better solution on the next iterate x_5 .

It can indeed be seen that the method deliberately goes further from the minimum to then find a better next iterate (note that two more iterations should be performed to reach a *good* number of iterations n respecting $n = 2^k - 1, k \in \mathbb{N}$). This is somehow counterintuitive and difficult to catch from a mathematical point of view, but we

will see that the silver schedule gradient descent is in fact a good algorithm and that, at well-chosen number of iterations $n = 2^k - 1, k \in \mathbb{N}$, it will perform better than the constant step size gradient descent, analyzed in previous section. The authors of [AP23] assert that their method (i.e., the silver step size gradient descent) achieves optimal convergence rates up to a constant factor among all possible fixed sequential step algorithms.

In the next subsection, our analysis, carried out using the PEP methodology (note that the PEP problem is still convex because the step sizes are fixed), will be compared with theirs and with other methods, such as the classical gradient descent.

In this Section 3, the setup used for the analysis is $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$. Recall that this setup can easily be translated to the other ones.

3.2 Convergence Rate Analysis and Comparison with the Classical Gradient Descent

3.2.1 Comparison with Classical Gradient Descent

The first analysis, using the PEPit toolbox, examines whether the silver schedule gradient method outperforms the classical gradient descent (as stated in [AP23]). The next plot shows the rates obtained for the *good* number of iterations of the silver schedule (i.e. the number of iterations $n = 2^k - 1, k \in \mathbb{N}$) and for the classical gradient descent (recall that it is a constant step size gradient descent with $h = 1/L$).

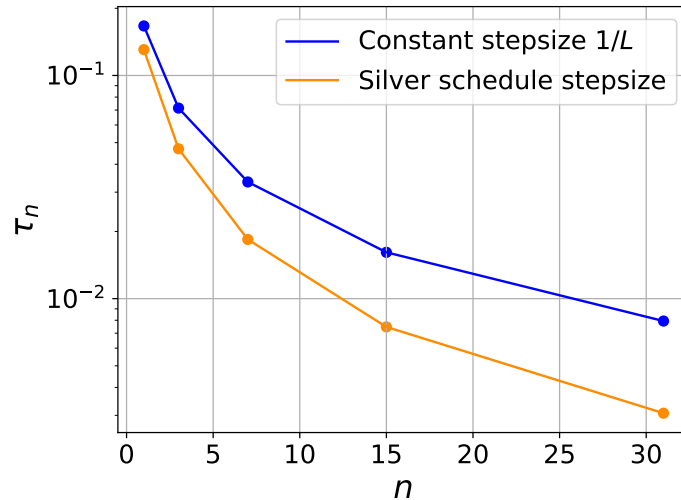


Figure 3.3: Comparison of the rates of silver schedule gradient descent with classical one for the *good* number of iterations $n = 2^k - 1, k \in \mathbb{N}$. ($\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

At first glance, it is obvious that the silver schedule performs better than the

classical gradient descent. It even seems that the gap between the two rates grows faster than a constant term with n , suggesting a rate of convergence better than $\mathcal{O}(\frac{1}{n})$ for the silver schedule gradient descent! This improved convergence will be discussed more theoretically in Section 3.2.2.

A limitation of the approach shown in Figure 3.3 is that we can only see the performance of the silver schedule gradient descent when it works best (at its *good* number of iterations). We also need to analyze its performance for other numbers of iterations. The results are displayed on the following plot.

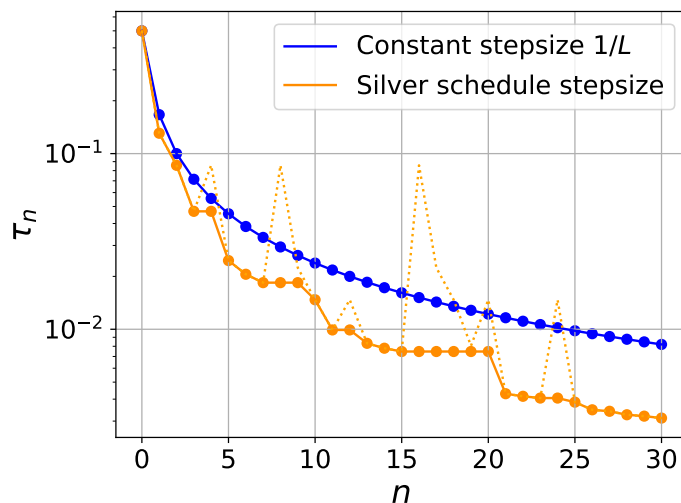


Figure 3.4: Comparison of the rates of silver schedule gradient descent with classical gradient descent with respect to the number of iterations. The thin orange line represents the rate of the silver schedule gradient descent after the current number of iterations, while the thick orange line shows the best rate achieved so far by storing the best iterate in memory. $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

Several observations can be made from the above plot. Firstly, without storing the best iterate (represented by the thin orange line), the silver schedule gradient descent can sometimes perform significantly worse than the classical gradient descent for specific iteration counts. The worst iteration counts occur just after a very large step (i.e. when the number of iterations n is such that $n = 2^k, k \in \mathbb{N}$). Indeed, as explained in the previous section, those very large step sizes make the algorithm go far from the optimal solution before gradually coming back to a better solution. The rate just after a large step is thus bad.

A simple way to improve the performance of the silver schedule gradient descent is by storing the current best iterate. It means that when a large step is performed, if an output is requested to the algorithm, it will return the previous best iterate. This can be seen by the thick orange line on the plot. This ensures that the silver schedule gradient descent consistently outperforms the classical gradient descent. The drawback is the additional memory cost required to store the current best iterate.

3.2.2 Improving the Convergence Rate from Altschuler and Parrilo

In [AP23], the authors give the following worst-case bound for the silver schedule gradient descent (for numbers of iterations n such that $n = 2^k - 1, k \in \mathbb{N}$):

$$r_k := \frac{1}{1 + \sqrt{4\rho^{2^k} - 3}}$$

This bound is, according to the authors, asymptotically tight, but, comparing the result given by the authors with the one obtained using the PEPit toolbox (an exact numerical solver), shows that this statement is not completely correct. Indeed, the ratio between the rate they provide, and the one found by the PEPit toolbox tends to 2 as n tends to infinity. Depending on the chosen definition of *asymptotically tight*, their statement can thus be seen as incorrect. The comparison between the two rates is illustrated on Figure 3.5.

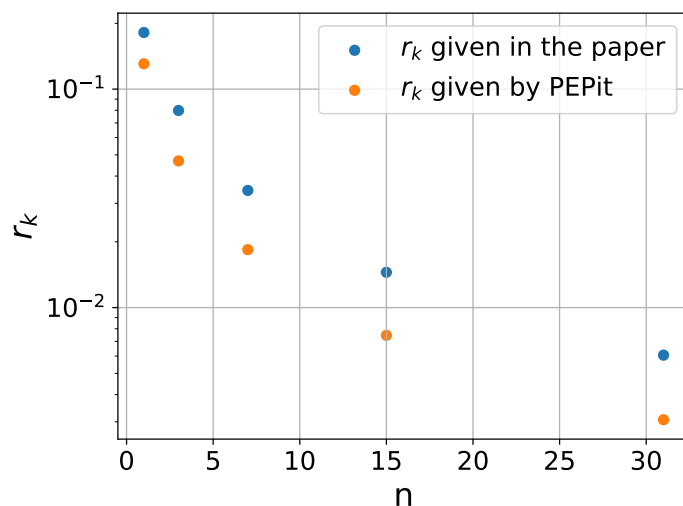


Figure 3.5: Comparison between the theoretical rate r_k given in [AP23] and the rate obtained using the exact solver in the PEPit toolbox.

It is clear that the observed worst-case rate r_k found using the PEP methodology is smaller than the one provided by the authors. This indicates that while the rate they give is correct, it is not tight. Based on the numerical evidence provided by the PEPit toolbox, we conjecture the following tight rate of convergence for the silver schedule gradient descent:

Conjecture 3.1. *Algorithm 3, when applied to an L -smooth convex function starting*

from iterate x_0 , generates iterates that, for any horizon $n = 2^k - 1, k \in \mathbb{N}$, satisfy:

$$\frac{1}{L}(f(x_n) - f_*) \leq \frac{1}{4\rho^k - 2} \|x_0 - x_*\|^2$$

This conjecture is supported by strong numerical evidence from the numerical solution of the (PEP) provided by the PEPit toolbox. However, this result has very recently been proved, but not yet published. I got this as a personal communication from M. Zamani (Dec 2023). We can further elaborate on this result as follows:

$$\begin{aligned} \frac{1}{L}(f(x_n) - f_*) &\leq \frac{1}{4\rho^k - 2} \|x_0 - x_*\|^2 \\ &\approx \frac{1}{4\rho^{\log_2(n)}} \|x_0 - x_*\|^2 \\ &= \frac{1}{4n^{\log_2(\rho)}} \|x_0 - x_*\|^2 \\ &\approx \frac{1}{4n^{1.2716}} \|x_0 - x_*\|^2, \end{aligned}$$

which clearly indicates a better rate of convergence compared to the constant step size gradient descent, for which $\frac{1}{L}(f(x_n) - f_*) \leq \frac{cst}{n} \|x_0 - x_*\|^2$ (where only the constant term cst can be improved by choosing the optimal step size, as discussed in Section 2.1).

Given this (assumed correct) tight worst-rate of convergence, we can thus get the number of iterations to get an ϵ -approximate solution with the silver schedule gradient descent.

Conjecture 3.2 (Number of iterations to get an ϵ -approximate solution with silver schedule gradient descent on L -smooth convex function). *For any horizon $n = 2^k - 1, k \in \mathbb{N}$, any dimension d , any L -smooth convex function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, and any initialization $x_0 \in \mathbb{R}^d$, in order to achieve an error of $f(x_n) - f_* \leq \epsilon$, it is sufficient to run Algorithm 3 for:*

$$n \leq \left(\frac{1}{2} + \frac{L\|x_0 - x_*\|^2}{4\epsilon} \right)^{\log_2 2} \leq 1 + \left(\frac{L\|x_0 - x_*\|^2}{4\epsilon} \right)^{0.786} \text{ iterations.}$$

Proof. This can be shown using the result from Conjecture 3.1 and the necessary condition $\frac{1}{4\rho^k - 2} \|x_0 - x_*\|^2 \leq \epsilon$. The second part of the inequality comes from the fact that $\log_2 2 \approx 0.786 \leq 1$ and the fact that $(a + b)^x \leq a^x + b^x \forall a, b$ when $x \leq 1$. $\square$

This result is nearly twice as good as the one provided in [AP23]. The PEP methodology has enabled us to obtain more precise results than those provided in the initial paper, demonstrating that this methodology can sometimes outperform traditional approaches to determining rates of convergence.

3.3 Inexact Gradient and Modified Silver Schedule

While the silver schedule gradient descent method performs well, it seems somewhat contrived, and potential limitations may exist. To assess its robustness, it is important to examine its behaviour when the gradient is inexact. Here, we employ the same *relative* definition of inexactness used in Sections 1.3 and 2.2.

The inexact silver schedule gradient descent combines the principles of Algorithms 2 and 3, and can be described as follows:

Algorithm 4 Inexact Silver Schedule Gradient Descent

- 1: Given an L -smooth convex function f , a starting iterate x_0 , and a relative inexactness δ
- 2: Initialize the silver schedule h to have a length of at least `max_iter` such that:

$$\begin{cases} h_1 &:= [\sqrt{2}] \\ h_{1:2n+1} &:= [h_{1:n}, 1 + \rho^{k-1}, h_{1:n}] \end{cases}$$

- 3: $k \leftarrow 0$
- 4: **while** $k < \text{max_iter}$ **do**
- 5: Let d_k be an approximate gradient with δ relative inexactness, i.e.

$$\|d_k - \nabla f(x_k)\|^2 \leq \delta^2 \|\nabla f(x_k)\|^2$$

- 6: Compute the next iterate as $x_{k+1} = x_k - \frac{h_{k+1}}{L} d_k$
 - 7: $k \leftarrow k + 1$
 - 8: **end while**
-

Hereunder, we can see the rate of convergence of the inexact silver schedule gradient descent for different values of δ and compare it to its exact version (thus with $\delta = 0$).

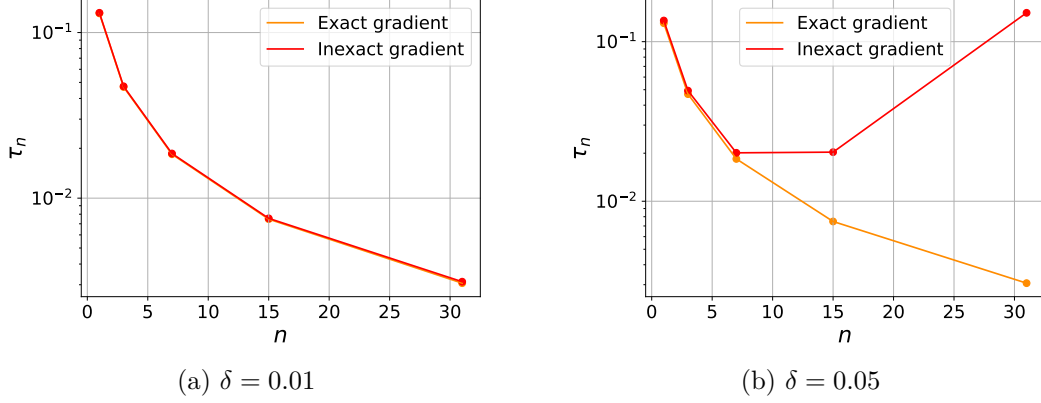


Figure 3.6: Comparison of the convergence rates of the silver schedule gradient descent with and without inexactness, for two different values of δ , depending on the number of iterations n . ($\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

It is evident that when the gradient is inexact, the silver schedule gradient descent starts to diverge at some point. Indeed, it has been observed that $\forall \delta > 0, \exists n_p$ such that Algorithm 4 starts to diverge for $n > n_p$. This means that the silver schedule is not *robust* to inexact gradient. This observation motivates the exploration in the next section.

3.3.1 Introduction to Modified Silver Schedule

Through experimentation with the silver schedule, we observed that adding two well-chosen step sizes at the beginning improves performance on inexact gradient without affecting performance on exact gradient. The *modified* silver schedule, denoted as $\tilde{h}_n$, is defined as follows, where $\tilde{h}_{n,i}$ is the i th element of the modified silver schedule of length n and $h_{n,i}$ is the i th element of the original silver schedule of length n defined in Equation (3.1):

$$\tilde{h}_{1:n} = [\sqrt{2}, 2, h_{1:n-2}]$$

Thus, it is simply the silver schedule with the first two steps added again at the beginning. This choice, although somewhat arbitrary, was determined using the PEPit toolbox. First, it is important to check that the *modified* silver schedule performs similarly to the silver schedule when the gradient is exact. The results are shown in the following plot (with thin and thick lines having the same meaning as in Figure 3.4):

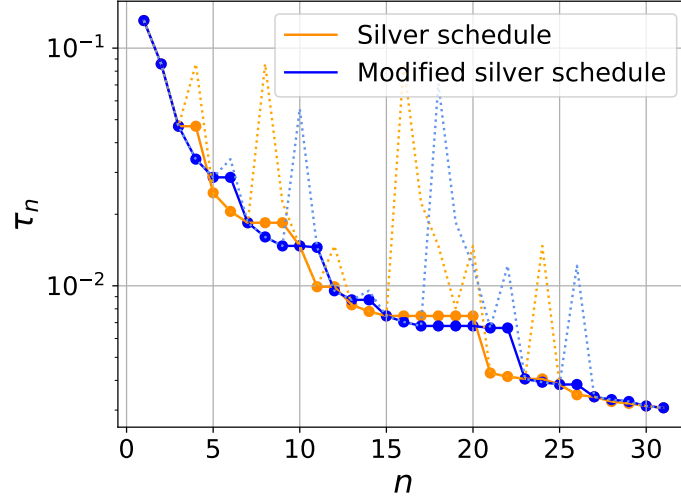


Figure 3.7: Comparison between the modified silver schedule gradient descent and the silver schedule gradient descent in the exact case. $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

We observe that the performance of both schedules is similar. The number of iterations n at which the modified silver gradient descent works best is $n = 2^k + 1$, $k \in \mathbb{N}$, which makes sense as these are the same as the silver schedule plus two, given the two additional terms in the modified silver schedule.

The rate of the modified silver schedule gradient descent compared to the silver schedule gradient descent and the classical gradient descent is shown in the following plot (for exact and inexact gradients with $\delta = 0.05$).

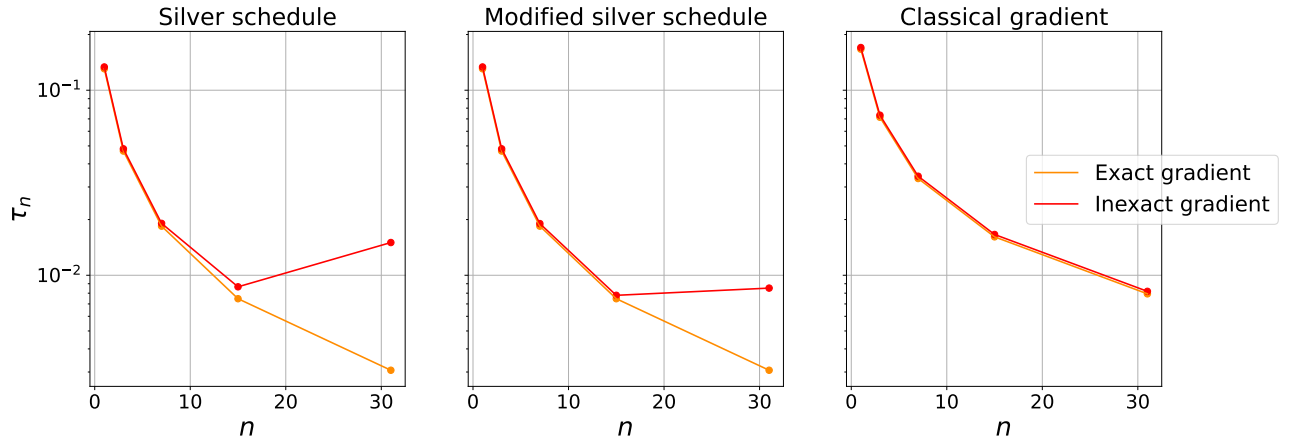


Figure 3.8: Comparison between the silver schedule inexact gradient descent, the modified silver schedule inexact gradient descent, and the classical inexact gradient depending on the number of iterations. ($\delta = 0.03$ and $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

Several phenomena can be observed from the plots above. Firstly, the classical gradient descent is significantly more robust to inexactness compared to both the

modified and original silver schedule gradient descents. The difference between the values of τ_n for exact and inexact gradients is small in the case of the classical gradient descent, indicating that the impact of inexactness is minimal. This is not the case for both the original and modified silver schedules. There is a noticeable difference in the value of τ_n when the gradient is exact versus inexact. However, the modified silver schedule is more robust to inexactness compared to the original silver schedule. Therefore, the modified schedule retains the same properties as the silver schedule in an exact gradient framework but performs better in the presence of inexact gradient. Hence, the modified silver schedule will be adopted as the main schedule for the remainder of this work.

One might wonder why the specific choice of adding $[\sqrt{2}, 2]$ was made for the modified silver schedule. Firstly, these values correspond to the first two steps of the silver schedule. It has been observed that adding any $2^k, k \in \mathbb{N}$ first terms of the schedule again at its beginning yields similar performance to the silver schedule. However, this specific choice of adding the first two elements of the schedule again at the beginning proved to be the best, particularly in terms of robustness to inexactness.

3.3.2 Modified Silver Schedule with Inexact Gradient

The modified silver schedule gradient descent appears to work well. However, there is still room for improvement. Like the original silver schedule, it eventually diverges when $\delta > 0$. The goal of this section is to analyze *when* this divergence occurs and how we can leverage this information to retain the benefits of the modified silver schedule while avoiding divergence due to gradient inexactness.

First, we will examine the rate τ_n for a fixed number of iterations n as a function of inexactness δ . This will be shown for three different values of n . Note that the first two graphs share the same x-scale, while the third does not (the setup remains $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$).

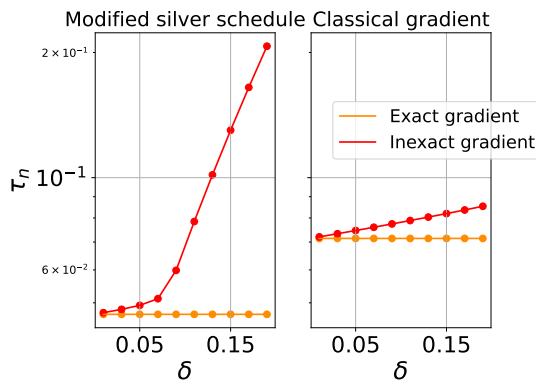


Figure 3.9: Value of τ_n depending on δ for fixed $n = 3$.

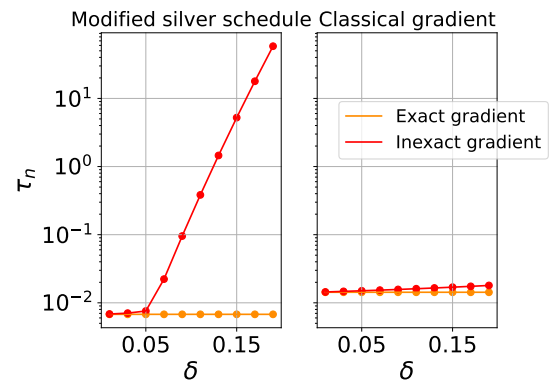


Figure 3.10: Value of τ_n depending on δ for fixed $n = 17$.

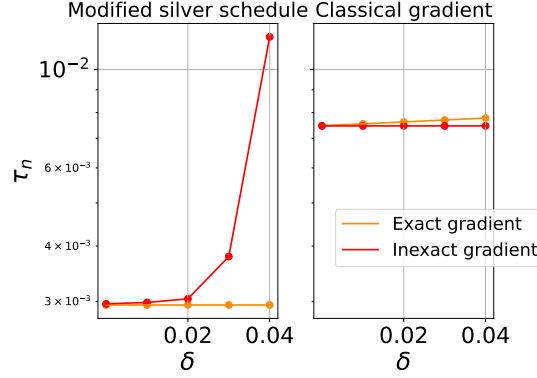


Figure 3.11: Value of τ_n depending on δ for fixed $n = 33$.

Two main observations can be made from these plots. First, the modified silver schedule fails for any n when $\delta > 0.1$. Additionally, if not at $\delta = 0.1$, the modified silver schedule gradient descent begins to diverge at approximately $\delta_{max} \approx \frac{1}{n}$. In the first plot, it diverges (performs worse than classical gradient descent) at $\delta_{max} = 0.1$, as the modified silver schedule cannot work beyond this value. In the second plot, it starts performing worse than classical gradient descent at $\delta_{max} \approx \frac{1}{17} \approx 0.06$, and this trend is again observed in the last plot. We can use these observations to develop an algorithm that leverages the positive aspects of the modified silver schedule while mitigating the impact of inexact gradient.

One coherent approach is outlined in Algorithm 5. This algorithm initially uses the modified silver schedule gradient descent for $i \leq n_{max} = 2^{\lfloor \log_2(\frac{3}{4\delta}) \rfloor} + 1$, based on the aforementioned observations. The factor $\frac{3}{4}$ is used to provide a *comfort zone*. Since the modified silver schedule tends to diverge rapidly when $n > n_{max}$, it is preferable to set n_{max} slightly smaller rather than larger. Therefore, the value is set to $\approx \frac{3}{4\delta}$ instead of $\frac{1}{\delta}$. For the modified silver schedule to work, it is necessary that $n_{max} = 2^{k_{max}} + 1$, where $k_{max} \in \mathbb{N}$. That's why n_{max} is defined as such. When $n > n_{max}$, the simple classical gradient descent will be applied. In conclusion, the inexact mixed silver schedule gradient descent algorithm uses the modified silver schedule up to the point where it is effective, and then switches to classical gradient descent when the modified silver schedule starts to fail due to inexactness. The results of this algorithm will be shown and compared with the next proposed algorithm.

Another algorithm can be considered. By recognizing that the *long* steps cause the modified silver schedule to blow up with inexact gradients, we can apply the modified silver schedule with *smaller* step sizes throughout the iterations. This approach leads to Algorithm 6. Recall that $\%$ denotes the modulo operator, where $a\%b$ is the remainder of the integer division of a by b .

This algorithm repeats the initial part of the modified silver schedule continuously. By doing so, it avoids the *longer* steps that cause divergence when the gradient is

Algorithm 5 Inexact Mixed Silver Schedule Gradient Descent

- 1: Given an L -smooth convex function f , a starting iterate x_0 and a relative inexactness $\delta \leq 0.1$
- 2: Set $n_{max} = 2^{\lceil \log_2(\frac{3}{4\delta}) \rceil} + 1$
- 3: Initialize the modified silver schedule $\tilde{h}$ with a length of n_{max} such as $\tilde{h}_{1:n_{max}+2} = [\sqrt{2}, 2, h_{1:n_{max}}]$ and $h_{1:n_{max}}$ is such that:

$$\begin{cases} h_1 &:= [\sqrt{2}] \\ h_{1:2n+1} &:= [h_{1:n}, 1 + \rho^{k-1}, h_{1:n}] \end{cases}$$

- 4: $i \leftarrow 0$
- 5: **while** $i < n_{max}$ **and** $i < \text{max_iter}$ **do**
- 6: Let d_i be an approximate gradient with δ relative inexactness, i.e.

$$\|d_i - \nabla f(x_i)\| \leq \delta \|\nabla f(x_i)\|$$

- 7: Compute the next iterate as $x_{i+1} = x_i - \frac{\tilde{h}_{i+1}}{L} d_i$
- 8: $i \leftarrow i + 1$
- 9: **end while**
- 10: **while** $i < \text{max_iter}$ **do**
- 11: Let d_i be an approximate gradient with δ relative inexactness, i.e.

$$\|d_i - \nabla f(x_i)\| \leq \delta \|\nabla f(x_i)\|$$

- 12: Compute the next iterate as $x_{i+1} = x_i - \frac{1}{L} d_i$
 - 13: $i \leftarrow i + 1$
 - 14: **end while**
-

Algorithm 6 Inexact Restart Silver Schedule Gradient Descent

- 1: Given an L -smooth convex function f , a starting iterate x_0 and a relative inexactness $\delta \leq 0.1$
- 2: Set $n_{max} = 2^{\lceil \log_2(\frac{3}{4\delta}) \rceil} + 1$
- 3: Initialize the modified silver schedule $\tilde{h}$ with a length of n_{max} such as $\tilde{h}_{1:n_{max}+2} = [\sqrt{2}, 2, h_{1:n_{max}}]$ and $h_{1:n_{max}}$ is such that:

$$\begin{cases} h_1 &:= [\sqrt{2}] \\ h_{1:2n+1} &:= [h_{1:n}, 1 + \rho^{k-1}, h_{1:n}] \end{cases}$$

- 4: $i \leftarrow 0$
- 5: **while** $i < \text{max_iter}$ **do**
- 6: Let d_i be an approximate gradient with δ relative inexactness, i.e.

$$\|d_i - \nabla f(x_i)\| \leq \delta \|\nabla f(x_i)\|$$

- 7: Compute the next iterate as $x_{i+1} = x_i - \frac{\tilde{h}_{(i+1)\%n_{max}}}{L} d_i$
 - 8: $i \leftarrow i + 1$
 - 9: **end while**
-

inexact.

The performance of these two algorithms, compared to classical gradient descent for different values of δ , is shown in Figures 3.12 and 3.13.

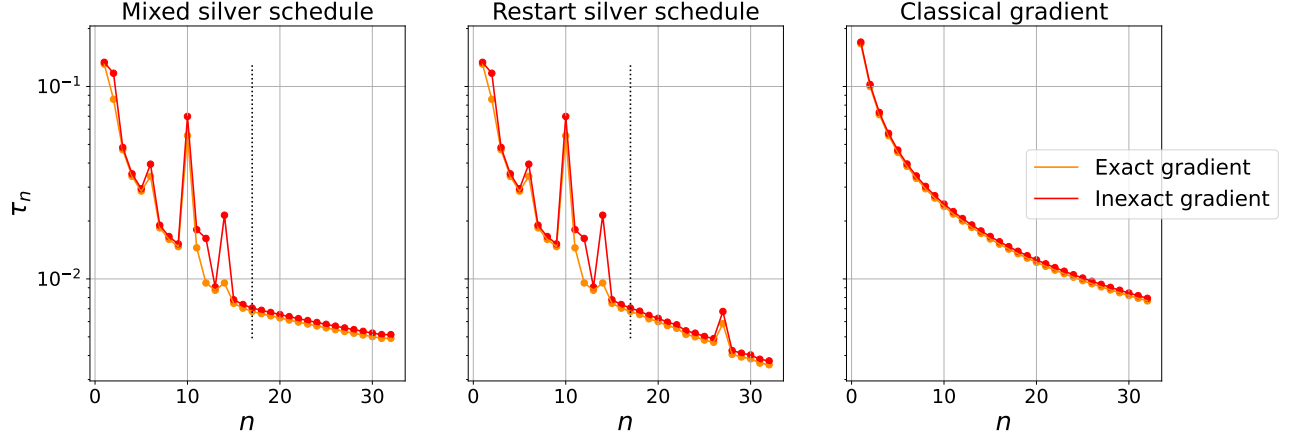


Figure 3.12: Comparison between the rates of the mixed silver schedule inexact gradient descent, the restart silver schedule inexact gradient descent and the classical inexact gradient descent depending on n . The vertical dotted lines represents the value of n_{max} . ($\delta = 0.03$ and $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

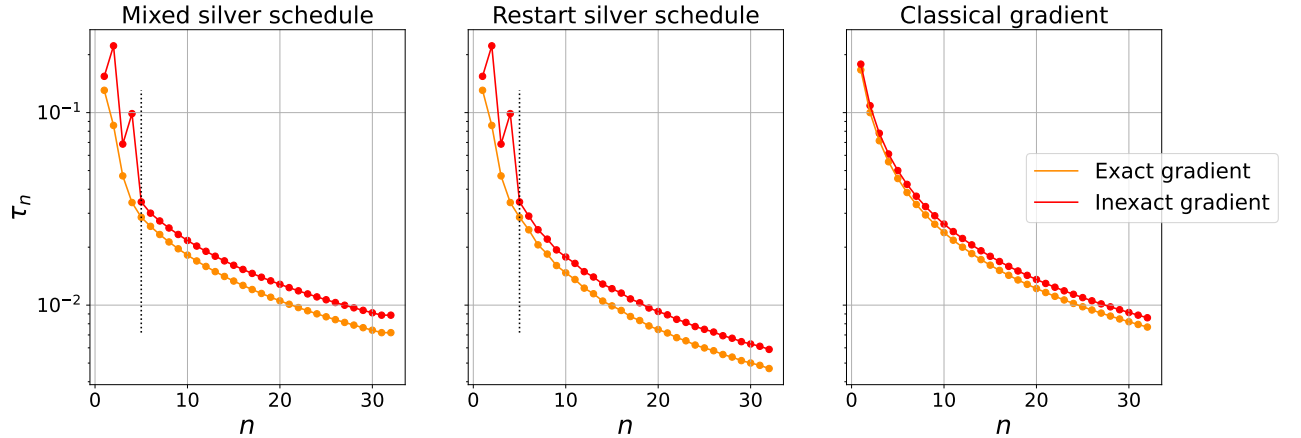


Figure 3.13: Comparison between the rates of the mixed silver schedule inexact gradient descent, the restart silver schedule inexact gradient descent and the classical inexact gradient descent depending on n . The vertical dotted lines represents the value of n_{max} . ($\delta = 0.1$ and $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

Both algorithms perform well, often surpassing the classical gradient descent when $\delta \leq 0.1$ (with similar performances for the mixed silver schedule when $\delta = 0.1$).

We can also examine the value of τ_n for these three algorithms, depending on δ for a fixed value of n . These results are shown in Figure 3.14.

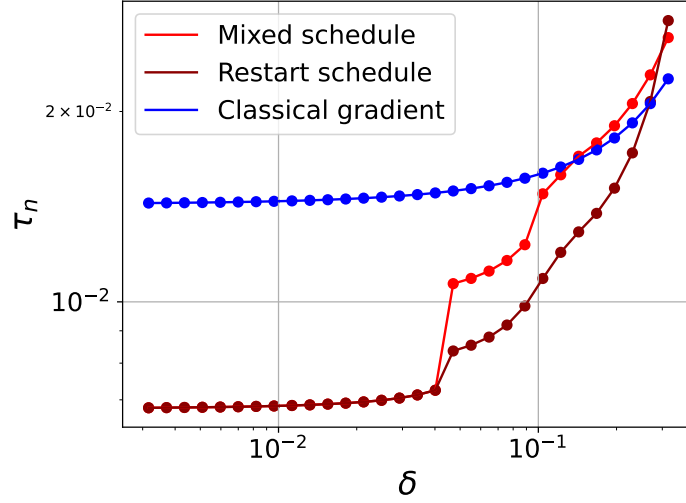


Figure 3.14: Comparison between the rates of the mixed silver schedule inexact gradient descent, the restart silver schedule inexact gradient descent and the classical inexact gradient descent depending on δ . ($n = 17$ and $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

It is evident that these algorithms outperform the classical gradient descent when $\delta \leq 0.1$. It means that we somehow managed to use the benefits of silver schedule but also made it work on inexact gradient ! It should be noted that both the mixed and restarted algorithms seem to have a convergence rate of order $\mathcal{O}(1/n)$, similar to gradient descent (while the silver schedule has a convergence rate of order $\mathcal{O}(1/n^{1.27})$). However, we have successfully tuned the constant factor that this rate is multiplied by. It is also interesting to compare these two new methods with the best constant step size for inexact gradients (as shown in Section 2.2, the step size $\frac{3}{2(1+\delta)}$ was a good choice for any n). The results for two different values of n are shown in Figure 3.15.

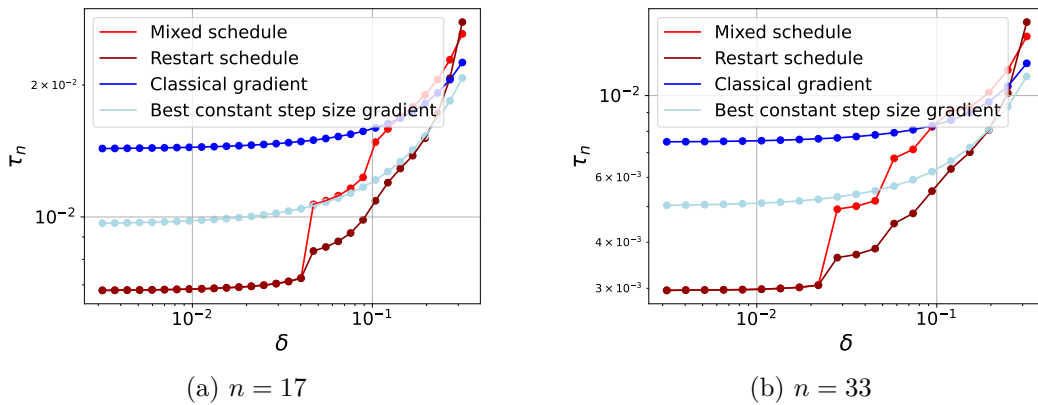


Figure 3.15: Rates of convergence of silver schedule gradient descent when the gradient is inexact as a function of number of iterations. ($\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

Both algorithms exhibit better results than the best constant step size gradient descent at some point, particularly up to $\delta = 0.1$ for the restart silver schedule.

It is also worth noting that these two new algorithms, especially the mixed one, could be further improved. For instance, in the mixed schedule, the best constant step size (see Section 2.2) could be used instead of the $1/L$ step size for the second part of the algorithm. However, this would require knowing the number of iterations in advance, which is often impractical. Algorithms that do not depend on a predefined number of iterations are generally more desirable. These minor adjustments could further enhance these algorithms. Thus, the benefits of the silver schedule can be harnessed even in the presence of inexact gradients, a result which was not initially guaranteed.

3.4 Conclusion on the Different Silver Schedules

In conclusion, accelerating first-order methods using fractal schedules with long steps is an innovative and promising approach in optimization. These methods have emerged recently, and thorough analysis is crucial before potentially applying them to real-world optimization problems. As a final illustration, we compare in Figure 3.16 the rate of convergence of the modified silver step size schedule exact gradient descent with the well-known accelerated gradient descent (introduced in [Nes83]). Recall that this algorithm is not memoryless, as it requires storing the previous iterates.

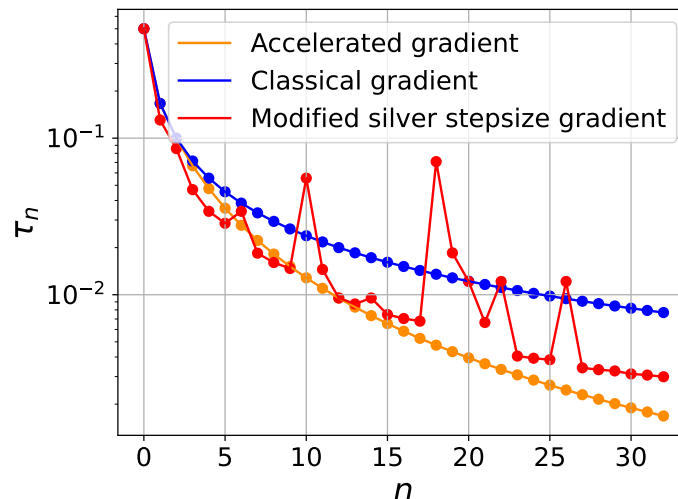


Figure 3.16: Rates of convergence of modified silver gradient descent compared with classical gradient descent and accelerated gradient descent, depending on n . $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

As expected, the modified silver schedule gradient descent performs better than

classical gradient descent but worse than accelerated gradient descent. For small values of n , the modified silver schedule may outperform accelerated gradient descent, but it is surpassed as n increases. Despite improving the rate of convergence to $\mathcal{O}(1/n^{1.27})$, the silver schedule has two significant drawbacks.

The first drawback is that this method only works for specific numbers of iterations ($2^k - 1$, where $k \in \mathbb{N}$, for the classical silver schedule). Additionally, as the number of iterations increases, the distance between these specific iterates grows exponentially (distance between iterates = 2^k , where $k \in \mathbb{N}$). This challenge can be softened by storing the current best iterate in memory while continuing to iterate. However, this approach involves additional storage costs, unlike the constant step size gradient method.

The second major drawback of the silver schedule method is its poor performance with inexact gradient. This issue was discussed in Section 3.3.2, and some remedies were proposed to address it. The algorithms proposed in this section however only work for *small* values of δ (when $\delta \leq 0.1$).

In conclusion, fractal schedules with long steps (such as the recently proposed schedule in [GSW24]) are promising. Careful analysis is crucial for better understanding first-order optimization algorithms. These accelerating algorithms could significantly impact the field, given the widespread daily use of optimization algorithms. This research area is thriving, and the Performance Estimation Framework is a vital tool for analyzing and understanding these algorithms, especially in cases of inexact gradients, which are common in many applications today (see Section 1.3).

4 Comparison of Algorithms

This last section aims at comparing the different algorithms discussed in this work, both from a theoretical worst-case convergence point of view (Section 4.1) and on a practical numerical example (Section 4.2). It will allow us to see which algorithms work better, theoretically and practically, both when the gradient is exact and when it is not.

4.1 Theoretical Comparison

In this section, we provide a comprehensive comparison of the theoretical results of various optimization algorithms discussed throughout this work. This comparison is crucial for understanding the performance trade-offs and guiding practitioners in selecting the most appropriate algorithm for different scenarios. We'll first look again at the rate of convergence of those different algorithms, depending on the number of iterations and with different levels of inexactness. Here is a reminder of the different algorithms discussed in this work.

- Exact Gradient Descent with Best Step Size (h_{max} being the solution of the non-linear equation $\frac{1}{2nh_{max}+1} = (1 - h_{max})^{2n}$).
- Inexact Gradient Descent with Best Step Size ($h_{max} = \frac{3}{2(1+\delta)} \forall n$).
- Silver Step Size Schedule (introduced in [AP23]).
- Modified Silver Step Size Schedule (the Silver Step Size schedule with two extra well-chosen steps at the beginning).
- Mixed Silver Step Size Schedule (the Modified Silver until possible and then Classical Gradient).
- Restart Silver Step Size Schedule (the Modified Silver until possible and then restart the same pattern).

We should note that the setup used for the rates of convergence is different between Section 2 ($\frac{1}{L}\|\nabla f(x_n)\|^2 \leq \sigma_n(f(x_0) - f(x_*))$) and Section 3 ($(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n\|x_0 - x_*\|^2)$). This is not a major problem, as the first one can be transformed into the second one simply dividing its rate by 4 (already discussed in Section 2.1). This statement hasn't been proved, but is strongly supported by numerical evidence. We remind that, mathematically, it means:

$$\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2 \iff \|\nabla f(x_n)\|^2 \leq 4\tau_n (f(x_0) - f(x_*)).$$

All rates from Section 2 can thus be transformed using the above relation to evaluate all algorithms on an equal footing.

The plots of the rate of convergence of the different algorithms, working on the class of smooth convex functions, depending on the number of iterations n will be made for different values of δ .

Exact gradient ($\delta = 0$): In this case, the mixed and restart silver schedules make no sense, as they are precisely defined when $\delta > 0$. The other algorithms will still be compared.

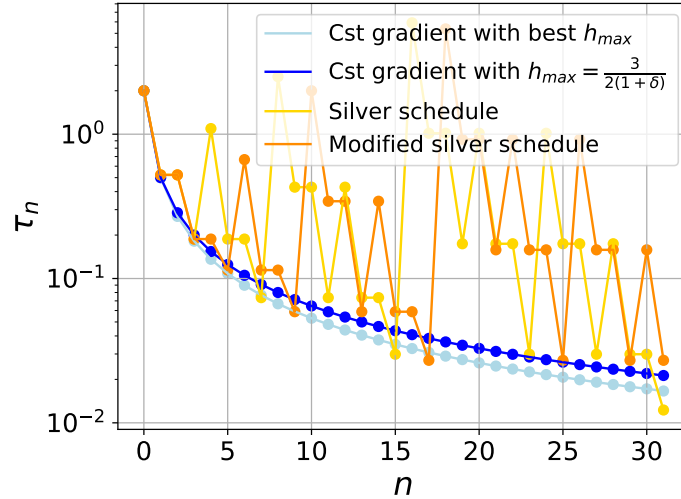


Figure 4.1: Comparison between rates of all discussed methods, depending on the number of iterations, for exact gradient ($\delta = 0$). ($\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

In the exact case, the silver schedules (both initial and modified ones) allows us to minimize smooth convex functions with a rate of order $\mathcal{O}\left(\frac{1}{n^{1.27}}\right)$, which is a big advantage in the long run. This cannot really be seen on Figure 4.1, as the number of iterations represented is a bit small, but for a very large value n , the advantage of the silver schedules gradient over the constant gradient descents will become significant (thanks to the better order of convergence!). However, the silver schedules can only achieve this rate for specific number of iterations.

The constant gradient with best h_{max} obviously works better than the other constant step size gradient descent, as it specifically uses the best possible step size depending on the number of iterations to perform. However, it requires knowing in advance the number of performed iterations, which can be a drawback in some

practical applications. The two constant rates are of order $\mathcal{O}\left(\frac{1}{n}\right)$, which is worse than the silver schedules.

In the exact case, if choosing a number of iterations working fine for one of the silver schedules (i.e. $2^k \pm 1, k \in \mathbb{N}$ depending on the silver schedule) is not a problem, then these methods should be preferred. If this is not the case, the constant step size methods should be preferred, with a preference for the one with the best h_{max} , if we are sure that the number of iterations performed will be at least equal to the number for which the h_{max} has been calculated, and the other in the other case.

Inexact gradient ($\delta = 0.05$): Both the mixed silver schedule and the restart silver schedule can now also be considered.

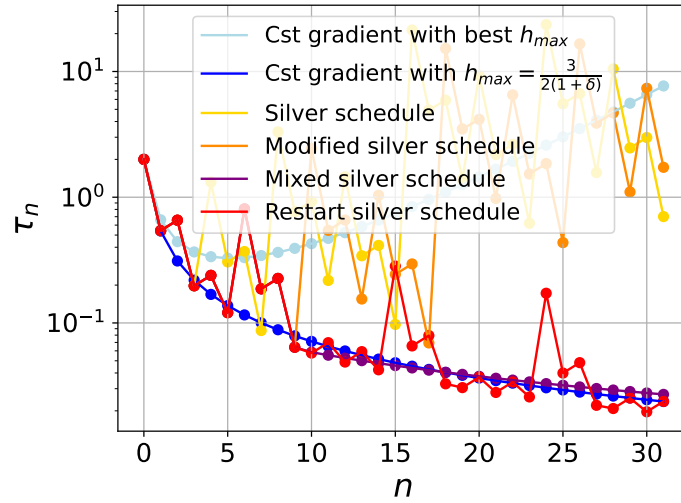


Figure 4.2: Comparison between rates of all discussed methods, depending on the number of iterations, for inexact gradient ($\delta = 0.05$). $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

In this case, with *small* inexactness (although 5% can already be considered significant), we can directly see that three methods no longer work, namely the constant gradient with best h_{max} , the silver schedule and the modified silver schedule. However. Those methods will thus no longer be analyzed for greater inexactness, as they will only get worse.

The three other methods seem to have the same kind of performances, with the restart silver schedule having slightly better performances than the other ones. However, this method has the general drawback of silver methods, as it only achieves its best performances at fixed number of iterations.

Inexact gradient ($\delta = 0.1$): The results are displayed in Figure 4.3 for the three remaining algorithms, namely the constant gradient with $h_{max} = \frac{3}{2(1+\delta)}$, the mixed silver schedule and the restart silver schedule.

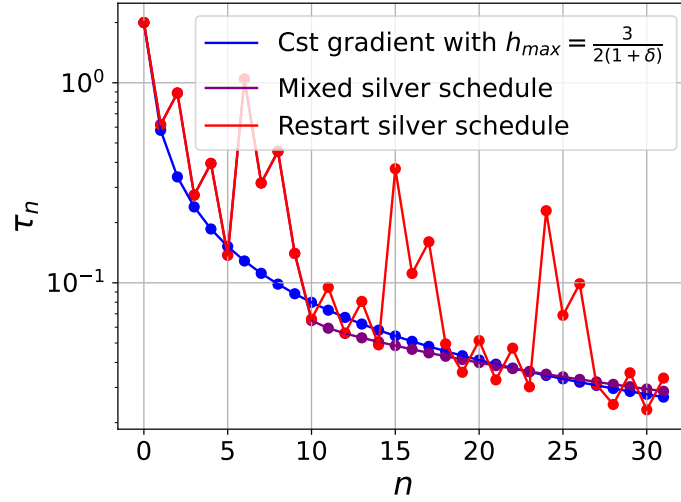


Figure 4.3: Comparison between rates of all discussed methods, depending on the number of iterations, for inexact gradient ($\delta = 0.1$). $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

In this case, no real difference can be observed compared to the case with $\delta = 0.05$, except that the restart schedule becomes even worse for its bad number of iterations, which doesn't have a huge impact on the analysis, after all.

To the analysis from the previous paragraph, we can also add the fact that in the long run, the mixed silver schedule gradient descent becomes worse than the constant gradient descent. This is because the constant step size used at the end of the algorithm is simply $h = \frac{1}{L}$, a size which could be improved, using the same length as for the constant gradient method (i.e. $h = \frac{3}{2L(1+\delta)}$). The creation of the mixed silver schedule gradient descent assumes unknown the new rates proven in Section 2.2. Using the innovations from both Sections 2.2 and 3.3.2 could lead to even better algorithms.

Inexact gradient ($\delta = 0.5$): Let's finally look at those methods when the gradient is *very* inexact. Note that 50% of relative error is huge, as it represents half of the initial gradient in norm.

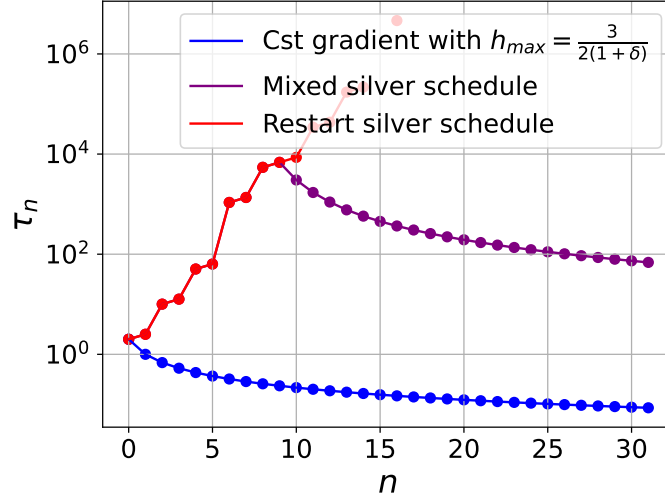


Figure 4.4: Comparison between rates of all discussed methods, depending on the number of iterations, for inexact gradient ($\delta = 0.5$). $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

With a big inexactness, even the silver methods tuned to work in the inexact framework doesn't work anymore. However, the constant gradient method with the safe choice of $h_{max} = \frac{3}{2(1+\delta)}$ still seems to work, even if the error on the gradient is as big as $\delta = 0.5$. This method is thus very robust to inexactness and seems to be the best choice in the case where $\delta \gg 0$.

The final theoretical comparison of all those algorithms is summarized in Table 1. Note that the order of the rates of convergence of the two last methods in the table are conjectured, and has not been proven.

As a conclusion, we can say that if the gradient is exact and that we can store the current best iterate, then silver methods are probably the most promising choices. If no storing is possible, then the gradient method with the best possible step size is the best one, provided that the number of performed iterations is at least as big as the predicted one.

If the inexactness is small enough ($\delta \leq 0.1$), then mixed and restart silver schedules may be considered. It will give better results than constant gradient, but the method will only work well for several numbers of iterations. On the other hand, a lot more numbers of iterations will be *good* values, compared to classical silver schedule.

If the inexactness is too big, or if we lack information on the size of this inexactness, then the constant step size gradient descent with $h_{max} = \frac{3}{2(1+\delta)}$ is the best choice. It's also important to note that it's a better idea to overestimate the inexactness than to underestimate it.

Algorithm	Rate of Convergence	Robustness to Inexactness	Iteration Requirement and Flexibility
Gradient Descent with Best Step Size	$\mathcal{O}(\frac{1}{n})$ with the best constant term	Low	Requires a lower bound on the number of iterations
Gradient Descent with $h_{max} = \frac{3}{2(1+\delta)}$	$\mathcal{O}(\frac{1}{n})$	High	Always works
Silver Step Size Schedule	$\mathcal{O}(\frac{1}{n^{1.27}})$	Low	Only works for specific number of iterations
Modified Silver Step Size Schedule	$\mathcal{O}(\frac{1}{n^{1.27}})$	Low	Only works for specific number of iterations
Mixed Silver Step Size Schedule	$\mathcal{O}(\frac{1}{n})$	Medium (when $\delta \leq 0.1$)	Only works for specific number of iterations (when $n < n_{max}$)
Restart Silver Step Size Schedule	$\mathcal{O}(\frac{1}{n})$ with the scd best constant term	Medium (when $\delta \leq 0.1$)	Only works for specific number of iterations

Table 1: Comparison of Optimization Algorithms

4.2 Experiment Analysis and Application to Linear Regression

All the algorithms have been analyzed, both numerically and/or theoretically, from a worst-case point of view. In practice, the algorithms may perform better than what the worst theoretical guarantees predicted. This last section aims at applying them to a common practical optimization problem, namely a linear regression.

The linear regression is a fundamental statistical method used to model the relationship between a dependent variable y and one or more independent variables x . The goal is to find the best-fitting linear equation:

$$y = mx + b,$$

where:

- y is the dependent variable.
- x is the independent variable.

- m is the slope of the line.
- b is the y-intercept.

The objective of linear regression is to find the optimal parameters m and b that minimize the discrepancy between the observed values and the values predicted by the linear model. This discrepancy is measured using the mean squared error (MSE), which serves as the objective function in the optimization problem:

$$J(b, m) = \frac{1}{P} \sum_{i=1}^P (y_i - (mx_i + b))^2 \quad (4.1)$$

Here, P is the number of data points, and (x_i, y_i) are the observed data points.

Note that this is not the worst-case function, which was a Huber function for all algorithms.

The optimization problem for linear regression can be formulated as follows:

$$\min_{b, m} J(b, m) = \min_{b, m} \frac{1}{P} \sum_{i=1}^P (y_i - (mx_i + b))^2$$

Note that this problem only contains two variables and could be solved explicitly, but that it would require solving a linear system. That's why gradient descent is sometimes preferred compared to the algebraic method.

In the experiment⁹, 50 randomly generated points were used to perform the linear regression (i.e. $P = 50$). This example is relevant for several reasons:

- Linear regression is widely used in various fields such as economics, engineering, and social sciences to understand relationships between variables and make predictions.
- The experiment demonstrates the practical application of the different algorithms in this paper in finding the optimal parameters for the linear regression model.
- By applying the algorithms to a practical problem, we can observe how well they perform compared to their worst-case theoretical guarantees. This helps in understanding their robustness and efficiency in real-world scenarios.
- The experiment also includes an exploration of the impact of inexact gradient, providing insights into the behaviour of the methods under non-ideal conditions.

⁹The initial experiment with constant step size gradient descent without inexactness comes from: <https://github.com/mattnedrich/GradientDescentExample>

This application of linear regression exemplifies the practical utility of optimization algorithms beyond their theoretical analysis. It bridges the gap between abstract algorithmic performance and real-world problem-solving, highlighting the strengths and limitations of the algorithms when applied to a common and essential problem.

The inexactness was handled in a way that comes close to the worst-case analysis, as the chosen direction was such that the added direction is a *bad* one. Indeed, the (scaled) direction towards the optimal solution of the problem was added to the gradient. Surprisingly, it appeared that such a *bad* direction was the one pointing towards the optimal solution¹⁰. Obviously, this direction is chosen such that $\|d_k - \nabla f(x_k)\|^2 = \delta^2 \|\nabla f(x_k)\|^2$. This direction can be visualized in Figure 4.6, considering the univariate objective function $f(x) = x^2$ and the current iterate $x_i = 4$.

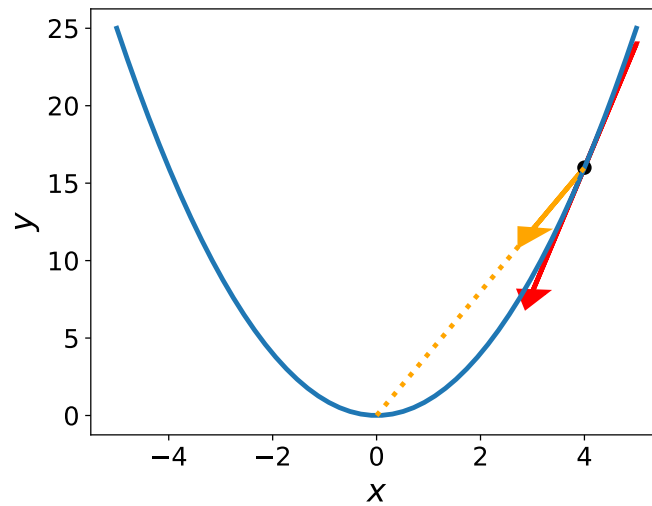


Figure 4.5: Visualisation of the direction added to the gradient. The objective function $f(\cdot)$ is in blue, the descent direction $-\nabla f(x_i)$ is in red and the added direction is in orange.

Let's first look at the rate τ_1 of one iteration of the gradient descent, depending on the step size h , for both exact and inexact gradient descents. Note that the smoothness constant L is not explicitly known for this experiment, and that the step size h here is the actual size used, meaning that it should not be scaled by a factor $1/L$.

¹⁰It surprisingly led to worse results (what we want, for a bad direction) as if the direction was chosen towards the opposite of the optimal solution, when $\delta \leq 0.5$!

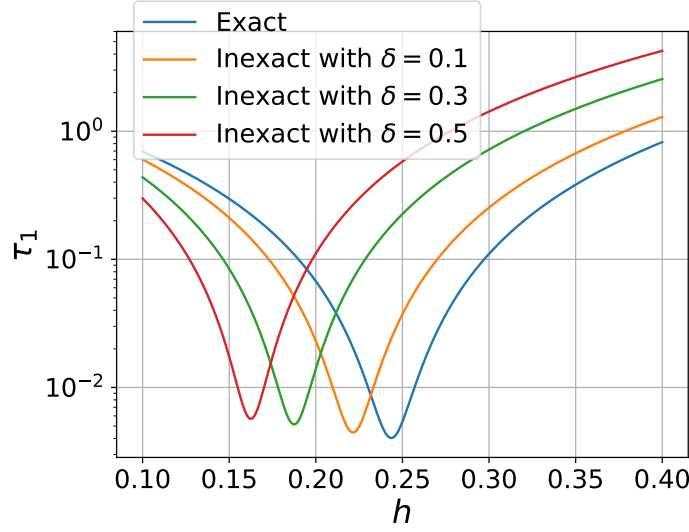


Figure 4.6: Rate τ_1 of one step of exact and inexact gradient descents on the experiment, depending on the step size h . ($\frac{1}{L}(f(x_1) - f_*) \leq \tau_1 \|x_0 - x_*\|^2$)

We can first see that, unlike expected theoretically, the curve at the left of the optimum h seems to grow fast in each case. It probably means that the gradient descent worked better than expected for the optimal step size, but that the range of values in which the step sizes work really well for this method is not that large. Indeed, choosing a twice smaller step size leads to a value of τ_1 that has deteriorated by several orders of magnitude, which was clearly not the case theoretically (recall Figure 2.1, for example).

Then, for the particular inexact direction d_0 used, we can see that the bigger the inexactness is, the smaller the step size should be, as expected theoretically. Furthermore, the ratio between the best step size in the exact case and in the inexact ones is very close to $1 + \delta$, which is also what the theory said (recall that the $h_{max} = \frac{3}{2L}$ in the exact case and $\frac{3}{2L(1+\delta)}$ in the inexact one, for one iteration of gradient descent). However, in terms of τ_1 , the inexact gradient methods performed a bit worse than expected, at least compared to the exact one. Recall indeed that the theoretically expected order of τ_1 is $\mathcal{O}(\frac{1}{2hL+1})$ in the exact case and $\mathcal{O}(\frac{1}{2hL(1-\delta)+1})$ in the inexact one for the regime on the left of h_{max} (hL is written here as usually the actual step size is $\frac{h}{L}$). Taking into account the factor $1 + \delta$ between the optimal step sizes in exact and inexact frameworks, the ratio between the values of τ_1 should give $\frac{\frac{2h_{max}L(1-\delta)}{1+\delta} + 1}{2h_{max}L+1} \approx 0.78$ in this case, for $\delta = 0.5$ and $h_{max} \approx 0.24$. In practice, we can observe that this ratio is around 0.7 for this level of inexactness. It strengthens the fact that the exact constant step size gradient method seems to perform (a lot) better than expected in the exact case.

Another thing we can look at is the optimal h (the one with the smallest corresponding rate τ_n) depending on the number of iterations, for both exact and

inexact gradient. Theoretically, it was shown that this value should grow with n in the exact case, but that if the gradient is inexact, then choosing $h = \frac{3}{2(1+\delta)}$ for any n is a prudent choice.

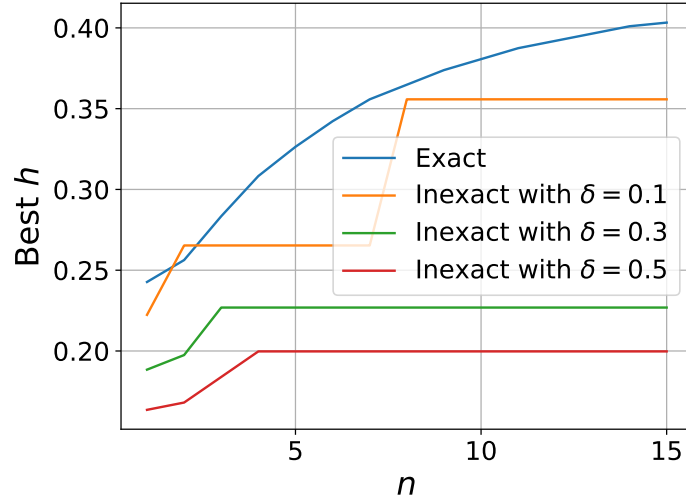


Figure 4.7: Best choice of constant step size h for each method on the experiment, depending on the number of iterations n .

In the exact case, as expected, this value grows with h . For a small inexactness ($\delta = 0.1$), the same behaviour seems to occur. However, for larger sizes of inexactness, the best step size seems to not change a lot with n , as expected theoretically. However, if a greater number of iterations is performed, at some point, the best step size will also grow for inexact gradients. This result is however not that strange. Indeed, recall that the value $h = \frac{3}{2(1+\delta)}$ was also chosen as the *intermediate* regime was not very well understood (see Figures 2.6 and 2.8). Better step sizes may exist, but the chosen one is a *prudent* choice. However, showing the results of this simple experiment for large values of n doesn't bring information, as the methods converges very quickly towards the optimal solution.

Modified silver step size schedule: Let's now apply the modified silver schedule gradient descent to this experiment. To do so, the constant L has to be determined. We will recall the results from the introduction and use Theorem 1.1, which says that constant L bounds by above the larger eigenvalue of the Hessian of the objective function. Doing so for this experiment leads to $L \approx 4.1$

Now that we have this approximated value for L , we can try the modified silver gradient descent with and without inexactness on this experiment. The results are displayed in Figure 4.8.

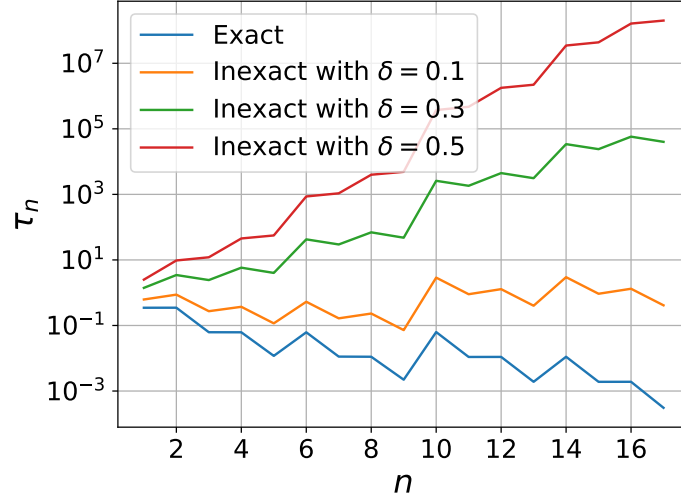


Figure 4.8: Rate τ_n of exact and inexact modified silver schedule gradient descents on the experiment, depending on the number of iterations n . ($\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

As expected theoretically, the modified silver schedule doesn't work when there is inexactness. However, with small inexactness, it seems that the silver schedule gradient descent works a bit better than expected theoretically. However, we will still only focus on the exact silver step size schedule gradient descent and compare it to constant step size gradient descents.

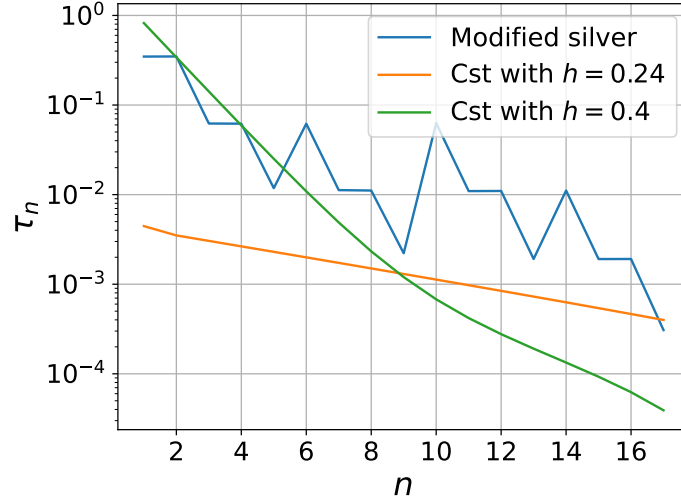


Figure 4.9: Comparison of rates τ_n of constant step size gradient descent (both with $h = 0.24$ and $h = 0.4$) and modified silver schedule gradient descent, depending on the number of iterations n . ($\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

We can see that if the right step size is chosen (i.e. the best step size depending on the number of iterations, which is $h = 0.4$ for the number of performed iterations),

then the constant step size gradient descent works better than the modified silver one. It's not because the modified silver gradient descent doesn't work properly, in fact it works pretty well on this experiment, but because the constant step size gradient descent works particularly well here. However, we can still see that the silver method works better than constant step size gradient descent if the chose step size is the best one for a single iteration of the method ($h = 0.24$ here), demonstrating that the method is still a good one on this problem.

We can further use the extra information we have about this problem: the objective function is quadratic (see Equation (4.1)). Knowing this, we can improve the modified silver schedule to try and minimize the function even faster. Indeed, for quadratic function gradient-based minimization, the step sizes $\frac{2}{L}$ are useless as they only make the gradient descent go from one side of the quadratic, to the other. Let's thus remove all the step sizes of lengths 2 in the silver schedule to see if the gradient method is well accelerated.

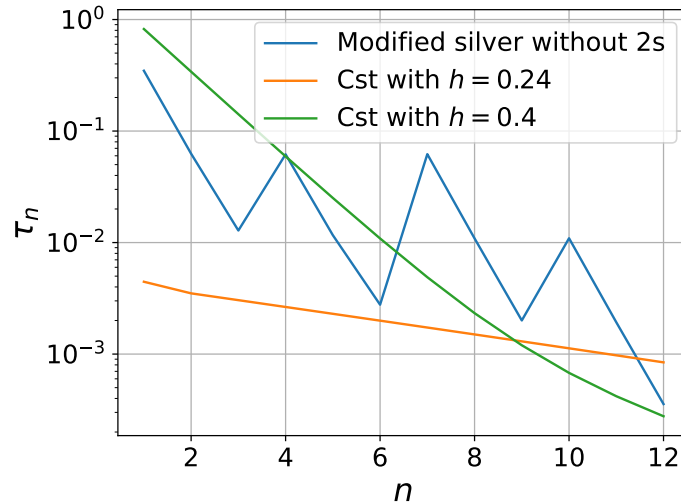


Figure 4.10: Comparison of rates τ_n of constant step size gradient descent (both with $h = 0.24$ and $h = 0.4$) and *modified silver schedule without 2s* gradient descent, depending on the number of iterations n . $(\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2)$

We can indeed see that now the method is nearly as fast as the best constant step size gradient method. This little trick here shows once again that any additional information on the objective function can (and should) be used to our advantage to improve the method.

Mixed and Restart Silver Schedules: To end this numerical experiment, let's finally look at the behaviour of the mixed and restart silver schedules when applied to this numerical experiment, with $\delta = 0.05$. Those algorithms will be compared to the constant step size gradient descent with best step size, taking into account the inexactness, i.e. $h = 0.37$. The results can be observed on Figure 4.11.

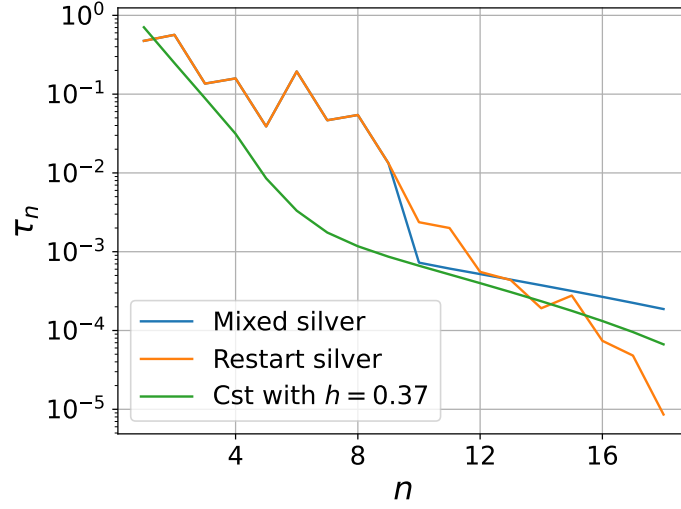


Figure 4.11: Comparison of rates τ_n of constant step size inexact gradient descent with best step size, mixed silver gradient descent and restart silver gradient descent, depending on the number of iterations n . ($\delta = 0.05$ and $\frac{1}{L}(f(x_n) - f_*) \leq \tau_n \|x_0 - x_*\|^2$)

The results of this experiment are very encouraging ! They tell us that the restart silver schedule gradient descent works better than the best possible constant step size inexact gradient descent, even on the experiment. This shows the potential of silver schedule ([AP23]) and its variants (introduced in this work), even in the inexact framework.

The conclusion of this experiment when dealing with inexact gradient is the same as in the theoretical section. When the inexactness is small ($\delta \leq 0.1$), then the restart silver schedule gradient descent should be used, while when the inexactness is greater, the constant step size gradient descent remain the most robust approach.

5 Conclusion

This work began with an introduction to optimization and key concepts, such as the Performance Estimation framework, and the concept of gradient with relative inexactness. These foundational ideas set the stage for the analysis of first-order optimization methods.

We then conducted a theoretical analysis on the constant step size gradient method, both in the exact and inexact frameworks. This helped us understand how those methods work and how the knowledge on the inexactness can be leveraged to optimize the objective function effectively.

Next, we analyzed the recently proposed silver step size schedule gradient descent from a numerical perspective. This led to the development of new algorithms designed to better handle inexact gradient descent.

Finally, a comprehensive comparison of the different algorithms was performed, both theoretically and on the practical example of a linear regression.

Take-away ideas.

First, the Performance Estimation framework is a very powerful tool to find and prove the rate of convergence of first-order optimization methods, sometimes able to find better (tighter) rates of convergence than with classical proofs.

The idea of designing methods to handle gradient with relative inexactness is very interesting, as some problems suffer from inherent inexactness, and methods being able to still solve them thus make perfect sense.

In this context of inexact gradient, the main thing to remember considering constant step size gradient methods is that using *shorter* step sizes compared to the exact case is a good idea. Furthermore, a good choice of step size doesn't increase with the number of performed iterations, unlike the exact case.

Then, the recently proposed silver-based gradient methods should be more and more considered, as they theoretically outperform constant step size gradient methods, and also work well on a practical experiment. They however have fewer guarantees when the gradient is inexact, but the algorithms newly proposed in this work seem to handle it, for *small* inexactness.

The exploration and the analysis performed in this work significantly contribute to the understanding of first-order optimization methods under exact and inexact gradients. By providing theoretical results on the convergence of those methods

on smooth convex functions suffering from inexact gradient information, this work bridges a crucial gap in optimization literature. It shows in particular that adapting the step sizes of the methods to the nature of gradient information leads to better convergence results and the potential of new methods, such as the silver schedule gradient method.

This work not only offers new perspectives on existing methods, but also lays the ground for future research. Some results from this work suggest that there is room for improvement and innovation in optimization algorithms, especially in improving algorithms handling inexactness.

Future work.

First, some conjectured results in this work, for example the order of the rates of convergence of the mixed and restart silver schedules gradient descents, would benefit from rigorous mathematical proofs.

Then, designing methods combining different sections of this work could lead to more powerful algorithms. For example, a mixed silver schedule gradient descent using the best possible constant step size, given known inexactness, would probably lead to a better mixed method.

Conducting more numerical experiments across various practical scenarios would also give more insight into the performances of the different methods from a practical point of view. While this work focused on *worst-case* scenarios, some methods may perform better *on average* despite having worse worst-case convergence rates.

Finally, an extension of this work to other class of functions, such as *non-convex* or *composite non-smooth* functions, would also offer a better understanding of these methods' performances across different problem types.

In summary, the innovation brought by this thesis could lead to more development, to finally get a better general understanding of first-order optimization methods.

Bibliography

- [AP23] Jason M. Altschuler and Pablo A. Parrilo. Acceleration by stepsize hedging ii: Silver stepsize schedule for smooth convex optimization, 2023.
- [BV04] Stephen P Boyd and Lieven Vandenbergh. *Convex optimization*. Cambridge university press, 2004.
- [d’A08] Alexandre d’Aspremont. Smooth optimization with approximate gradient. *SIAM Journal on Optimization*, 19(3):1171–1183, 2008.
- [DGN14] Olivier Devolder, François Glineur, and Yurii Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Mathematical Programming*, 146:37–75, 2014.
- [DKGT20] Etienne De Klerk, François Glineur, and Adrien B Taylor. Worst-case convergence analysis of inexact gradient and newton methods through semidefinite programming performance estimation. *SIAM Journal on Optimization*, 30(3):2053–2082, 2020.
- [DT14] Yoel Drori and Marc Teboulle. Performance of first-order methods for smooth convex minimization: a novel approach. *Mathematical Programming*, 145(1-2):451–482, 2014.
- [GG23] François Glineur and Geovani Nunes Grapiglia. Course linma2471: Optimization models and methods 2, 2023. Course given at UCLouvain.
- [GMG⁺22] Baptiste Goujaud, Céline Moucer, François Glineur, Julien Hendrickx, Adrien Taylor, and Aymeric Dieuleveut. Pepit: computer-assisted worst-case analyses of first-order optimization methods in python. *arXiv preprint arXiv:2201.04040*, 2022.
- [GSW24] Benjamin Grimmer, Kevin Shu, and Alex L. Wang. Accelerated objective gap and gradient norm convergence for gradient descent via long steps, 2024.
- [KOPR23] Jaeyeon Kim, Asuman Ozdaglar, Chanwoo Park, and Ernest K. Ryu. Time-reversed dissipation induces duality between minimizing gradient norm and function value, 2023.
- [Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Doklady Akademii Nauk SSSR*, 269(3):543, 1983.
- [Nes03] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

- [Tay17] Adrien B Taylor. *Convex interpolation and performance estimation of first-order methods for convex optimization*. PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2017.
- [THG17] Adrien B Taylor, Julien M Hendrickx, and François Glineur. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161:307–345, 2017.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl