UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Ivo Fagundes David de Oliveira

Limits And Improvements On Searching And Optimization: From One Dimensional Problems To Multi-objective Optimization

Belo Horizonte
2021

Ivo Fagundes David de Oliveira

Limits And Improvements On Searching And Optimization: From One Dimensional Problems To Multi-objective Optimization

Versão final

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Ricardo Hiroshi Caldeira Takahashi

Belo Horizonte
2021

R484l

Oliveira, Ivo Fagundes David de.
Limits and improvements on searching and optimization [recurso eletrônico]: from one dimensional problems to multi-objective optimization/ Ivo Fagundes David de Oliveira. - 2021.
1 recurso online (125 f. : il., color.) : pdf.

Orientador: Ricardo Hiroshi Caldeira Takahashi.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 115-120.

Exigências do sistema: Adobe Acrobat Reader.

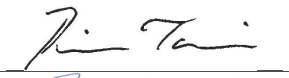
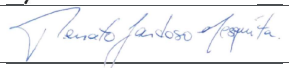
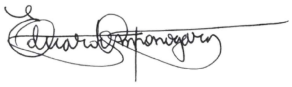
1. Engenharia elétrica - Teses. 2. Otimização matemática – Teses. 3. Otimização Multiobjetivo – Teses. I. Takahashi, Ricardo Hiroshi Caldeira. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pela Bibliotecária Letícia Alves Vieira - CRB-6/2337
Biblioteca Prof. Mário Werneck - Escola de Engenharia da UFMG

ATA DA 387ª DEFESA DE TESE DE DOUTORADO
DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ATA DE DEFESA DE TESE DE DOUTORADO do aluno **Ivo Fagundes David de Oliveira** - registro de matrícula de número 2018752175. Às 13:00 horas do dia 06 do mês de dezembro de 2021, reuniu-se na Escola de Engenharia da UFMG a Comissão Examinadora da TESE DE DOUTORADO para julgar, em exame final, o trabalho intitulado **"Limits and Improvements on Searching and Optimization: From One Dimensional Problems to Multi-objective Optimization"** da Área de Concentração em Sistemas de Computação e Telecomunicações. O Prof. Ricardo Hiroshi Caldeira Takahashi, orientador do aluno, abriu a sessão apresentando os membros da Comissão e, dando continuidade aos trabalhos, informou aos presentes que, de acordo com o Regulamento do Programa no seu Art. 8.16, será considerado APROVADO na defesa da Tese de Doutorado o candidato que obtiver a aprovação unânime dos membros da Comissão Examinadora. Em seguida deu início à apresentação do trabalho pelo Candidato. Ao final da apresentação seguiu-se a arguição do candidato pelos examinadores. Logo após o término da arguição a Comissão Examinadora se reuniu, sem a presença do Candidato e do público, e elegeu o Prof. Ricardo H. C. Takahashi para presidir a fase de avaliação do trabalho, constituída de deliberação individual de APROVAÇÃO ou de REPROVAÇÃO e expedição do resultado final. As deliberações individuais de cada membro da Comissão Examinadora foram as seguintes:

Membro da Comissão Examinadora	Instituição de Origem	Deliberação	Assinatura
Prof. Dr. Ricardo Hiroshi Caldeira Takahashi - Orientador	DMAT (UFMG)	aprovado	
Prof. Dr. Renato Cardoso Mesquita	DEE (UFMG)	aprovado	
Prof. Dr. Alexandre Salles da Cunha	DCC (UFMG)	aprovado	
Prof. Dr. Alexandre Cláudio Botazzo Delbem	Dep. de Sistemas de Computação (USP-SC)	aprovado	
Prof. Dr. Eduardo Camponogara	Dep. de Automação e Sistemas (UFSC)	aprovado	

Tendo como base as deliberações dos membros da Comissão Examinadora a Tese de Doutorado foi aprovada. O resultado final de aprovação foi comunicado publicamente ao Candidato pelo Presidente da Comissão, ressaltando que a obtenção do Grau de Doutor em ENGENHARIA ELÉTRICA fica condicionada à entrega do TEXTO FINAL da Tese de Doutorado.O Candidato terá um prazo máximo de 30 (trinta) dias, a partir desta data, para fazer as CORREÇÕES DE FORMA e entregar o texto final da Tese de Doutorado na secretaria do PPGEE/UFMG. As correções de forma exigidas pelos membros da Comissão Examinadora deverão ser registradas em um exemplar do texto da Tese de Doutorado, cuja verificação ficará sob a responsabilidade do Presidente da Banca Examinadora. Nada mais havendo a tratar o Presidente encerrou a reunião e lavrou a presente ATA, que será assinada pelo Presidente da Comissão Examinadora. Belo Horizonte, 06 de dezembro de 2021.

ASSINATURA DO PRESIDENTE DA COMISSÃO EXAMINADORA

Acknowledgments

I would first like to express my sincere gratitude to professor Takahashi whose guidance, support and encouragement has been invaluable throughout the doctorate program. I thank my parents for their wisdom and continuing effort to help their son reach this landmark. And, I wish to thank my wife Anna for her faithful aid, reassurance and love that has helped me go on with joy. Finally, I thank God for this wonderful privilege of being here and for all the insights that I fully attribute to Him.

Affiliations This thesis was written when the first author I.F.D. Oliveira (ivodavid@gmail.com) was a Ph.D. student in the the doctorate program of the Electrical Engineering Department of the Federal University of Minas Gerais. The first author also holds an affiliation to the Institute of Science, Engineering and Technology of the Federal University of the Valleys of Jequitinhonha and Mucuri as an assistant professor; address: R. Cruzeiro, 1, Teófilo Otoni, Minas Gerais, Brazil, 39803-371. Professor R.H.C. Takahashi (taka@mat.ufmg.br) is a full professor of both the Mathematics Department and the Electrical Engineering Department of the Federal University of Minas Gerais; address Av. Pres. Antônio Carlos, 6627, Belo Horizonte, Minas Gerais, Brazil, 31270-901.

Funding This work was supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – Brazil.

Summary. This thesis presents a series of improvements on four different classical searching methods employed for solving different well established problems. The methods improved on and their corresponding problems are: (i) the bisection method for continuous root-searching problems; (ii) the binary search algorithm for discrete list-searching; (iii) the back-tracking technique for inexact Armijo-type searching; and (iv) the n-dimensional steepest descent method for non-linear multi-objective optimization. Different types of improvements are aimed for in each context that produce an overall reduction in the the number of calls to the external function being searched. However, all four improvements proposed have one thing in common: the worst-case upper-bound of our methods either outperform the state-of-the-art, or, where the state-of-the-art has already attained an optimal worst-case performance, we match the performance of the optimal bound while improving on either average performance, asymptotic performance or both. Thus, in this sense, the methods we propose are *strict* improvements on classical solutions, attained with no additional assumptions on the problems considered nor with any additional costs other than the computation of the methods themselves. The manuscript starts with a broad introduction which discusses the importance of the problems considered and the classical solutions employed in several different fields. The main contributions are given in the following four chapters; one corresponding to each problem tackled. Each chapter corresponds to one published (or soon to be published) result intimately related to the four problems considered which are augmented with original unpublished material. Finally, in the sixth and final chapter we point to possible ramifications of the findings hereby delineated which present potential for future developments.

Keywords: binary searching, root searching, line searching, backtracking list searching, gradient method, multi-objective optimization

Resumo. Esta tese apresenta uma série de melhorias em quatro métodos clássicos de busca empregados para resolver quatro problemas bem estabelecidos. Os métodos aprimorados e seus problemas correspondentes são: (i) o método da bissecção para problemas de busca de raízes; (ii) o algoritmo de busca binária para procura em listas discretas; (iii) a técnica de back-tracking para buscas inexatas do tipo Armijo; e (iv) o método de otimização utilizando a direção de maior descida para problemas multi-objetivo. Diferentes tipos de melhorias são produzidas em cada instância que, de forma geral, produzem uma redução no número de chamadas à função externa que está sendo procurada. No entanto, todas as quatro melhorias propostas têm uma coisa em comum: as garantias de pior caso dos nossos métodos sempre apresentam uma melhoria em relação ao estado da arte e, quando o estado da arte já apresenta um desempenho de pior caso ótimo, então, os nossos métodos apresentam um desempenho médio ou desempenho assintótico aprimorados em relação ao estado da arte. Neste sentido, os métodos que propomos são melhorias estritas sobre as soluções clássicas, obtidas sem suposições adicionais sobre os problemas considerados e nem com custos adicionais escondidos. O manuscrito começa com uma ampla introdução que discute a importância dos problemas considerados e as soluções clássicas empregadas em vários campos diferentes. As principais contribuições são dadas no quatro capítulos subsequentes. Cada capítulo corresponde a uma resultado publicado (ou em vias de ser publicado) com a adição de material exclusivo à tese íntimamente relacionados com os quatro problemas considerados. No sexto e último capítulo, apontamos as possíveis ramificações das descobertas aqui delineadas, que apresentam potencial para desenvolvimentos futuros.

Palavras-chave: busca binária, busca de raíz, busca em linha, backtracking, busca em lista, método do gradiente, otimização multi-objetivo

Contents

Introduction

Searching an ordered list is one of the most fundamental tasks in computer science (Knuth, 1998). List searching can be found anywhere from simple algorithms to more complex computational systems, often implicit, or entailed within subroutines of parent algorithms. In a like manner, within the literature of numerical analysis and optimization, the task of searching for a root of a continuous function $f : \mathbb{R} \to \mathbb{R}$ (or an extremum) seems to play a similar fundamental role (Nesterov, 2018). In both settings, the discrete and the continuous counterpart, the standard approach in the literature has been to employ *binary searching*, also referred to as the *bisection method* within the numerical analysis field. Binary searching has been the method of choice due to it's minmax optimality (Sikorski, 1982), i.e. it requires the least amount of queries under worst case conditions; where, in the first setting, consulting one entry of the list makes for one query and in the second setting one query is the computation of the function value (or derivative) on one point. It is due to it's minmax performance in different settings (Sikorski, 1982; Knuth, 1998; Vieira, Takahashi, & Saldanha, 2012; Nesterov, 2018), that binary searching is widely regarded as an optimal procedure, and, is often a first alternative when considering different approaches to solve searching problems.

More complex tasks in computer science, numerical analysis and optimization that go beyond one dimensional problems are instead often described as multi-dimensional optimization problems. Refer to Examples 1.1.1 to 1.1.3 in (Nesterov, 2018) for illustrations on how different broadly encompassing problems (continuous and discrete) can be described in this manner. These more complex tasks are often first framed as a single objective optimization problem with objective function defined from $\mathbb{R}^n$ to $\mathbb{R}$ and then, more generally, as the complexity of the searching problem requirements grows, these may be augmented as searching problems for Pareto-efficient solutions of multi-objective optimization problems with objective functions defined from $\mathbb{R}^n$ to $\mathbb{R}^m$; where $n, m \in \mathbb{N}$. And, similar to how binary searching strategies occupy a central role in single-dimensional searching problems, descent direction strategies (and in particular those derived from the steepest-descent direction) play a fundamental role in the development of more efficient searching routines for multi-variable problems. And thus, in this way both binary searching as well as steepest descent type methods have served, to the very least, as building blocks for

the development of more complex algorithms, and, improvements on such methods can amount to significant time savings across applications and in different fields.

In this manuscript we consider four well known and extensively studied searching problems and we propose improvements to the long-lasting solutions that have played this fundamental *building block* role that binary searching and steepest descent type methods have played. Some such problems tackled in this thesis are generally considered to be solved; that is, the standard and widely known solutions to such problems are broadly perceived as optimal procedures to which no improvement can be made; and, the binary searching strategy employed in list searching is certainly a prime example of this. The purpose of this thesis is primarily to provide long overseen improvements to such methods that, somewhat surprisingly, can amount to significantly large time-savings (as demonstrated throughout). The four problems we consider, which include list searching and root searching, can be represented in the following form:

$$\text{Find } x^* \in \mathcal{S} \text{ such that } v(x^*) = 0. \tag{1.1}$$

For (i) the list searching problem $\mathcal{S}$ is a set of indices $\{0, 1, 2, ..., n\}$ and v is a sorted list L where $v(k) = L_k$ for k assuming any value from 0 to n. For (ii) the root-searching problem and (iii) the inexact Armijo searching problem $\mathcal{S}$ is an interval $[a, b] \subset \mathbb{R}$ and v is a continuous function $v(x) = f(x)$ from the interval $[a, b]$ to $\mathbb{R}$. And finally, for (iv) the multi-objective optimization problem with objective function $f : \mathbb{R}^n \to \mathbb{R}^m$, the set $\mathcal{S}$ is $\mathbb{R}^n$ and v is a continuous functional which takes f and a point in $\mathbb{R}^n$ and maps to some measure of proximity to critical conditions on $\mathbb{R}_+$. And, to a greater or lesser degree, the standard solutions to (i)-(iv) are closely related to either binary searching strategies or steepest descent strategies; both of which are discussed ahead within the following Sections.

The reason binary strategies take such a central role in many algorithms is discussed in greater detail in Section 1.1, however, the main idea is that binary strategies make an efficient use of bit-wise information of the type "a solution lies within this sub-domain" which then typically produce optimal worst case guarantees. Also, however to a lesser degree, steepest descent type methods for multi-variable minimization problems have also been shown to attain optimal worst-case guarantees for some broad classes of problems, albeit, the importance of steepest descent type methods is perhaps not due so much to it's worst case guarantees but rather due to it's simplicity when compared to other methods and it's widespread use as a benchmark of performance over-which other methods can improve. Hence, any approach that aims on improving such methods must also ensure that, to the very least, the worst-case upper-bounds produced by the "new improved method" will match those of the standard solutions and, somewhat subjectively, have a simple intuitive form over-which other methods can build upon. Here we propose precisely this, methods in which the worst-case upper-bounds either outperform the state-of-the-art, or, match it when the state-of-the-art has already attained an optimal worst-case performance. And when worst-case performance is tied, we aim at providing results that guarantee improvements on either the average performance, the asymptotic performance or both, with strategies that can be easily built-on to serve as fundamental building blocks for more complex computational tools.

To do this the first step is to characterize the known minmax results for the query complexity of searching strategies.

Thesis organization

In this thesis, as we deal with four different problems, we summarize minmax characterization results that concern one dimensional problems into overarching terms in the following section entitled *On binary searching*, and, the results concerning minmax guarantees of steepest-descent type methods for multi-variate optimization problems in the final section of this chapter entitled *On steepest-descent searching*. Since the former section introduces common concepts present in the problems addressed in Chapters 2 to 4, the definitions and propositions are exposed with less recourse to formality to allow for more broad encompassing and introductory concepts that are only formally and fully fleshed out in the respective chapters being introduced. This section also includes an intuitive toy example to illustrate the techniques used in identifying and constructing the ITP method in Chapters 2 and 3, as well as a subsection discussing alternative competing strategies that have often been evoked as possible substitutes to the canonical binary search strategy. In the latter section *On steepest-descent searching* we point to the known minmax results on gradient descent type methods in the literature as well as open questions in the field. Hopefully, this introductory chapter substantiates the scope of the doctorate thesis here contained, and, provides an overarching context for the following chapters.

The remainder of the thesis is structured as follows:

In Chapters 2 and 3 we describe and analyse the root-searching and the list searching problems respectively. There we flesh out the minmax results alluded to in Section 1.1 and find, somewhat surprisingly, that there almost always exists a rich class of methods that, similar to binary searching strategies, attain minmax optimality; i.e. binary searching is not *uniquely* minmax optimal. We evaluate both the average performance and the asymptotic performance of the methods within the class of minmax methods, for the respective problems, and find that binary searching happens to attain the worst possible average performance; and thus seems to be a poor choice when compared to other methods within the class. We then identify and recommend one particular minmax strategy, which we name the *ITP method*, and show that in the discrete list searching setting it attains an optimal average performance and in the continuous root-searching setting it attains an optimal asymptotic performance, both of which come at no cost on the minmax performance of the bisection method. Hence, differently from previous improvements, we find the first viable substitute to the bisection method that comes with no cost other than the computation of the method itself. In Chapter 4 we describe and analyse the inexact searching problem introduced by Armijo (Armijo, 1966) and the standard back-tracking solution employed in the literature. There we begin by characterizing the performance guarantees that can be ensured by standard back-tracking practices often used in inexact line-searching and often present as a key component of descent direction algorithms. And then, we show that these widespread practices lack in terms of worst-case performance when compared to optimal lower-bounds derived under standard assumptions on the line-searching problem. And, finally, we propose a method that attains the minmax optimal lower-bound on the number of calls to the objective function, lowering the query complexity by a logarithmic factor. The method proposed is termed *the geometric bisection method* and, when combined with the ITP method proposed in the previous chapters the proposed method additionally produces multi-logarithmic speed-up in terms of asymptotic performance. In Chapter 5 we describe and analyse the multi-objective optimiza-

tion problem. There we begin by proposing a new descent direction, which we name the *central descent direction*, and a measure of proximity to critical conditions that contain many advantages over the classical definitions used in the literature. The improved descent direction is first shown to satisfy a series of robustness guarantees which cannot be ensured by the classical *Cauchy*-definition of steepest descent, and then, the measures of proximity are shown to have well understood geometric interpretations which make them natural choices for first-order descent direction type methods. Finally, we propose a new incremental-based method to approach critical conditions with reduced computational cost. This is done by combining the newly proposed descent direction with an *anchored* incremental strategy, which is then shown to produce critical points at a reduced computational cost. Finally, in Chapter 6 we discuss our findings and point to potential directions of research.

1.1 On binary searching

Binary searching is a divide and conquer technique (Knuth, 1998) that starts by (i) dividing the domain of solutions $\mathcal{S}$ in two parts $\mathcal{H}$ and $\mathcal{H}^C$ of equal size, and then (ii) it verifies (or queries) whether a solution x^* is located in one of the parts. These two operations are performed alternately and selectively until an output $\hat{x}$ can be found. A satisfying output $\hat{x}$ can usually be pinpointed when S is sufficiently small (e.g. when S is left with only one element then $\hat{x} = x^*$ is easily obtained), and, under continuous settings it is common to accept solutions $\hat{x}$ such that $|\hat{x} - x^*|_\dagger$ is sufficiently small, where $|\cdot|_\dagger$ is some given metric on the elements of S. Thus (iii) a verification of this condition is added to the iterative process to terminate the search. Combining these we can represent the classical binary search procedure with the following diagram:

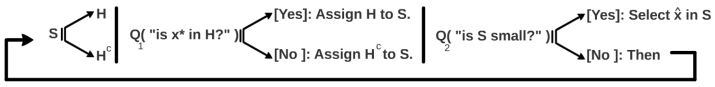


Fig. 1.1: Schematic diagram of the binary search strategy. Here the binary search algorithm is displayed as a while-loop, however, under different settings for-loops or other structures can be used to exploit computational architecture (Cannizzo, 2018).

When S takes the form of a sorted list, the division operation splits the list in two halves, the query Q_1 takes the form of "is $v(x_{1/2}) \leq 0$?" where $x_{1/2}$ is the index of the middle term of list L, and the query Q_2 takes the form of "is the number of terms in L less than or equal to 1?". And, of course, the assignment operations are done virtually by keeping track of the indices of the remaining list. Furthermore, it is easy to verify that the root-searching version of binary searching follows an analogous structure, and, a slightly more careful analysis of the Armijo back-tracking procedure reveals that it too follows a similar pattern (as is extensively discussed in Chapter 4).

This construction finds applications in several different fields, including single and multi-objective optimization, ranking and learning problems, game theory and applications, artificial intelligence amongst others (Yao & Yao, 1976; Sikorski, 1982; Knuth, 1998; Luenberger & Ye, 2018; Nesterov, 2018). However, the one dimensional versions, i.e. when applied to one dimensional problems of the form (1.1), are perhaps the most wide-spread. When dealing with the continuous setting, i.e. root-searching problems (Sikorski, 1982), it is called the bisection method; and, when dealing with discrete domains, i.e. of searching lists (Yao & Yao, 1976), it is called binary searching. Chapter 2 deals with the first type and and Chapter 3 with the second type. Both of these one dimensional versions play a key role in the construction of more complex algorithms. And, since many multidimensional problems are often broken into several one-dimensional sub-problems (Vieira et al., 2012; Nesterov, 2018), one dimensional binary searching serves as a fundamental building block for more complex algorithms.

As mentioned above, binary search is often the method of choice due to it's minmax performance under several different settings. Minmax optimality, which is a context-dependent property, is typically a consequence of the following context-independent feature of binary searching:

Proposition 1. *Given an initial domain S_0 of size $|S_0|_\dagger$, after k iterations of the binary search algorithm, the remaining domain S_k of possible solutions is of size less than or equal to $|S_0|_\dagger/2^k$.*

Proposition 1 guarantees that binary search performed with metric $|\cdot|_\dagger$ will produce a sequence $\mathcal{S}_k$ such that $|S_k|_\dagger$ converges to zero. And, under several different settings, which include the classical root-searching and list searching problems here considered, this is sufficient to guarantee minmax optimality. In the following, we define ν, when considering root searching problems, as $\nu \equiv \Delta/2\epsilon$ where ϵ is the target precision and $\Delta \equiv b - a$; and, when considering list searching ν is defined as $\nu \equiv n$.

Proposition 2. *Given a searching task in the form (1.1), binary searching isolates a solution $\hat{x}$ with no more than $n_{1/2} \equiv \lceil \log_2 \nu \rceil$ queries to $v(x)$. And, no other method can provide the same guarantees with less than $n_{1/2}$ queries.*

Analogous guarantees as those found in Propositions 1 and 2 are often the motivation behind binary searching strategies. Theorem 1.1 of Section 2.1 and Equation (3) of Section 2.2 are examples of these results correspondingly tailored for the searching problems considered in this thesis. Refer to (Oliveira, 2013) for an example of these features being exploited for attaining efficiency under alternative metrics. These are however metric dependent; i.e. different metrics $|\cdot|_\dagger$ will produce correspondingly different guarantees. And, perhaps more importantly, it is necessary that the measure implied by the query "is $\mathcal{S}$ small?" and the metric used in the "division of the domain of solutions $\mathcal{S}$ into two parts $\mathcal{H}$ and $\mathcal{H}^C$ of equal size" must be the same for these guarantees to hold. As will be seen in Chapter 4, for the inexact Armijo searching problem it seems that the literature has overseen this fact relying on a division and a verification operation on slightly different metrics, and, with a proper pairing of such metrics our results in Theorem 12 show that much reduction in computational cost is attainable. There are, however, settings when such

a pairing of metrics might not even be possible, and, this can potentially reduce the global efficiency of the search.

Perhaps a good illustration of the importance of the pairing of the metric with the division operation is obtained by considering a simple searching problem defined on a two dimensional unit disk $\mathcal{S} \equiv \{(x, y) \in \mathbb{R}^2 \text{ st. } x^2 + y^2 \leq 1\}$. If the goal of the search is to produce a sufficiently small subspace of solutions $\mathcal{S}$ such that the L_2 distance d between each pair of solutions (x_1, y_1) and (x_2, y_2) in $\mathcal{S}$ satisfies $d \equiv \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq \epsilon$ for some given $\epsilon > 0$, then, we would expect a consistent implementation of the classical binary searching strategy would require that each query subdivide the domain of solutions $\mathcal{S}$ in producing two subsets $\mathcal{H}$ and $\mathcal{H}^C$ with half the measure of the original set. The maximum L_2 distance between two points on the unit disc is of $d = 2$, however, it is not difficult to see that the unit disc $\mathcal{S}$ cannot be subdivided into two parts while simultaneously reducing in half the maximum L_2 distance between the points in each part $\mathcal{H}$ and $\mathcal{H}^C$. Thus, for problems posed with this type of issue, the identification of the metrics are as crucial as the the development of an efficient searching algorithm to accompany it. The last problem we consider, the multi-objective optimization problem, our contributions are precisely of this nature: we first identify a metric that properly measures proximity to desirable solutions and then we propose an efficient method to accompany such recommendation.

Is it possible to outperform binary searching?

We illustrate the use of Proposition 1 and possible improvements on binary searching by analysing the following well known riddle: "A boy thinks of an integer number between 1 and 100. How many questions must be asked to find his number given that he will answer yes or no to any question?". The classical answer to this riddle is obtained by applying binary search by dividing the numbers in two groups of equal size successively. With one question you will eliminate 50 numbers, in the second 25 and so on and so forth. This way we find the answer "at most seven questions are needed" since $\lceil \log_2 100 \rceil = 7$. Consider now the best-case performance of binary searching on the above mentioned riddle: the best case is attained when the sequence of sets are of sizes 50, 25, 12, 6, 3 and finally 1. Hence, binary search requires at least 6 iterations. Is there any method that also requires no more than 7 questions while attaining a best case performance of less than or equal to 5? If so, then this alternative method can be seen as a strict improvement on binary search, since it outperforms binary searching with respect to best case performance while performing no worse than binary searching with respect to minmax performance.

A careful analysis will show that the answer to the above question is positive. And, it gives an illustration of alternatives different than binary search that also enjoy minmax optimality[1], i.e. binary searching is not alone. Our improvements identified in Chapters 2 and 3 build on this simple intuition to find strategies that

[1] One solution is to start by subdividing the set in two groups of size 64 and 36. If the number is in the larger set you proceed with binary search and terminate in no more than 7 steps, and, if in the smaller set you subdivide it into two sets of size 32 and 4. Again, if it is located in the larger set you proceed with binary search and otherwise you finish in at most two more questions; i.e., a best case of 4 questions in total with no more than 7 in the worst case.

enjoy better-than-binary-search performances under different metrics while conserving minmax optimality. Before we proceed to our description of steepest descent methods, we discuss current alternatives to binary searching that are widely known and available in the literature in the following.

Alternatives to binary searching

Several alternatives to binary searching are known in the literature, including linear and grid searching (Knuth, 1998; Luenberger & Ye, 2018), interpolation strategies (Yao & Yao, 1976; Perl & Reingold, 1977; Perl, Itai, & Avni, 1978; Argyros & Khattri, 2013), and hybrid approaches (Brent, 1971; Bus & Dekker, 1975; Ridders, 1979; Novak & Ritter, 1993; Novak, Ritter, & Woźniakowski, 1995); all of which fall short in terms of minmax optimality when compared to binary searching. However, these are typically associated with alternative metrics of performance as will be discussed ahead.

Linear and grid searching strategies are, simply put, a brute force approach to searching. In discrete settings it is defined by a sequential read of all the entries of the list until the list is either exhausted or the target value is found. When the given list is unsorted or if it is sorted by frequency rather than the target feature, then no better strategy can be followed. In the continuous setting, linear searching is done on a finite grid over the space of alternatives, hence it is referred to as grid search. For lists of size n, the worst case performance of linear searching is of n queries, and, if the target value can assume any value on the list with equal probability, then the expected query complexity is of n/2. However, if these probabilities are not uniform, and the list is sorted by frequency in which the target value is chosen, then the expected query complexity is of $p_1 + 2p_2 + ... + np_n$, where p_k is the probability that the target value be found in the k'th entry of the list. Under extremely skewed distributions the expected query complexity of linear search will outperform binary search, however, at a cost of an exponentially worst minmax performance. Thus, this strategy only seems to be justified for small lists, lists sorted by frequency or lists with unknown structure.

Interpolation based strategies are perhaps better illustrated under continuous settings. Assuming function $f : [a, b] \to \mathbb{R}$ is sufficiently smooth, interpolation error bounds guarantee tighter approximations to the root the higher the degree of the interpolation, and, the smaller interval $[a, b]$ is. This can then be exploited to produce sequential approximations to the root that converge at orders of convergence above one, and hence, the query complexity of interpolation strategies are of the order of $\log\log 1/\epsilon$ where ϵ is the target precision. For this reason, under the conditions of tight interpolation bounds, fast convergence to the root may be guaranteed. Classical examples of interpolation based strategies include the secant method and Newton/quasi-Newton methods. In the discrete setting, under the assumption that the list was generated by sorting n independent random samples from a uniform distribution, linear interpolation also turns out to achieve an expected query complexity of the order of $\log\log n$ which is known to be optimal on the average (Yao & Yao, 1976). Hence, in either setting, interpolation based strategies achieve exponentially faster convergence to a solution when compared to binary searching. However, this also comes at the cost of an exponentially worse minmax performance since interpolation search may reduce to grid searching under misspecified conditions, i.e.

when the list is not generated by sorted samples of a uniform distribution or when the function does not provide tight interpolation bounds at initialization.

Hybrid methods attempt to alleviate the trade-off of interpolation strategies typically by alternating between binary and interpolation steps. Good examples of this can be found in (Brent, 1971; Bus & Dekker, 1975; Ridders, 1979; Novak et al., 1995). State of the art solvers have often opted for hybrid approaches since they achieve orders of convergence only slightly less than unconstrained interpolation approaches. And, if at least one binary step is taken every k steps, they can guarantee that the method will take no more than k times the number of iterations that binary search would require overall. Nevertheless, binary searching will always find instances where interpolation is not helpful, and hence, even hybrid approaches may come at a high cost under non-smooth conditions, or, in the discrete setting, when the list is not generated by sorting samples from a uniform distribution. The natural goal that arises in the construction of hybrid methods is, of course, to minimize the trade-off between the benefits of interpolation and the benefits of binary searching. What are the fundamental trade-offs between minmax and alternative metrics (average or asymptotic)? Can high levels of asymptotic or average performance be achieved with little or no trade-off on minmax performance?

The answer to the above question is, once again, positive. And, in the following chapter we will show that almost always no trade-off of minmax performance is needed to achieve high orders of convergence and/or high average performance. And, the ITP method described therein is one way to achieve this.

1.2 On steepest-descent searching

Descent direction methods are iterative approximation techniques that attempt to produce near optimal solutions to single or multi objective optimization problems (Knuth, 1998; Boyd & Vandenberghe, 2009; Nesterov, 2018). They start with an initial estimate $\boldsymbol{x}$ in $\mathcal{S}$ and iteratively (i) compute a descent direction $\boldsymbol{d}$, i.e. a direction in which $f(\boldsymbol{x} + \alpha\boldsymbol{d})$ is locally decreasing with respect to $\alpha > 0$, and (ii) calculate a step-size α producing a new estimate $\boldsymbol{x}_{\text{new}} \leftarrow \boldsymbol{x}_{\text{old}} + \alpha\boldsymbol{d}$ such that $f(\boldsymbol{x}_{\text{new}}) < f(\boldsymbol{x}_{\text{old}})$. For single objective first order methods, the computation of the descent direction $\boldsymbol{d}$ typically involves the querying of the gradient $\nabla f(\cdot)$ of the objective function on the current estimate $\boldsymbol{x}$; and, the construction of the step-size $\alpha > 0$ typically involves the implementation of a line-search technique which queries the function value over multiple points on the line $\boldsymbol{x} + \alpha\boldsymbol{d}$. These two operations are performed alternately and selectively until an output $\hat{x}$ can be found, and, the overall computational cost is the combination of all gradient and function values queried throughout the process, as well as any other higher dimensional derivative computed by the method of choice. Here, a satisfying output $\hat{x}$ can usually be pinpointed when some pre-defined measure of proximity to optimal conditions is sufficiently small (e.g. when $\nabla f(\boldsymbol{x})$ is less than some pre-specified tolerance $\epsilon > 0$). Thus (iii) a verification of this condition is added to the iterative process to terminate the search. Combining these we can represent the classical descent direction searching methods with the following diagram:

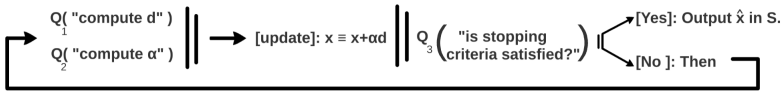


Fig. 1.2: Schematic diagram of the descent direction algorithms. Here Q_1 to Q_3 represent the key computations in the algorithm where a query is typically performed. In Q_1, first order derivatives or derivatives of higher order are computed in order to construct a descent direction, in Q_2 a line search is performed which typically involves querying the function value on multiple points in order to find a local minima over the line or simply to find a point that satisfies some sufficient decrease condition, and in Q_3 the data collected in Q_1 and Q_2 are typically evaluated to estimate whether satisfiable proximity to a solution has been reached.

For single objective optimization, when $\boldsymbol{d}$ is defined as $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$ then the descent direction method is called the steepest descent method or simply the gradient descent method, irrespective of how α or the stopping criteria are defined (Armijo, 1966; Fliege & Svaiter, 2000). For multi objective optimization, the steepest descent direction defined in (Fliege & Svaiter, 2000) is a direction constructed in the intersection of the negative subspaces of the gradients of all the objective functions simultaneously. This direction is defined and discussed in detail in Chapter 5. The computation of α in Q_2, on the other hand, typically reduces to a one dimensional searching problem (Armijo, 1966; Fliege & Svaiter, 2000; Fliege, Vaz, & Vicente, 2019; Truong & Nguyen, 2021) of the type considered in Chapter 4, thus, hopefully this thesis provides improvements on both building blocks of the standard descent direction approach. Finally, the stopping criteria is verified, typically by analysing queried data obtained throughout the run in queries Q_1 and Q_2: this may be through verification of stagnation of objective function, of estimated solution, or more directly by verification of the gradient value on the estimated solution. In Chapter 5, we also propose a new and improved measure of proximity to critical conditions for multi objective optimization problems which directly translate to natural choices for Q_3 discussed therein.

There are several reasons for the popularity of the steepest descent method and it's variants. Perhaps, first and foremost it has often been the method of choice due to it's simplicity and ease of generalization to different classes of problems, and, to the very least has served as a benchmark of performance over which other methods typically aim at improving on (Knuth, 1998; Nocedal & Wright, 2006; Boyd & Vandenberghe, 2009; Nesterov, 2018). A well known fact amongst practitioners is that higher order interpolative methods such as Newton-type methods and/or quasi-Newton methods typically perform better than simple steepest descent strategies (Fletcher, 1987; Conn, Gould, & Toint, 2000; Nocedal & Wright, 2006; Luenberger & Ye, 2018), however, a perhaps more known fact amongst theoreticians is that over broad classes of problems the gradient method actually happens to produce near optimal performance guarantees with respect to worst case metrics (Cartis, Gould, & Toint, 2010; Nesterov, 2018). And, in fact, Newton's method despite accelerated asymptotic performance in the proximity of the solution to smooth functions, the added cost of computing second order derivatives has been shown to have no benefit

when the worst case performance is taken into account; see (Cartis et al., 2010) for a proof of this fact when the objective function is assumed to be twice continuously differentiable, bounded from below, and with bounded Lipschitz continuous gradient.

In the following we provide two worst case results from (Nesterov, 2018) which demonstrate the near optimality of the gradient method over strongly convex functions. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be strongly convex, also referred to as $\mu-$ strong convex, if it is continuously differentiable and there exists a constant $\mu > 0$ such that for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ we have

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) + \tfrac{1}{2}\mu||\boldsymbol{y} - \boldsymbol{x}||^2.$$

And, we refer the reader to Theorems 2.1.12 and 2.1.14 in Chapter 2 of (Nesterov, 2018) for a more complete statement of the inequalities as well as the full proofs (also of interest to the reader might be Theorems 3.2.1 and 3.2.2 where similar inequalities are proved without strong convexity).

Theorem 1 (Lower bound for first order methods - Strong Convexity). *For any $\boldsymbol{x}_0 \in \mathcal{S}$ and any constants $\mu, L > 0$ for every method in which $\boldsymbol{x}_k \in \boldsymbol{x}_0 + Lin\{\nabla f(\boldsymbol{x}_0), ..., \nabla f(\boldsymbol{x}_{k-1})\}$ there exists an infinity differentiable $\mu-$strong convex function f with $L-$Lipschitz continuous gradients such that:*

$$||\boldsymbol{x}_k - \boldsymbol{x}^*|| \geq c^k||\boldsymbol{x}_0 - \boldsymbol{x}^*||; \tag{1.2}$$

where $c \in (0, 1)$ is a function of μ and L, and $\boldsymbol{x}^*$ is the global minima of f.

Theorem 2 (Upper bound for steepest descent - Strong Convexity). *For any $\boldsymbol{x}_0 \in \mathcal{S}$ and any $\mu-$strongly convex function f with $L-$Lipschitz continuous gradients, the steepest descent method implemented with $\alpha = 1/(L + \mu)$ produces estimates $\boldsymbol{x}_k$ such that:*

$$||\boldsymbol{x}_k - \boldsymbol{x}^*|| \leq C^k||\boldsymbol{x}_0 - \boldsymbol{x}^*||; \tag{1.3}$$

where $C \in (0, 1)$ is a function of μ and L, and $\boldsymbol{x}^*$ is the global minima of f.

Theorem 1 provides a hard limit on the worst-case speed of convergence of first order methods, i.e. methods in which the estimates $\boldsymbol{x}_k$ are produced by linear combinations of the directions entailed by the gradients on all previously visited points $\boldsymbol{x}_k \in \boldsymbol{x}_0 + \text{Lin}\{\nabla f(\boldsymbol{x}_0), ..., \nabla f(\boldsymbol{x}_{k-1})\}$. Simultaneously, Theorem 2 shows that the gradient descent method matches that speed of convergence up to the value of the constant C which is greater than c. For tighter and improved first order methods that produces a value of C that match the lower-bound c refer to the accelerated gradient methods in (Nesterov, 2018) and the related literature.

The point of Theorems 1 and 2 is that over the class of strongly convex functions with Lipschitz continuous gradients, the steepest descent method is *near* optimal with respect to worst-case performance for first order methods. In this way, the steepest descent method occupies a similar position that binary searching takes amongst the class of one dimensional solvers, albeit with slightly weaker guarantees. We emphasize however, that this *near* optimal worst-case performance of the gradient method is dependent on both the class of functions considered, as well as the metric of proximity adopted to quantify the distance to the desired solution. In this thesis, specifically in Chapter 5, we assume no form of convexity on the class of

functions under consideration, but only that the functions are lower-bounded and have Lipschitz continuous gradients. For this class of problems much weaker worst case guarantees are known, and, the production of an optimal lower-bound similar to Theorem 1 remains an open problem as pointed out by previous authors (Cartis et al., 2010; Nesterov, 2018). To the best of our knowledge, the best worst-case guarantees produced by first order methods in the literature so far are again attained by the steepest descent method, and, rather than upper-bounding the distance to the optimal solution $\boldsymbol{x}^*$ in decision space they upper-bound the equivalent metric in gradient space, i.e. upper-bounding the norm of the gradient. They are of the form:

Theorem 3 (Upper bound for steepest descent - No Convexity). *For any $\boldsymbol{x}_0 \in \mathcal{S}$ and any function f with $L-$Lipschitz continuous gradients, the steepest descent method (implemented with either Armijo-backtracking, constant step-size or exact minimization) produces estimates $\boldsymbol{x}_t$ for $t = 1, ..., k$ such that:*

$$\min_{t=1,...,k} ||\nabla f(\boldsymbol{x}_t)|| \leq \boldsymbol{C}/\sqrt{k}; \tag{1.4}$$

where $\boldsymbol{C} > 0$ is a function of $f, \boldsymbol{x}_0$ and L.

And thus proximity to *critical conditions* are ensured at the rate of $\sim 1/\sqrt{k}$. This will translate to proximity to a desired solution $\boldsymbol{x}^*$ if additional assumptions are made on the objective function such as convexity or strong convexity (Zhou, 2018). Furthermore, as shown in Theorem 1.2.4 of (Nesterov, 2018), if the function is assumed to be twice continuously differentiable (in the above result it is only assumed to be differentiable once) then the speed at which the estimate $\boldsymbol{x}_k$ approaches the solution $\boldsymbol{x}^*$ is shown to be of the order of $\sim \boldsymbol{\mathcal{C}}^k$ for some $\boldsymbol{\mathcal{C}} \in (0, 1)$ when $\boldsymbol{x}_0$ is initiated sufficiently close to $\boldsymbol{x}^*$, similar to the guarantee given in Theorem 2. Hence, the guarantee in Theorem 3 seems to be the most fundamental convergence result for multi-variate optimization from which more complex ones are derived when more regularity assumptions on the objective function are made.

For multi-objective optimization, a similar result to Theorem 3 was first derived in (Fliege et al., 2019), and, to the best of our knowledge, has remained the best worst-case guarantee produced by first order methods in the multi-objective optimization literature so far under equivalent assumptions on the objective functions. The multi-objective steepest descent equivalent, first proposed in (Fliege & Svaiter, 2000) and analysed in (Fliege et al., 2019), requires computing all objective functions in each iteration in order to iteratively produce a measure of proximity to critical conditions similar to $||\nabla f(\cdot)|| \leq \epsilon$ at a $\sim 1/\sqrt{k}$ speed. In Chapter 5, amongst other results, we provide what we believe to be an improvement on this upper-bound for the multi-objective optimization case; albeit making use of a new descent direction as well as a new metric of proximity to critical conditions in gradient space which significantly differs from the one defined in (Fliege & Svaiter, 2000) and (Fliege et al., 2019). With our construction we provide a new worst-case guarantee which approximates critical conditions with an incremental scheme which requires only two gradient computations per iteration rather than computing all gradients as the original *full-batch* scheme. And thus, we produce for our metric a $\sim 1/\sqrt{k}$ speed of convergence as in Theorem 3, requiring only a fraction of the number of gradient computations as the original results.

References

Argyros, I. K., & Khattri, S. K. (2013). On the secant method. *Journal of Complexity*, *29*(6), 454-471.

Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, *16*(1), 1-3.

Boyd, S., & Vandenberghe, L. (2009). Convex optimization. In (7th ed.). Cambridge, UK: Cambridge University Press.

Brent, R. P. (1971). An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, *14*(4), 422–425. doi: https://doi.org/10.1093/comjnl/14.4.422

Bus, J. C. P., & Dekker, T. J. (1975). Two efficient algorithms with guaranteed convergence for finding a zero of a function. *ACM Transactions on Mathematical Software*, *1*(4), 330–345. doi: https://doi.org/10.1145/355656.355659

Cannizzo, F. (2018). Fast and vectorizable alternative to binary search in O(1) applicable to a wide domain of sorted arrays of floating point numbers. *Journal of Parallel and Distributed Computing*, *113*(5), 37. doi: https://doi.org/10.1016/j.jpdc.2017.10.007

Cartis, C., Gould, N. I. M., & Toint, P. L. (2010, September). On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, *20*(6), 2833-2852. doi: https://doi.org/10.1137/090774100

Conn, A. R., Gould, N. I. M., & Toint, P. L. (2000). *Trust region methods*. SIAM Society for Industrial and Applied Mathematics.

Fletcher, R. (1987). *Practical methods of optimization* (Second ed.). New York, NY, USA: John Wiley & Sons.

Fliege, J., & Svaiter, B. F. (2000). Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, *51*, 479-494. doi: https://doi.org/10.1007/s001860000043

Fliege, J., Vaz, A. I. F., & Vicente, L. N. (2019). Complexity of gradient descent for multiobjective optimization. *Optimization Methods and Software*, *34*(5), 949-959. doi: https://doi.org/10.1080/10556788.2018.1510928

Knuth, D. E. (1998). The art of computer programming - sorting and searching. In (2nd ed., Vol. 3, chap. 6.2). Addison-Wesley.

Luenberger, D. G., & Ye, Y. (2018). Linear and nonlinear programming. In (4th ed.). Springer International Publishing.

Nesterov, Y. (2018). Lectures on convex optimization. In (2nd ed., Vol. 137). Springer International Publishing.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (second ed.). New York, NY, USA: Springer.

Novak, E., & Ritter, K. (1993). Some complexity results for zero finding for univariate functions. *Journal of Complexity*, *9*(1), 15-40. doi: https://doi.org/10.1006/jcom.1993.1003

Novak, E., Ritter, K., & Woźniakowski, H. (1995). Average-case optimality of a hybrid secant-bisection method. *Mathematics of Computation*, *64*(212), 1517–1539. doi: https://doi.org/10.2307/2153369

Oliveira, I. F. D. (2013). Optimal black-box sequential searching. Masters Thesis of the Mathematics Department of the Federal University of Minas Gerais. Retrieved from `http://hdl.handle.net/1843/EABA-98VHPQ` (Advised by Professor R.H.C. Takahashi.)

Perl, Y., Itai, A., & Avni, H. (1978). Interpolation search—a log log n search. *Communications of the ACM*, *21*, 550-553. doi: https://doi.org/10.1145/359545.359557

Perl, Y., & Reingold, E. M. (1977). Understanding the complexity of interpolation search. *Information Processing Letters*, *6*(6), 219-222. doi: https://doi.org/10.1016/0020-0190(77)90072-2

Ridders, C. (1979). A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, *26*(11), 979–980. doi: https://doi.org/10.1109/TCS.1979.1084580

Sikorski, K. (1982). Bisection is optimal. *Numerische Mathematik*, *40*(1), 111-117. doi: https://doi.org/10.1007/BF01459080

Truong, T. T., & Nguyen, H. T. (2021, September). Backtracking gradient descent method and some applications in large scale optimisation. part 2: Algorithms and experiments. *Applied Mathematics & Optimization*, *84*, 2557–2586. doi: https://doi.org/10.1007/s00245-020-09718-8

Vieira, D. A. G., Takahashi, R. H. C., & Saldanha, R. R. (2012). Multicriteria optimization with a multiobjective golden section line search. *Mathematical Programming*, *131*(1-2), 131–161. Retrieved from `https://doi.org/10.1007/s10107-010-0347-9` doi: 10.1007/s10107-010-0347-9

Yao, A. C., & Yao, F. F. (1976). The complexity of searching an ordered random table. *Proceedings of the Seventeenth Annual Symposium on Foundations of Computer Science*, 173-177. doi: https://doi.org/10.1109/SFCS.1976.32

Zhou, X. (2018). On the fenchel duality between strong convexity and lipschitz continuous gradient. *arXiv Preprint arXiv:1803.06573*.

The root searching problem

Summary. This chapter, contained in pages 22 to 54, is based on the the paper *An enhancement of the bisection method average performance preserving minmax optimality* first submitted to the Transactions on Mathematical Software on the 2nd of February of 2019 (Oliveira & Takahashi, 2020). A final revised version was accepted for publication on the 9th of September of 2020 and can be found in the following address: `https://dl.acm.org/doi/10.1145/3423597`. In the current form presented below, minor changes are made to make for a more coherent read within the context of this thesis, however, it is kept self contained and an independent read of Chapter 2 is possible without need of reference to external material within the thesis. Any major content foreign to the original paper is highlighted with dark blue text, and, the remainder, where little or no alteration is made, we present the content in plain black text.

In this chapter we formally study the root searching problem introduced in Chapter 1. In particular we identify a class of root searching methods that surprisingly outperform the bisection method on the average performance while retaining minmax optimality. The improvement on the average applies for any continuous distributional hypothesis. We also pinpoint and recommend one specific method within the class and show that under mild initial conditions it can attain an order of convergence of up to 1.618, i.e. the same as the secant method. Hence we attain both an improved average performance and an improved order of convergence with no cost on the minmax optimality of the bisection method. Numerical experiments were also conducted and we find that on regular functions, the proposed method required a number of function evaluations similar to current state-of-the-art methods, about 24% to 37% of the total evaluations required by the bisection procedure. In the case of problems with non-regular functions, the proposed method performed significantly better than the state of the art, requiring on average about 82% of the total evaluations required for the bisection method while the other methods were outperformed by the minmax bound of the bisection method. In the worst case, while current state of the art commercial solvers required between two to three times the number of function evaluations of the bisection, our proposed method remained within the minmax bounds of the bisection method.

2.1 Introduction

The traditional root-searching problem (Brent, 1971; Bus & Dekker, 1975; Argyros & Khattri, 2013) is stated as:

$$\text{Find } x^* \text{ such that } f(x^*) = 0; \tag{2.1}$$

where $f : \mathbb{R} \mapsto \mathbb{R}$ is a continuous function over an interval $[a, b]$ such that $f(a)f(b) < 0$. The *Intermediate Value Theorem* guarantees the existence of a solution to (2.1) and, if furthermore f is assumed to be strictly monotone, then uniqueness can be guaranteed. Close variants to (2.1) include searching lists (Yao & Yao, 1976), roots of polynomials (McNamee & Pan, 2012) as well as searching for minima of univariate functions (Kiefer, 1953).

Broadly speaking, a numerical solver for problem (2.1) is an algorithm that evaluates function f, f' or any other higher order derivative on a finite number of points and returns an estimate $\hat{x}$ that satisfies $|\hat{x} - x^*| \leq \epsilon$ for some pre-specified precision $\epsilon > 0$ (Press, Teukolsky, Vetterling, & Flannery, 2007; Chapra & Canale, 2010). Solvers are typically initiated with the values $a, b, f(a), f(b)$ and ϵ, and different solvers often specify recursively how to obtain $\tilde{x}_k$ from the data collected on previously visited points $\tilde{x}_0, \tilde{x}_1, ..., \tilde{x}_{k-1}$, terminating when $\hat{x} = \tilde{x}_n$ is identified. Methods of constructing such sequences are referred to by several different names including *root-searching methods* (Gal & Miranker, 1977), *non-linear equation solvers* (Nerinckx & Haegemans, 1976) and *zero-finding* (Le, 1985b) to mention a few.

Different methods mainly differ on (i) the type of data collected on each iteration, (ii) the mapping from the collected data to the next point $\tilde{x}_k \in (a, b)$, and (iii) the stopping criterion. Methods that require a minimal amount of function and derivative evaluations are preferred (Traub, 1963; Nerinckx & Haegemans, 1976; Le, 1985a; Shrager, 1985; Novak, 1989; Ritter, 1994; Wu, 2005; Zhang, 2011), due to the assumption that on average the time to evaluate the function and/or its derivative is much more costly (or time consuming) than any other calculations involved in the execution of the method itself. We also make use of this assumption henceforth.

Throughout, whenever we refer to problem (2.1), function $f : \mathbb{R} \to \mathbb{R}$ will be assumed to be continuous and strictly monotone, however, the methods presented in this text are suitable for dealing with a more general version of the root-searching problem which may be stated as:

$$\text{Find } x^* \in [a, b] \text{ such that } f(x^* - \delta)f(x^* + \delta) \leq 0 \tag{2.2}$$

for some δ less than or equal to a pre-specified value $\epsilon > 0$, given that $f(a)f(b) < 0$. Function monotonicity and solution uniqueness are not assumed and, in the case of discontinuous functions, the solution to this problem may be a point of discontinuity in which the sign of the function changes.

Binary Search

Binary search, particularly known as the bisection method within the field, is a cornerstone in the analysis of root-searching methods (Nerinckx & Haegemans, 1976; Sikorski, 1982; Le, 1985a; Sikorski, 1985; Novak, Ritter, & Woźniakowski, 1995; Press et al., 2007; Chapra & Canale, 2010). Both its historical and practical importance are

hard to overestimate as its use in tackling problem (2.1) is widespread, many times as a subroutine of a parent algorithm (Eiger, Sikorski, & Stenger, 1984; Kearfott, 1987; Vrahatis, 1988). It should be noticed that the bisection method also solves problem (2.2). The bisection method begins with the value of $a, b, y_a = f(a)$ and $y_b = f(b)$ and at each iteration it stores and updates lower and upper bounds for the location of x^*. This is done by evaluating f at $\tilde{x}_k = (a_k + b_k)/2$ and if $f(\tilde{x}_k)f(a_k) > 0$ then a_{k+1} and b_{k+1} are defined as $a_{k+1} = \tilde{x}_k$ and $b_{k+1} = b_k$; otherwise if $f(\tilde{x}_k)f(b_k) > 0$ then $a_{k+1} = a_k$ and $b_{k+1} = \tilde{x}_k$; and if $f(\tilde{x}_k) = 0$ then $a_{k+1} = b_{k+1} = \tilde{x}_k$. The bisection method stops when Δ_n, defined as $\Delta_n \equiv b_n - a_n$, satisfies $\Delta_n \leq 2\epsilon$ and then returns $\hat{x} = (a_n + b_n)/2$. This is summarized in the following algorithm where the indices are omitted and $\tilde{x}$ in line (i) is taken to be $(a + b)/2$.

Algorithm 0: The Bracketing Algorithm

Input: $a, b, y_a, y_b, f, \epsilon$

(i). Choose $\tilde{x} \in (a, b)$ and evaluate $f(\tilde{x})$;

(ii). Update a and b according to the values of $f(a), f(b)$ and $f(\tilde{x})$;

(iii). If $\Delta > 2\epsilon$ return to (i), else terminate.

Output: $\hat{x} = (a + b)/2$

Perhaps the strongest appeal to use the bisection method is in its minmax optimality:

Theorem 4. *For any instance of problem (2.1) the bisection method requires at most:*

$$n_{1/2} = \left\lceil \log_2 \frac{b-a}{2\epsilon} \right\rceil \tag{2.3}$$

function evaluations to locate $\hat{x}$ such that $|\hat{x} - x^*| \leq \epsilon$. No other method can provide the same guarantee in less than $n_{1/2}$ function evaluations.

Theorem 4 has a few remarkable implications. A first and immediate consequence is that the bisection method is optimal in a worst case sense over the class of continuous and strictly monotone functions. This also happens to be true for several other classes of functions. For more on the optimality of the bisection method and related proofs refer to Sikorski (1982, 1985). A second implication, perhaps more subtle, is that the upper bound (2.3) cannot be broken by any method, even if it is allowed at each step to evaluate derivatives of f on any finite amount of points. Thus Theorem 4 identifies the *critical information* to be obtained at each step, namely the comparative value between $f(\tilde{x})$ and the values of $f(a)$ and $f(b)$.

We point out that the bisection method almost always requires $n_{1/2}$ iterations, because it can only terminate in less iterations if (luckily) it happens that $f(\tilde{x}_k) = 0$ for some $k < n_{1/2}$. The subset of points the bisection method can visit for a given problem is always finite, thus, for any distribution over instances of (2.1) that induces a continuous distribution $\mathcal{D}^*$ over (a, b) such that $x^* \sim \mathcal{D}^*$, the expected number of iterations required by the bisection method will always[1] be $n_{1/2}$. Notice also that

[1] More precisely, an early stop will occur only when any of the points of the sequence generated by the bisection algorithm, truncated up to the machine finite precision,

Theorem 4 does not guarantee uniqueness of the bisection method, i.e. depending on the initial conditions of (2.1) other choices of $\tilde{x}$ different than $x_{1/2}$ can also bracket the solution up to ϵ precision with $n_{1/2}$ iterations. Is it possible to characterize the class of methods that attain minmax optimality? Can any minmax method terminate in less than $n_{1/2}$ iterations without (luckily) finding $f(\tilde{x}_k) = 0$?

In this chapter, we answer the above questions affirmatively. The intuition behind our findings is that the bisection almost always produces an estimate $\hat{x}$ with a precision higher than needed, i.e. with an error less than ϵ. Hence, a careful analysis allows one to backtrack and see how much one can deviate from the bisection while still remaining within the bounds of Theorem 4. It turns out that the class of minmax strategies is characterized by a neighborhood of $x_{1/2}$ in each iteration (described in Theorem 5 ahead). This class encompasses both randomized and interpolation based strategies, and more importantly, a refined use of the available information at each iteration allows for an improvement on the *average performance* as well as the *order of convergence* with no risk of requiring more than $n_{1/2}$ iterations (both of which are defined ahead).

Chapter Outline.

In the remainder of this section we provide a small review of different metrics of performance for root searching methods, namely: *asymptotic order of convergence* and *average performance*. Then, in Section 2.2 we characterize the class of minmax optimal root-searching methods that, as the bisection method, solve problem (2.1) with ϵ precision in no more than $n_{1/2}$ iterations. Then, we identify one specific minmax method and show that under regularity conditions a super-linear convergence can be attained. Finally, we test our method against several classical alternatives to the bisection method and show both theoretical and empirical evidence that this method is a viable substitute not only to the bisection method, but also to current state of the art solvers. This chapter is also followed by an appendix with detailed proofs, a line-by-line pseudo-code for ease of implementation of the main method, and additional numerical experiments.

2.1.1 Other metrics and competing strategies

Other root-searching methods pattern similarly to the bisection method. They belong to a broad class of root-searching methods which we will refer to as bracketing algorithms. Bracketing algorithms mainly differ on the choice of $\tilde{x}$ in line (i) of Algorithm 0, though, some also differ in the stopping criterion in line (iii) as well as the output $\hat{x}$. We emphasize that many root searching algorithms belong to this class. Below we display the choice for $\tilde{x}$ in line (i) for the bisection method, the regula-falsi and Ridders' rule:

$$\textbf{Bisection:} \qquad x_{1/2} = \frac{a+b}{2}; \tag{2.4}$$

$$\textbf{Regula-Falsi:} \qquad x_f = \frac{bf(a) - af(b)}{f(a) - f(b)}; \tag{2.5}$$

becomes equal to the value of x^*, also truncated up to the machine precision. The probability of such an event will be negligible for any continuous distribution $\mathcal{D}^*$.

$$\textbf{Ridders' Rule:} \qquad x_r = x_{1/2} \pm \frac{(b-a)f(x_{1/2})}{2\sqrt{f(x_{1/2})^2 - f(a)f(b)}}. \tag{2.6}$$

The regula-falsi method combines linear interpolation with bracketing in an attempt to attain faster convergence (Dowell & Jarratt, 1971; Novak et al., 1995). It can be shown that, under regularity conditions, one of the end points, a or b, will always converge to x^*, though the stopping criterion $b - a \leq 2\epsilon$ may never be satisfied. In fact, in the proximity of the root of strictly convex (or concave) functions $b - a$ never converges to 0. Alternatively, Ridders' method performs an exponential interpolation between the extremities a and b and an auxiliary point, namely $x_{1/2}$. This comes at a cost of two function evaluations per iteration to attempt faster convergence (Ridders, 1979; Press et al., 2007) and, since both $f(x_r)$ and $f(x_{1/2})$ are evaluated once in each iteration, the method performs at most twice the amount of function evaluations as the bisection method in order to terminate given that line (ii) of Algorithm 0 updates the values of a and b accordingly.

Remark 1. Other popular methods, such as Brent's method (Brent, 1971; Zhang, 2011; Stage, 2013), Muller's method (Muller, 1956), Bus and Dekker's method (Bus & Dekker, 1975) etc., can often be slightly modified to keep track of upper and lower bounds for x^* after each function evaluation. Thus, virtually any method can be patterned as a bracketing algorithm.

Deviation from the bisection method is often motivated by one of two reasons: to attain a higher *order of convergence* (Muller, 1956; Brent, 1971; Le, 1985b, 1985a; Norton, 1985; Shrager, 1985; Ford, 1995; Wu, 2005; Segura, 2010; Zhang, 2011; Argyros & Khattri, 2013) or attain an improved *average performance* (Novak, 1989; Ritter, 1994; Novak et al., 1995). Throughout the literature the improvements have always come at the expense of the worst-case performance of the bisection method. The average performance of a given method is defined as the expected number of function evaluations required in order to locate $\hat{x}$ such that $|\hat{x} - x^*| \leq \epsilon$ for some pre-specified distribution over instances of (2.1). Order of convergence, in turn, is typically defined if there exist $\bar{x}$, $\beta \geq 1$ and $\alpha > 0$ in $\mathbb{R}$ such that:

$$\lim_{k\to\infty} \frac{|\tilde{x}_{k+1} - \bar{x}|}{|\tilde{x}_k - \bar{x}|^\beta} = \alpha; \tag{2.7}$$

in which case we say that the sequence converges with an order of convergence β and with an asymptotic error constant α; and thus the term *order of convergence* is used interchangeably with *asymptotic performance*. Of course, when dealing with methods that are guaranteed to terminate after a finite amount of iterations (such as Ridders rule, or the Bisection Method etc.), then in order to use definition (2.7) the finite sequence is either extended to an infinite sequence by ignoring the stopping criteria, or simply a slightly different definition is adopted where $|\tilde{x}_{k+1} - \bar{x}|$ is written as a function of $|\tilde{x}_k - \bar{x}|$ decomposed in two parts: the first being $\alpha|\tilde{x}_k - \bar{x}|^\beta$ and the second containing terms of higher order.

Methods with high order of convergence are often interpreted as requiring less function evaluations than methods with low order of convergence. If $\beta > 1$, and if the asymptotic behavior described in (2.7) kicks in early on, then, the number of iterations can be shown to be of the order of $\sim \log_2 \log_2(1/\epsilon)$ with constants that depend on Δ_0, α and β. However, even when convergence is guaranteed and conditions for a high order of convergence are met, asymptotic behavior may only

kick in very late in a run. Hence, order of convergence often does not translate to number of function evaluations in a finite iteration scenario. Secondly, high order of convergence is typically a local property that requires proximity to the solution x^*, and finding an initial solution x_0 that is sufficiently close to x^* is, in general, as hard as solving the original problem. Hence, upper-bounds on the worst case behavior are critical to ensure that a high order of convergence can be attained in the first place, and thus, even state of the art solvers that aim for high orders of convergence will typically also ensure upper-bounds on the worst case behavior which, as mentioned earlier, have always been weaker than bisection guarantees (Brent, 1971; Bus & Dekker, 1975).

When compared to asymptotic results, much less literature has been produced on the average performance of root-searching methods (Graf, Novak, & Papageorgiou, 1989; Novak, 1989; Novak & Ritter, 1993; Ritter, 1994; Novak et al., 1995). While high order of convergence is typically dependent on local properties (as discussed above), average results are dependent on the assumed distribution over instances of (2.1). Under the correct distributional assumption considerable speedups can be attained, see (Novak et al., 1995) for a nice example. However, as in the asymptotic results, accelerated convergence has always come at the cost of a reduced worst-case performance. And, since general purpose solvers typically operate uninformed of the underlying distributional framework, the use of distribution specific methods can be hard to justify in practice. Hence, similarly to asymptotic results, the state of the art of methods crafted for high average performance are also accompanied by worst case guarantees which, again, are weaker than bisection guarantees (Novak et al., 1995).

In this study, we identify and test a simple and yet novel method that improves on the bisection method by yielding a superior average and asymptotic performance without loss on the minmax performance. This is a key difference from former studies, since any method that proposes to improve on the bisection method and yet lacks on minmax optimality will always find situations where the traditional bisection method is preferred. Our improvements on the average applies to any distribution over instances of (2.1) that induce a continuous distribution $\mathcal{D}^*$ over (a, b) where $x^* \sim \mathcal{D}^*$, and our improvements on the order of convergence require similar smoothness conditions as those required by traditional methods.

2.2 Main results

In this section we will show that in the neighborhood of $x_{1/2}$ there almost always exists an interval $\mathcal{I}$ such that any choice of $\tilde{x} \in \mathcal{I}$ retains overall minmax optimality. This result is precisely stated in Theorem 5 ahead, however, before the main statement we will begin by showing how this can be derived quite simply for the first iteration of Algorithm 0.

First notice that the value of $n_{1/2}$ in (2.3) can be equivalently identified as the minimal integer $n \in \mathbb{N}$ that satisfies the inequality: $\Delta_0 \leq \epsilon 2^{(n+1)}$. This inequality, and slight variations of it, will play a key role in our analysis. Assuming $n_{1/2} \geq 1$, after one iteration of Algorithm 0 a minmax method is left with at most $n_{1/2} - 1$ function evaluations. Thus, we find that:

$$\Delta_1 \leq \epsilon 2^{n_{1/2}}. \tag{2.8}$$

Notice furthermore that if Δ_{i+1} is not null then it can take only two possible values, either $\Delta_{i+1} = \tilde{x}_i - a_i$ or $\Delta_{i+1} = b_i - \tilde{x}_i$ depending on the value of the sign of $f(\tilde{x}_i)$. From this and a little algebra one can show that:

$$\Delta_{i+1} = \Delta_i/2 \pm |\tilde{x}_i - x_{1/2}|. \tag{2.9}$$

Combining (2.8) and (2.9) we find that a minmax optimal strategy must have $\tilde{x}$ in the first iteration satisfying the following inequality:

$$|\tilde{x} - x_{1/2}| \leq \epsilon 2^{n_{1/2}} - \frac{b-a}{2}. \tag{2.10}$$

The right hand side of equation (2.10) is null when $(b-a)/\epsilon = 2^m$ for some $m \in \mathbb{N}$ and otherwise it is equal to some positive value $r < (b-a)/2$. It is only in the unlikely case when $(b_0 - a_0)/\epsilon$ is a power of 2 that $x_{1/2}$ will turn out to be the unique minmax strategy in step 1. In every other situation, the bisection method is not alone. This is illustrated by Figure 2.1.

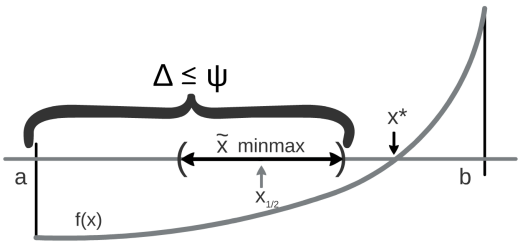


Fig. 2.1: There exists a symmetrical interval in the neighborhood of $x_{1/2}$ in which any $\tilde{x}$ within this set is minmax optimal. These points cut the original interval (a, b) into two sub-intervals of lengths Δ that are no greater than $\psi \equiv \epsilon \cdot 2^{n_{1/2}}$, which corresponds to the maximal interval length in which the search is still guaranteed to finish in $n_{1/2} - 1$ iterations.

In the following we apply the same reasoning to find the equivalent of (2.10) for further iterations beyond the first one; we find that:

Theorem 5. *A bracketing strategy is minmax optimal if and only if in every iteration $k = 0, 1, 2...$ of Algorithm 0, the value of $\tilde{x} = \tilde{x}_k$ in line (i) satisfies:*

$$|\tilde{x}_k - x_{1/2}| \leq \epsilon 2^{n_{1/2}-k} - \frac{b_k - a_k}{2}; \tag{2.11}$$

when $n_{1/2} - k = \lceil \log_2 \frac{b_k - a_k}{2\epsilon} \rceil$, and, in any other case $\tilde{x}_k$ can assume any value in (a_k, b_k).

Theorem 5 identifies the full class of bracketing methods that, as the bisection method, attain minmax optimality. The bisection method is, in general, accompanied by a broad class of minmax strategies given that $(b_0 - a_0)/\epsilon$ is not equal to 2^m for some $m \in \mathbb{N}$. The value of $\tilde{x}$ may be chosen by means of interpolation, randomization

or any other means as long as $\tilde{x}$ remains within the neighborhood established by Theorem 5. This result delimits a rich class of root searching strategies that, with respect to minmax performance, are all equally viable.

One remarkable consequence of Theorem 5 is that virtually any minmax strategy is a viable substitute to the traditional bisection method, specifically when average performances are taken into consideration. This is so because, as mentioned earlier, for any distribution over instances of (2.1) that induces a continuous distribution $\mathcal{D}^*$ over (a, b) with $x^* \sim \mathcal{D}^*$ the expected number of iterations the bisection method requires to terminate is always $n_{1/2}$. A trivial consequence of Theorem 5 is that $n_{1/2}$ is an upper bound on the expected number of iterations for the entire class of minmax strategies. Thus, amongst this class, the bisection method happens to have the worst possible average performance over continuous distributions:

Corollary 1. *For any distribution over instances of (2.1) that induces a continuous distribution $\mathcal{D}^*$ over (a, b) such that $x^* \sim \mathcal{D}^*$ the expected number of iterations n_{av} of any minmax strategy satisfies:*

$$n_{av} \leq n_{1/2}. \tag{2.12}$$

The inequality in Corollary 1 may, given the appropriate conditions, be strict. For example, it is possible to show that if $\lceil (b-a)/\epsilon \rceil \neq 2^k$ for some $k \in \mathbb{N}$, then, a random sample over the interval defined by (2.11) suffices as a choice of $\tilde{x}$ to attain a strict inequality in (2.12). Hence, even a random "blind guess" strategy within the set identified by Theorem 5 can outperform the traditional bisection method on average, while keeping the bound on the worst-case performance. It should be noticed, however, that a "blind guess" strategy would still lead to an n_{av} close to $n_{1/2}$ since a typical run of a blind guess strategy quickly degenerates (2.11) to a very small neighbourhood around $x_{1/2}$ after a few "bad guesses". However, the inequality in Corollary 1 may lead to a relevant difference between n_{av} and $n_{1/2}$ if the rule for choosing $\tilde{x}$ is based on information that allows the choice of favorable cuts with an increased probability. This information may be provided, for instance, by interpolation-based guesses.

Before we proceed to the recommendation of one specific minmax method, we discuss two natural variations of (2.11) with immediate applications in the crafting of methods for solving (2.1). The first: if we wish to characterize the class of methods that require at most $n_{\max}$ iterations with $n_{\max} > n_{1/2}$, all that is needed is to replace $n_{1/2}$ in (2.11) with $n_{\max}$. Thus, this variation of inequality (2.11) encompasses previous methods that provided weaker than minmax guarantees (such as Ridders' rule (Ridders, 1979), Bus and Dekker (Bus & Dekker, 1975) etc.), and, one immediate application of this inequality is that, for example, it is easy to use it to build an "enhancement of Ridders' method" and of others. Since (2.11) with $n_{\max} > n_{1/2}$ will always be initiated with at least $n_{\max} - n_{1/2}$ unconstrained iterations, then this can be done quite simply by implementing any higher-order interpolation method while keeping track of the range described by the modified version of (2.11). Thus, we can easily guarantee the same worst-case performance of Ridders' Method while attaining the order of convergence of the fastest known method, and, if $\tilde{x}$ falls outside the bounds of the inequality then all one needs to do is to fall back to binary search. A second variation of (2.11) is obtained by a more conservative approach. While (2.11) alone can guarantee an upper-bound on the global iteration count, it does not avoid

Algorithm 0 from "wasting" free iterations gained throughout a run. While a bracketing strategy with $\tilde{x} \in (a, b)$ can guarantee that $\Delta_{k+1} < \Delta_k$, it cannot guarantee that $\lceil \log_2(b_{k+1} - a_{k+1})/2\epsilon \rceil < \lceil \log_2(b_k - a_k)/2\epsilon \rceil$. Hence, in order to guarantee that Algorithm 0 is "at least as good" as the bisection method in reducing the iteration count at each step we must enforce that $|\tilde{x}_k - x_{1/2}| \leq \epsilon 2^{\lceil \log_2(b_k - a_k)/2\epsilon \rceil} - (b_k - a_k)/2$ even when minmax optimality does not require it. This second variation of (2.11) can be of interest in scenarios where traditional oracles (such as interpolation based estimators) are unreliable and "wasteful" iterations are expected, but also in situations in which function f contains discontinuities or contains flat regions, which are known to be difficult corner cases for interpolation based methods.

In summary, Theorem 5 and the variations pointed out seem to provide a rich set of viable alternatives to the bisection method that can: (i) retain minmax optimality and (ii) terminate on average in n_{av} iterations with $n_{av} \leq n_{1/2}$ for any continuous distribution $\mathcal{D}^*$. And, if we adopt the first and less restrictive variation thereof we can easily (iii) attain orders of convergence compatible with the fastest known methods; and, if we adopt the second and more restrictive version thereof we can (iv) avoid "wasteful" iterations if estimators are unreliable.

2.2.1 One interpolation based minmax strategy

In this subsection we identify one specific minmax strategy that, given the appropriate conditions, converges at a super-linear rate. The basic idea of the strategy hereby delineated is to make a projection of a linear estimator onto the subset of minmax methods described by (2.11). However, before the projection step we perform a truncation on the estimator for reasons that will become clear later on. For ease of read and implementation, we provide in the appendix a full line-by-line description of the ITP method algorithm delineated and analysed in this section.

Interpolation, Truncation and Projection (ITP Method).

Let ϕ denote the golden ratio $\frac{1}{2}(1 + \sqrt{5})$ and let $\kappa_1 \in \mathbb{R}_+$ and $\kappa_2 \in [1, 1 + \phi)$ be two user provided constants. Now define:

$$\sigma \equiv \text{sign}(x_{1/2} - x_f) \qquad \text{and} \qquad \delta \equiv \kappa_1 |b - a|^{\kappa_2}, \tag{2.13}$$

where $x_{1/2}$ and x_f are as in (2.4) and (2.5) respectively, and define x_t as

$$x_t \equiv x_f + \sigma\delta \tag{2.14}$$

if $\delta \leq |x_{1/2} - x_f|$ and $x_t \equiv x_{1/2}$ otherwise. We refer to x_t as the truncation of x_f. Also, in accordance with (2.11), define the minmax radius r_k and interval $\mathcal{I}_k$ as

$$r_k \equiv \epsilon 2^{n_{1/2} - k} - \tfrac{b_k - a_k}{2} \qquad \text{and} \qquad \mathcal{I}_k \equiv \left[x_{1/2} - r_k \, , \, x_{1/2} + r_k \right] \tag{2.15}$$

Now, in each step k the ITP method defines $\tilde{x}_k$ as the projection of x_t onto $\mathcal{I}_k$, i.e.

$$x_{ITP} \equiv \begin{cases} x_t & \text{if } |x_t - x_{1/2}| \leq r_k; \\ x_{1/2} - \sigma r_k & \text{otherwise.} \end{cases} \tag{2.16}$$

In the following theorem we will assume that the minmax interval $\mathcal{I}_0$ around $x_{1/2}$ in the first iteration $k = 0$ is "not too small", i.e. that $\frac{2r_0}{b_0 - a_0}$ is not much

smaller than one. This avoids the collapsing of $\mathcal{I}_k$ to $x_{1/2}$ (in which case the ITP method behaves identical to the bisection method), and also, as shown in the proof of Theorem 6, in combination with the other conditions it guarantees that the steady state with super-linear convergence can be reached within a few iterations.

Theorem 6. *Assume that Algorithm* 0 *is implemented with* $\tilde{x}$ *equal to* x_{ITP}*, and function* f *is* $\mathcal{C}^2$ *with* x^* *a simple root. Then, if* a *and* b *are sufficiently close to* x^* *and* $\mathcal{I}_0$ *is not too small, the residual* Δ *converges to zero with an order of convergence of* $\sqrt{\kappa_2}$*, i.e. there exists* $\kappa_3 > 0$ *such that:*

$$\Delta_{n+1} \sim \kappa_3 \Delta_n^{\sqrt{\kappa_2}}; \tag{2.17}$$

where $\kappa_3 = \kappa_1^{1/(1+\sqrt{\kappa_2})}$ *if* $\kappa_2 > 1$ *and* $\kappa_3 = 1/2$ *otherwise.*

Proof. The overall structure of the proof is as follows: We begin by analyzing Algorithm 0 when $\tilde{x}$ is taken to be equal to x_t (without the projection). We will see that if function f is in $\mathcal{C}^2$ and x^* is a simple root, then Algorithm 0 with $\tilde{x} = x_t$ produces a residual Δ_n that vanishes with an order of convergence of $\sqrt{\kappa_2}$. The proof makes use of a lemma that describes the worst case behavior of the unprojected x_t. Then, in order to prove the asymptotic behavior of the ITP method, we show that under the conditions in which x_t converges with an order of $\sqrt{\kappa_2}$, if $\mathcal{I}_0$ is not too small, then in each step the minmax interval described by (2.15) will increase and eventually allow that $\tilde{x}$ be any value within (a_n, b_n). At this point onward the projection step will continually be equal to x_t and thus $\tilde{x}$ will also have an order of convergence of $\sqrt{\kappa_2}$. The detailed development of this proof is presented in the appendix. □

Since $\kappa_2 \in [1, \phi + 1)$, the asymptotic order of convergence $\sqrt{\kappa_2}$ in Theorem 6 can range anywhere in $[1, \phi)$. The upper bound, given by the golden ratio, is the asymptotic order of convergence of the secant method. Hence, the ITP method provides means of attaining a high order of convergence while preserving minmax optimality.

While higher values of κ_2 are associated with a higher order of convergence, as a consequence of Lemma 1 (and the discussions within the proof of Theorem 6), a higher order of convergence comes at a trade-off. The first and immediate trade-off of opting for high values of κ_2 is that the worst case guarantees of x_t (but not of $\tilde{x}$) are weakened with higher values of κ_2. Thus, if Algorithm 0 has produced "spare iterations" after several successful steps in making $\Delta_{n+1} < \Delta_n/2$, then, iterations that do not produce $\Delta_{n+1} \leq \Delta_n/2$ can more rapidly spend those "spare iterations". The second trade-off associated with high values of κ_2 is that the super linear convergence is only attained when the error δ is greater than the estimation error $|x_f - x^*|$. For low values of κ_2 this steady state condition is more easily attained than for high values, i.e. for high values of κ_2, the values of a and b must be initiated closer to x^* in order for x_t to be initiated at steady state condition. Thus, if the estimation error is high in the first iterations, then an over-commitment with the estimator x_f associated with the choice of a high value for κ_2 might come at a cost of having (2.15) degenerating to the bisection method early on. The choice of a constant κ_1 is associated with a similar trade-off: low values for κ_1 produce tighter asymptotic behavior, and, higher values facilitate the initiation of steady state behavior of x_t attained when the truncation error δ is greater than the estimation error $|x_f - x^*|$.

In the following we additionally provide two images in figures 2.2 and 2.3 to illustrate the construction of the ITP method:

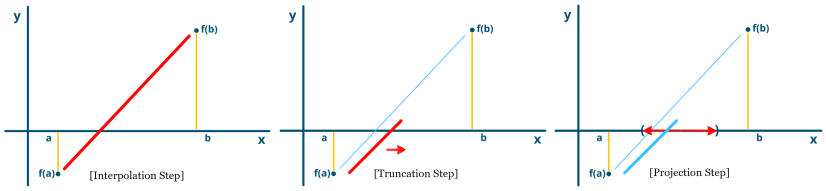


Fig. 2.2: The ITP method is constructed in three steps. In the first step (illustrated in the left most image) a simple linear interpolation is performed; then, in the second step (illustrated in the middle) the truncation perturbs the linear estimate towards the mid point; and, finally, in the third step (in the right) a verification is made of whether the point is in the minmax interval and if so it is kept and if not the closest point in the interval is taken.

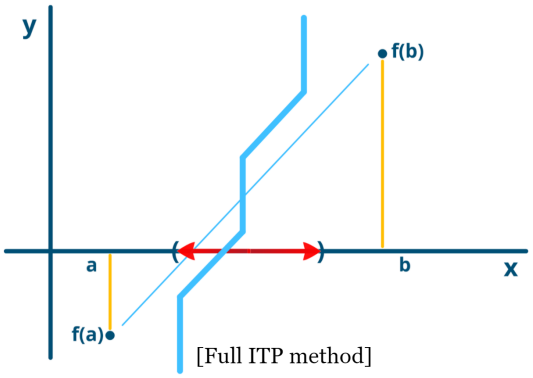


Fig. 2.3: An illustration of all three steps of the ITP method combined. The thin blue line represents the regula-falsi linear interpolation, in red we depict the minmax interval, and, the thick blue line represents the result of all three operations of interpolation, truncation and projection producing the ITP estimate.

Remark 2. The truncation step makes $x_t = x_{1/2} + \sigma\delta$ if $\delta \leq |x_{1/2} - x_f|$ and $x_t = x_{1/2}$ otherwise. The intuition behind this operation is firstly to attempt to make Δ_{k+1} less than $\Delta_k/2$. Notice that this only occurs when $\tilde{x}$ is chosen to be some value between $x_{1/2}$ and x^*. Thus, since x_f is our "best guess" for the location of x^* the truncation is performed from x_f towards $x_{1/2}$ as defined in σ and with size δ. When δ is greater than $|x_{1/2} - x_f|$, then $x_{1/2} + \sigma\delta$ is no longer between the values of x_f

and $x_{1/2}$ and thus a more sensible choice is to make $x_t = x_{1/2}$ and wait for δ to become sufficiently small, i.e. less than $|x_{1/2} - x_f|$, in the next iteration.

Remark 3. If in (2.15) we use $n_{\max}$ instead of $n_{1/2}$ with $n_{\max} > n_{1/2}$, then, the first $n_{\max} - n_{1/2}$ iterations of Algorithm 0 are always free to query any value of $\tilde{x}$ in (a, b). Only if the residuals do not satisfy $\Delta_{k+1} \leq \Delta_k/2$ for several iterations it will occur that the constraint in (2.15) reduces the region of (a, b) available to query. Thus, by adopting $n_{\max}$ greater than $n_{1/2}$ a few things happen: (i) Algorithm 0 is already initiated with $\tilde{x} = x_t$; and (ii) a more "forgiving" version of the ITP method is obtained that allows for a few iterations with $\Delta_{k+1} > \Delta_k/2$ before reducing (2.15) to $x_{1/2}$; and (iii) the first iterations of Algorithm 0, despite not being minmax, will be accompanied by the worst-case bounds of Lemma 1. Thus, the condition "if $\mathcal{I}_0$ is not too small" in Theorem 6 can be dropped in exchange for relaxing minmax optimality by adopting $n_{\max} > n_{1/2}$.

2.2.2 Additional axiomatic support for the ITP method

In this subsection we provide one additional, simple yet elegant, support for the ITP method via an axiomatic construction similar to the one found in "Why linear interpolation" by Pownuk 2017 (Pownuk & Kreinovich, 2017). The axiomatic construction is not complete in the sense that it pin-points the ITP method uniquely, however, it does discard other methods proposed so far in the literature. For simplicity, and without loss of generality, we assume henceforth that $y_a = f(a) < 0$ and $y_b = f(b) > 0$.

First we state our base premises.

$\mathcal{P}_1$ **(Memory Limitation)** *The choice of $\tilde{x}$ is a function continuously dependent on a, b, y_a, y_b and ϵ alone, i.e.: $\tilde{x} = g(a, b, y_a, y_b, \epsilon)$.*
$\mathcal{P}_2$ **(Domain Translation)** *If the domain of f is shifted by a constant amount, so will the choice of $\tilde{x}$, i.e. $g(a + \Delta, b + \Delta, y_a, y_b, \epsilon) = g(a, b, y_a, y_b, \epsilon) + \Delta$.*
$\mathcal{P}_3$ **(Domain Dilation)** *If the domain of f is expanded (or contracted) by a constant amount, so will the choice of $\tilde{x}$, i.e. $g(\kappa a, \kappa b, y_a, y_b, \kappa\epsilon) = \kappa g(a, b, y_a, y_b, \epsilon)$.*
$\mathcal{P}_4$ **(Range Rescaling)** *The choice of $\tilde{x}$ is invariant to re-scaling of the range of f, i.e. $g(a, b, \kappa y_a, \kappa y_b, \epsilon) = g(a, b, y_a, y_b, \epsilon)$.*
$\mathcal{P}_5$ **(Range Orientation)** *The choice of $\tilde{x}$ is consistent with the orientation in the range of f, i.e. $g(-1, 1, y_a, y_b, \epsilon) = -g(-1, 1, -y_b, -y_a, \epsilon)$.*

Premise $\mathcal{P}_1$ describes the choice of $\tilde{x}$ as a function of the information on the extremities a and b together with the stopping criteria ϵ. The limitation entailed in $\mathcal{P}_1$ is that it does not allow for the use of auxiliary points, nor does it allow for the use of other data, such as data collected from previous iterations or the evaluation of derivatives of f. The main motivation for $\mathcal{P}_1$ is to "require no more information than the bisection method" since by Theorem 4 no more information is needed for worst case optimality; i.e. it can be viewed as an instance of Occam's razor principle. Premises $\mathcal{P}_2$ to $\mathcal{P}_5$ are scale-invariance premises. These describe the hypothesis that *no metric is assumed*, and thus, the types of methods of choice of $\tilde{x}$ that are desirable must be metric independent in both the domain and range of f. Before we present the following result, we provide one definition, namely, that a function h defined from $\mathbb{R}^2$ to $\mathbb{R}^2$ is anti-symmetric if $h(-u, v) = -h(u, v)$ for all u, v, then:

Theorem 7. *Assume that $\tilde{x}$ satisfies $\mathcal{P}_1$ to $\mathcal{P}_5$; then there exists an anti-symmetric function $h : \mathbb{R}^2 \to [-1, 1]$ such that:*

$$\tilde{x} = x_{1/2} + \frac{b-a}{2} h\left(\frac{y_b + y_a}{y_a - y_b}, \frac{b-a}{\epsilon}\right). \tag{2.18}$$

Proof. Using $\mathcal{P}_1$ and $\mathcal{P}_2$ we may rewrite $\tilde{x} = g(a, b, y_a, y_b, \epsilon)$ as $g\left(a - \frac{b-a}{2} + \frac{b-a}{2}, b - \frac{b-a}{2} + \frac{b-a}{2}, y_a, y_b, \epsilon\right)$ and thus

$$\tilde{x} = \tfrac{a+b}{2} + g\left(-\tfrac{b-a}{2}, \tfrac{b-a}{2}, y_a, y_b, \epsilon\right).$$

Now, from $\mathcal{P}_3$ we obtain $g\left(-\frac{b-a}{2}, \frac{b-a}{2}, y_a, y_b, \epsilon\right) = \frac{b-a}{2} g\left(-1, 1, y_a, y_b, \frac{2\epsilon}{b-a}\right)$; and thus

$$\tilde{x} = \tfrac{a+b}{2} + \tfrac{b-a}{2} g\left(-1, 1, y_a, y_b, \tfrac{2\epsilon}{b-a}\right).$$

Furthermore, from $\mathcal{P}_4$ we obtain $g\left(-1, 1, y_a, y_b, \frac{2\epsilon}{b-a}\right) = g\left(-1, 1, 1, y_b/y_a, \frac{2\epsilon}{b-a}\right)$ which gives us

$$\tilde{x} = \tfrac{a+b}{2} + \tfrac{b-a}{2} g\left(-1, 1, 1, \tfrac{y_b}{y_a}, \tfrac{2\epsilon}{b-a}\right).$$

This uniquely determines $\tilde{x}$ up to the function of two parameters $H(z, w) = g(-1, 1, 1, z, w)$, where $z = \frac{y_b}{y_a}$ and $w = \frac{2\epsilon}{b-a}$. All that is left is to algebraically rewrite $\frac{y_b}{y_a}$ as a function of $\frac{y_b + y_a}{y_a - y_b}$ which we omit for simplicity (as well as recognize that $\frac{2\epsilon}{b-a}$ is a function of $\frac{b-a}{\epsilon}$). This completes the proof that

$$\tilde{x} = x_{1/2} + \frac{b-a}{2} h\left(\frac{y_b + y_a}{y_a - y_b}, \frac{b-a}{\epsilon}\right).$$

The anti-symmetry of $h(\cdot, \cdot)$ follows imetidately from $\mathcal{P}_5$. □

Theorem 7 states that any metric-independent strategy that satisfies $\mathcal{P}_1$ must be of the form (2.18). It reduces the class of functions of g that map from $\mathbb{R}^4$ to $\mathbb{R}$ to a significantly smaller class that is completely determined up to one function h from $\mathbb{R}^2$ to $\mathbb{R}$. Notice that the bisection method, the regula-falsi and the ITP method are of this form however Ridder's method is not (and to the best of our knowledge, neither is Brent's method or other methods previously proposed in the literature). However the bisection method is recovered via reduction when $h(u, v) = 0$ and the Regula-falsi method is recovered when $h(u, v) = u$. The ITP method makes full use of both input parameters. *Theorem 7 thus provides yet one more elegant support for the use of the ITP method as it is the only known method to satisfy all 5 premises and be sensitive to both input parameters.*

Remark 4. Notice that since h in (2.18) is anti-symmetric, it will always satisfy $h(0, v) = 0$, i.e., when $|y_a| = |y_b|$ then $\tilde{x}$ is chosen to be $x_{1/2}$. This is quite intuitive, since, when $|y_a| = |y_b|$ one cannot distinguish a side, neither a nor b, that is more likely to be closer to the root. Conversely, when $|y_a| < |y_b|$, it is intuitive to expect a to be closer to the root than b, and thus one can expect to find that a choice of $\tilde{x} < x_{1/2}$ is wisest (or more efficient). This property is guaranteed by the ITP construction and also by the regula-falsi, however it is not ensured by Premises $\mathcal{P}_1$ to $\mathcal{P}_5$ alone (the bisection method does not obey this monotonicity property).

2.3 Numerical experiments

In this section we perform numerical experiments to compare the ITP method with other well known strategies. Our empirical tests will assume that $y_a = f(a)$ and $y_b = f(b)$ are given in advance and that the x axis has been scaled so that $a = -1, b = 1$. With these initial conditions we adopt $\epsilon = 10^{-10}$. Additional experiments are also provided in the appendix with further results.

We implement the ITP method with parameters $\kappa_1 = 0.1$, $\kappa_2 = 2$ and $n_{\max} = n_{1/2}$, and use the standard stopping criterion of $b-a \leq 2\epsilon$. As mentioned earlier, low values of κ_1 and high values of κ_2 are associated with a faster speed of convergence, however, if a and b are not initiated sufficiently close to x^* then the possibility of producing $\Delta_{n+1} > \Delta_n/2$ can make (2.11) rapidly degenerate to $x_{1/2}$ forfeiting the high speed of convergence. On the opposite end, high values of κ_1 and low values of κ_2 can achieve $\Delta_{n+1} \leq \Delta_n/2$ more easily, however with weaker asymptotic guarantees. Given that for our initial conditions Δ_0 is equal to 2, then with $\kappa_1 = 0.1$ and $\kappa_2 = 2$ we have that $\kappa_1 \Delta_0^{\kappa_2} = 0.4$ which corresponds to 20% of the initial interval. Hence, the ITP method initiates quite conservatively by producing $x_t = x_{1/2}$ for any value of x_f within a 20% radius of $x_{1/2}$, and by never producing x_t within a 20% distance of any of the extremities in the first iteration. Beyond the first iteration, the ITP method will behave more or less conservatively according to whether Δ_{k+1} is greater than or less than $\Delta_k/2$.

The bisection method will require at most $n_{1/2} = 34$ iterations to terminate under the initial conditions considered. Moreover, the first iteration of the bisection method chooses $\tilde{x} = 0$ and the remaining subproblem with $\Delta = 1$ requires at most 33 function evaluations to terminate. In fact any interval (a, b) of length in $\Delta \leq 1.7179869184$ also terminates in at most 33 iterations since $\epsilon 2^{34}$ is equal to 1.7179869184. Thus, in the first iteration, any value of $\tilde{x} \in I = [-0.7179869184, 0.7179869184]$ is also minmax optimal, and this is precisely what Theorem 5 produces. Suppose for the sake of argument that in the first iteration, the projection in the ITP strategy is given by $\tilde{x} = 0.7179869184$. Then, there are two possible situations: (i) if $f(\tilde{x}) > 0$ then we are left with an interval of length 1.7179869184 in which case equation (2.11) will from that iteration onward reduce to $x_{1/2}$ and thus terminate in 33 iterations; (ii) if $f(\tilde{x}) \leq 0$ then we are left with an interval of length 0.2820130816 for which at most 31 iterations are necessary to locate the root. In case (ii) we have reduced two iterations at no cost and in case (i) nothing is lost since a minmax optimal method terminates in no more than $n_{1/2}$ iterations.

2.3.1 Comparison with other methods

In the following, we present a table that shows the respective count of function evaluations[2] required by the ITP strategy with the aforementioned parameters and several well established methods known in the literature: Ridders, Regula-Falsi, Illinois, Secant and Matlab's fzero routine[3]. With the exception of Matlab's fzero and

[2] The table shows the number of function evaluations and not the number of iterations because the Ridders method performs two function evaluations per iteration.

[3] In this experiment, we employed the Matlab's fzero function in its Revision 1.1.12.3, from 2013/11/03. As stated in MathWork's documentation, the algo-

the secant method, all routines were implemented with the standard stopping criterion of $b - a \leq 2\epsilon$. Since the secant method does not attempt bracketing, it was implemented with the stopping criterion of $\min\{|f(a)|, |f(b)|\} \leq \epsilon$, and hence it is displayed separately. Matlab's fzero has by default an inbuilt hybrid stopping criterion that evaluates relative error in x or absolute error in x according to the working regime. Under the conditions of this experiment, all instances made use of the absolute error criterion, i.e. it's stopping criterion is equivalent to the remaining methods, and hence it is displayed jointly with the remainder. For more on Matlab's fzero internal working see Remark 5.

The experiments are performed on twenty four functions. The first twelve functions are infinitely differentiable and contain only one simple root over the domain $[-1, 1]$. With the exception of the two polynomials and the Weirstrass approximation, the regular functions are also strictly monotone within the domain $[-1, 1]$. The remaining twelve functions in Table 2.1 are irregular. The first four ones contain zeros with multiplicity higher than one, the following four contain discontinuities, and the last four ones contain multiple roots (some of which also contain discontinuities). See Remark 6 for more on the irregular functions.

Remark 5. By design, Matlab's fzero does not return $\hat{x} = x_{1/2}$ and instead it returns a or b depending on the values of y_a or y_b. There are justifiable reasons for this choice, however, it may (if unlucky) produce a final estimate $\hat{x}$ that does not satisfy $|\hat{x} - x^*| \leq \epsilon$ despite $b - a \leq 2\epsilon$. Typically, the standard stopping criterion $b - a \leq 2\epsilon$ is motivated by the fact that the absolute error of $\hat{x} = x_{1/2}$ is less than or equal to ϵ, hence if our comparison between methods required that the estimate $\hat{x}$ produced by Matlab satisfied $|\hat{x} - x^*| \leq \epsilon$, then Matlab's reported function count in Tables 2.1 and 2.2 would increase. Thus, strictly speaking, what we provide should be treated as a lower bound on Matlab's function count. And therefore, to better compare Matlab's performance with respect to the other methods, we keep track of the final estimate provided by Matlab and indicate if the last step's "gamble" was successful in producing $|\hat{x} - x^*| \leq \epsilon$ or not in Table 2.1. We denote with '*' each unsuccessful run.

Remark 6. We point out that function count while searching ill-behaved functions may reflect different aspects of the overall performance of a method. For example, when searching an interval with multiple roots, different methods need not converge to the same solution, and hence one is not necessarily measuring only the "speed of convergence". Rather, what is being measured with function count is a combination of the number of iterations a method takes to isolate one root plus the iterations it takes in converging to the isolated root, i.e. a root-searching equivalent of the well known trade-off between exploration versus exploitation. Previous authors have also recognized the importance of evaluating non-linear solvers on functions with these types of irregularities (Rice, 1969; Nerinckx & Haegemans, 1976) due to their recurrence in applications.

---

rithm, created by T. Dekker, uses a combination of bisection, secant and inverse quadratic interpolation methods. An Algol 60 version, with some improvements, is given by R. P. Brent in "Algorithms for Minimization Without Derivatives", Prentice-Hall, 1973. The fzero routine is based on a Fortran version presented in Forsythe, Malcolm and Moler, "Computer Methods for Mathematical Computations", Prentice-Hall, 1976.

Table 2.1: Number of function evaluations required to solve (2.1) for various regular (the first group of twelve functions) and non-regular functions (the second group of twelve functions) where binary searching would require exactly 34 function evaluations. We mark '∞' any run that reached our cap of 10^4 function evaluations without solving (2.1), and, we mark with an '*' the instances where Matlab made a premature stop providing an estimate $\hat{x}$ that does not satisfy the specified condition $|\hat{x} - x^*| \leq \epsilon$.

		Reg.F.	Illinois	Matlab	Ridders	ITP	Secant
Well Behaved Func.							
Lambert	$xe^x - 1$	32	9	8	8	8	10
Trigonometric 1	$\tan(x - \frac{1}{10})$	47	6	7	10	8	6
Trigonometric 2	$\sin(x) + \frac{1}{2}$	11	7	6	14	8	6
Polynomial 1	$4x^5 + x^2 + 1$	∞	12	8	14	18	8
Polynomial 2	$x + x^{10} - 1$	∞	11	10	10	16	13
Exponential	$\pi^x - e$	15	8	7	8	8	7
Logarithmic	$\ln\lvert x - \frac{10}{9}\rvert$	71	10	8	10	7	9
Posynomial	$\frac{1}{3} + \text{sign}(x)\lvert x\rvert^{1/3} + x^3$	∞	15	8	24	32	10
Weierstrass	$\frac{1}{10^3} + \sum_{i=1}^{10} \sin(\frac{\pi i^3 x}{2})/\pi i^3$	∞	11	9	16	9	11
Poly.Frac.	$(x + \frac{2}{3})/(x + \frac{101}{100})$	∞	13	10	14	21	∞
Normal CDF	$\Phi(x - 1) - \sqrt{2}/4$	13	8	6	16	8	7
Normal PDF	$\phi(x - 1) - \sqrt{2}/4$	19	10	8	14	8	9
Ill-Behaved Func.							
# Non-simple Zero:							
Polynomial 3	$(x10^6 - 1)^3$	∞	144	63	62	34	33
Exp.Poly.	$e^x(x10^6 - 1)^3$	∞	146	93	62	34	77
Tan.Poly.	$(x-\frac{1}{3})^2 \tan^{-1}(x-\frac{1}{3})$	∞	110	95	56	34	27
Circles	$\text{sign}(3x+1)\left(1-\sqrt{1-(3x+1)^2/9^2}\right)$	∞	26	64	36	23	21
# Discontinuous:							
Step Function	$(x > \frac{1-10^6}{10^6})(\frac{1+10^6}{10^6}) - 1$	∞	155	36	68	34	∞
Geometric	$\frac{1}{21x-1}(x \neq \frac{1}{21})$	∞	69	35*	46	34	46
Trunc.Poly.	$\left(\frac{x}{2}\right)^2 + \lceil \frac{x}{2} \rceil - \frac{1}{2}$	34	38	33	26	34	14
Staircase	$\lceil 10x - 1 \rceil + \frac{1}{2}$	31	34	29*	24	31	∞
# Multiple Roots:							
Noisy Line	$x + \frac{\sin(x10^6)}{10} + \frac{1}{10^3}$	25	20	18	24	19	104
Warsaw	$(x > -1)(1 + \sin(\frac{1}{x+1})) - 1$	21	12	9	16	12	49
Sawtooth	$202x - 2\left\lfloor \frac{2x+10^{-2}}{2 \cdot 10^{-2}} \right\rfloor - \frac{1}{10}$	2	2	2	6	10	2
Sawtooth Cube	$\left(202x - 2\left\lfloor \frac{2x+10^{-2}}{2 \cdot 10^{-2}} \right\rfloor - \frac{1}{10}\right)^3$	∞	96	78	42	34	4
Global Average		∞	**40.5**	**27.1**	**26.1**	**20.2**	∞
Global Worst Case		∞	**155**	**95**	**68**	**34**	∞

Table 2.2: Central Tendency Metrics. An empirical measure of the mode, the median and the mean of each convergent method is displayed. Since the mode is non-unique in several cases, we display the mid-range of the smallest interval with a unique maximum frequency count within the interval.

	Reg.F.	Illinois	Matlab	Ridders	ITP	Secant
Well Behaved Functions -						
Mode	∞	10.5	8	14	8	8
Median	59	10	8	14	8	9.5
Mean	∞	10.0	7.9	13.2	12.6	∞
Ill-Behaved Functions -						
Mode	∞	145	34.5	25	34	∞
Median	∞	82.5	35.5	39	34	39.5
Mean	∞	71.0	46.3	39.0	27.8	∞

The last row of Table 2.1 summarizes the global performance of each method by providing the empirical average and the empirical worst case over all runs. When considering all functions a clear correlation between the global average and the global worst can be observed since the ranking of performance with respect to average or with respect to worst case produce identical results. The ITP method required the least amount of function evaluations followed by Ridders, Matlab, Illinois and then Regula Falsi respectively under both criteria. With respect to worst case Ridders method required twice as many function evaluations than the ITP method, Matlab required almost three times the number of function evaluations as the ITP method and Illinois neared four and a half times the number of function evaluations as the ITP method. The ITP method, with respect to worst case, required the same number of iterations as the bisection method as predicted by Theorem 5. Now, with respect to the global average Ridders method required roughly 30% more function evaluations than the ITP method, Matlab required roughly 34% more function evaluations than the ITP method and Illinois roughly twice the number of iterations of the ITP method. The bisection method, when compared to the ITP method, required roughly 70% more iterations than it's counterpart. It is worth noticing that in a multi-objective sense only the ITP method truly outperformed the bisection method since every other method was outperformed by the bisection method in at least 25% of the functions considered. One possible (informal) explanation for the observed correlation between global worst case and global average is that the fraction π of instances of (2.1) that require interpolation based methods to fall back to their worst case guarantees are the dominant term in the global average. This seems to be the case for small values of ϵ when n_{av} is of the order of $n_{av} \approx \pi n_{\text{worst}} + (1-\pi)n_{\text{asymptotic}} \approx \pi(C_1 \log \frac{b-a}{\epsilon}) + (1-\pi)(C_2 \log\log \frac{b-a}{\epsilon})$ as is somewhat expected for interpolation based methods, i.e. for averages of this form we have that $n_{av} \sim \pi(C_1 \log \frac{b-a}{\epsilon})$ for small values of $\epsilon/(b-a)$.

It can be observed in Table 2.1 that the behavior of the algorithms when dealing with the regular functions is much different from their behavior when non-regular

functions are tackled. All algorithms present a much better performance for the regular functions. Table 2.2 provides three central tendency measures for each method over each subset of functions: the Mean is estimated by averaging the number of iterations required in each experiment; the Median is estimated by averaging the two mid-values within each group; and, the Mode is estimated by taking the most recurrent number within the experiments, and, when this is not unique we take the mid-range of the interval which is simultaneously minimal and contains the most amount of numbers produced by the experiment (eg. the mode of $\{0, 1, 1\}$ is 1 and of $\{0, 1, 1, 2, 2\}$ is 1.5).

Well Behaved Functions

When considering only well behaved functions, Matlab and the ITP method seem to attain similar performances (Matlab having a slight advantage if the numbers are taken for their face value and the ITP method having a slight advantage if Matlab's numbers are taken as lower bound as explained in Remark 5). The ITP and Matlab achieved the lowest mode and median simultaneously, while the mean was minimized by Matlab alone. With respect to the mean we find that Illinois method attained a performance between Matlab and ITP and was the next best method under mode and mean metrics. Depending on the metric considered the ITP method required roughly from 24% to 37% the number of function evaluations as the bisection, and, Ridders method (the worst amongst the four methods that converged on all instances) requires roughly from 39% to 41%. The Regula-Falsi performed considerably worse than the other methods, and the well known fact that the regula-falsi may never reach the stopping criterion $b - a \leq 2\epsilon$ for locally convex or concave functions is observed in practice quite often. Somewhat unexpectedly, the secant method did not converge for one regular function which considerably impaired it's performance, however it attains a mode of 8.0 and a median of 9.5 which is close in performance to the ITP method and to Matlab's fzero. We conclude from this experiment on regular functions that *for well behaved functions the ITP method provides a performance that is competitive with state of the art methods* under different central tendency metrics.

Ill-Behaved Functions

For ill-behaved functions, the ITP method seems to outperform the other methods with respect to central tendency metrics. It achieves the lowest median, which is equal to the bisection, and the lowest mean, which is roughly twenty percent less than bisection. Ridders method attained the lowest mode over ill-behaved functions, however, it performed roughly 40% more iterations than the ITP method on the average. Illinois performed very poorly under all metrics requiring more than twice the number of iterations as the bisection on the average being followed by the secant method and the regula falsi which did not converge on many instances. The Matlab's fzero and Ridders method (the second and third best in the category) required more iterations than the bisection on most instances (roughly 60% of instances) and Matlab performed worse than the bisection method when evaluated with any central tendency metrics (from 1% to 36% worse than the bisection). From evaluating the performance of methods over this subset of functions we conclude that *over ill-behaved functions the ITP method provides significant improvements in performance when compared to state-of-the art methods* under different central tendency metrics.

Of course, if a different set of functions had been chosen for this experiment, the figures presented in the Table 2.1 would be different. It should be noticed that the relative ordering of averages may be strongly modified by operations of removal of a single row, which means that the averages should not be interpreted as if they meant the expected average performance of the methods under some notion of distributions over instances of (2.1). Due to the intrinsic difficulties involved in the construction of proper samplings from distributions of functions, no attempt is developed here for finding a numerical approximation of the average performances of the methods over any such a distribution. In this way, the averages presented in Table 2.1 should be interpreted only as rough proxies of the "average performances" of the algorithms.

In order to understand in greater detail the differences between the convergence patterns of the algorithms, another set of runs is performed. As the Regula-Falsi was found to be clearly worse than the other algorithms, it is not included in this analysis. In first place, the convergence patterns in the case of regular functions is studied, so the following functions that were used in the former experiment are employed for this comparison: Lambert, Trigonometric 2, Exponential and Logarithm. Once again, the initial (a, b) interval is set with $a = -1$ and $b = 1$. In this experiment, all the algorithms employ an as stopping criterion the condition $|\hat{x} - x^*| < 10^{-14}$, in which $\hat{x}$ represents the best current estimate of the solution and x^* represents the exact solution (which was previously recorded before the algorithm execution). In the case of the ITP, Ridders and the Illinois algorithms, $\hat{x}$ is calculated using the expression of x_f for the current interval extremes a and b. In the case of the Secant and Matlab's fzero, $\hat{x}$ is the current solution of the method. Again, the ITP algorithm parameters are defined as: $\kappa_1 = 0.1$ and $\kappa_2 = 2$. The evolution of the indicator $|\hat{x}(k) - x^*|$ for all algorithms on the four functions is shown in Figure 2.4.

The results of this experiment may be sumarized as following:

- The main conclusion we make from this experiment is that, once again, most of the algorithms considered here behave similarly in the set of regular functions employed in this study. For instance, the number of function evaluations for reaching error levels of 10^{-5}, 10^{-10} and 10^{-15} are similar for most algorithms in most problem instances.
- As noticeable exceptions, the Ridders algorithm is significantly better than the other ones in the case of the Lambert function and significantly worse than the other ones for Trigonometric 2 function, and the Secant algorithm is significantly worse than the other ones in the case of Lambert function.
- A superlinear convergence trend can be inferred in all cases. Further experiments, not represented in Figure 2.4, indicate that the relation $|\hat{x}(k+1) - x^*| \approx \alpha|\hat{x}(k) - x^*|^\beta$ fits the curves in that figure for β not smaller than 1.3, not greater than 1.8, and nearly equal to 1.6 for most curves.

Finally, an experiment is presented in order to characterize the convergence patterns of the ITP, in comparison with other algorithms, in the case of non-regular functions. The Figure 2.5 presents the evolution of indicators $\log(|a(k) - x^*|)$ and $\log(|b(k) - x^*|)$, for the algorithms ITP and Ridders, and $\log(|\hat{x} - x^*|)$, for Matlab's fzero algorithm, as functions of the function evaluation count k, when the algorithms are run on the function Polynomial 3, which has a root with multiplicity three in the search interval. Once again, the initial (a, b) interval is set with $a = -1$ and $b = 1$. The ITP algorithm employs the condition $|b(k) - a(k)| < 10^{-14}$ as the stopping criterion, and the other algorithms are stopped when $k = 50$. The ITP

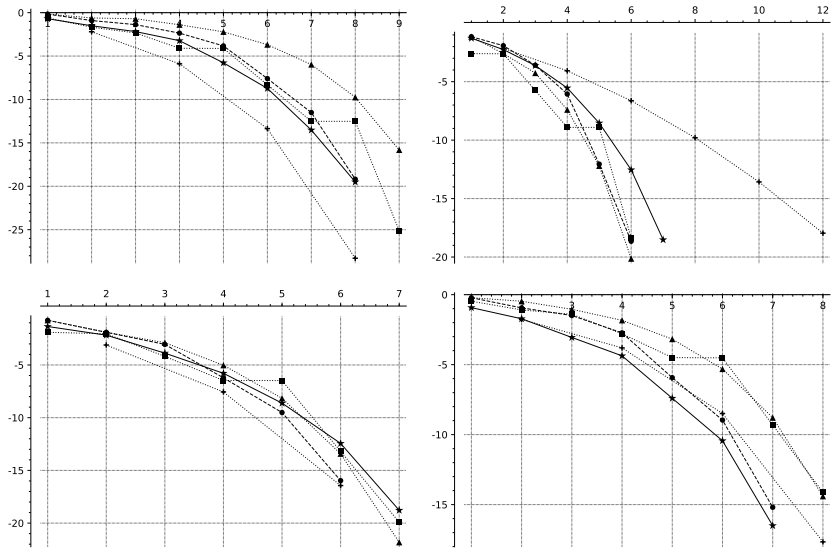


Fig. 2.4: Evolution of $\log|\hat{x} - x^*|$, for the methods: ITP (★), Ridders (+), Illinois (■), Secant (▲) and Matlab's fzero (●) in one run of each algorithm, for the functions: "Lambert" (top left), "Trigonometric 2" (top right), "Exponential" (bottom left), "Logarithm" (bottom right). The stopping criterion was $|\hat{x} - x^*| < 10^{-14}$. The ITP parameters are set as $\kappa_1 = 0.1$ and $\kappa_2 = 2$. The function evaluation count k is represented in the horizontal axis.

algorithm parameters are defined as: $\kappa_1 = 0.1$ and $\kappa_2 = 2$. The reference line $d(k) = \log(2 \cdot (1/2)^k)$, which indicates the theoretical rate of convergence of the bisection procedure, is also displayed in the figure.

It can be seen in Figure 2.5 that the behavior of the algorithms becomes rather different in this case, when compared with the regular function cases. Regimes of 'cautious steps' appear, as the algorithms detect that interpolation information is not reliable. In the case of ITP, this cautious regime approximately follows the same convergence rate of the bisection method. On the other hand, the Matlab's fzero and the Ridders algorithms converge at much slower rates, as a consequence of the trade-offs adopted in their design, which exchanged possible worst-case guarantees for enhanced asymptotic performances.

2.4 Discussion

In this chapter we have identified a class of root-searching methods that outperform the bisection method on the average performance while incurring in no additional cost on the worst-case performance. We both characterize that class and verify, somewhat surprisingly, that the traditional bisection method performs poorly on

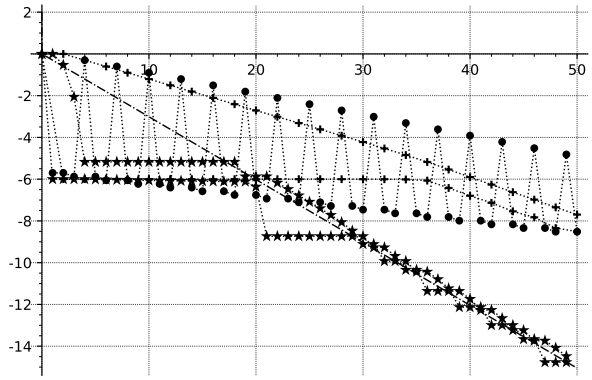


Fig. 2.5: Evolution of $\log(|a(k) - x^*|)$ and $\log(|b(k) - x^*|)$, for the methods ITP (★) and Ridders (+), and of $\log|\hat{x} - x^*|$ for Matlab's fzero (•), in one run of each algorithm, for the function "Polynomial". The stopping criterion for the ITP method was $|b(k) - a(k)| < 10^{-14}$, and for the other methods $k > 50$ was adopted. The ITP parameters are set as $\kappa_1 = 0.1$ and $\kappa_2 = 2$. The dot-dash line represents the reference line $d(k) = \log(2 \cdot (1/2)^k)$. The function evaluation count k is represented in the horizontal axis.

the average when compared to other strategies within that class. Worst case performance is taken over functions whose signal changes on a single point $x^* \in (a, b)$ and the average result holds for any distribution over instances of (2.1) that induces a continuous distribution $\mathcal{D}^*$ over (a, b) such that $x^* \sim \mathcal{D}^*$. The precise statements of those findings are given in Theorem 5 and Corollary 1.

Following our characterization of the class of minmax strategies we then identified a specific rule, the ITP strategy, for constructing a more efficient bracketing algorithm. As proven in Theorem 6, our method can attain an order of convergence of up to 1.618, i.e. the same as the secant method, while retaining the minmax optimality of the bisection method. In summary, the method is comprised of three steps: **I**nterpolation, **T**runcation and **P**rojection, which transform the traditional Regula-Falsi method into a *minmax* strategy with superlinear convergence.

Empirical results suggest that large gains can be expected, in terms of average performance, when compared to both the traditional bisection method as well as state of the art solvers. In the experiments presented here, the ITP required an average of 24% to 37% of the total number of iterations required by the bisection procedure over well behaved functions – a performance similar to that presented by most other methods with superlinear convergence. And, for ill-behaved functions, the average number of iterations required by ITP was roughly 82% of that required by bisection and at most the same in the worst case – no other state of the art method was capable of even matching the bisection method on the average. Hence, the ITP method seems to outperform state of the art solvers in terms of empirical performance when several metrics and regimes of operation are taken into consideration. Furthermore, when taking into account the theoretical guarantees provided by the ITP method, it seems to be a viable improvement on current practices in the

crafting of numerical solvers. And, since the recommended strategy makes use of no more information than the aforementioned methods, the increase in performance comes at no cost other than the computation of $\tilde{x}$ itself.

Historically, the bisection method has been the default method of choice when little is known about the underlying function being investigated. This preference is due to the minmax optimality of the traditional bisection method. Our results, however, show that this also has often been an inefficient choice. Here, we identified one specific viable substitute to the bisection method, the ITP, that strictly outperforms the traditional approach, concerning the expected number of iterations for reaching the solution, while preserving the same minmax optimality. However, a perhaps more important contribution of this chapter is the identification of a relatively unexplored class of methods that cannot be outperformed in terms of worst-case performance and thus deserves further investigation for the development of even more efficient methods for future applications.

Future work

Other authors have reported that the stopping criteria can play a key role in the performance of root-searching methods (Novak, 1989). Future work may include investigating how different stopping criteria can further improve our findings. Preliminary experiments (not reported here) suggest us that the number of iterations can be substantially improved when the ITP parameters κ_1, κ_2 and $n_{\max}$ are fine-tuned together with stopping criteria. Also, discrete analogues of the root-searching problem (Perl, Itai, & Avni, 1978; Laber, Milidiú, & Pessoa, 2002), i.e. searching through sorted arrays, as well as one-dimensional maximum searching (Kiefer, 1953) can equally benefit if analogue results were to be obtained. As reflected in our numerical results, our construction of $\tilde{x}$ in Algorithm 0 can benefit from smooth functions in which x_f is a good predictor; however, it may be possible to improve average performance by modeling the root-searching problem as a game theoretic model. This can be done by finding the Nash equilibrium for a game in which the choice of $\mathcal{D}^*$is performed by an adversarial agent and the choice of the root searching strategy by an agent that wishes to minimize the average number of iterations.

2.A Appendix

2.A.1 Proof of Theorem 6

The following lemma describes the worst case behavior of Algorithm 0 when implemented with $\tilde{x}$ equal to x_t. This lemma is used within the proof of Theorem 6.

Lemma 1. *Given that $\tilde{x}$ is taken to be equal to x_t in line (i) of Algorithm 0 with $\kappa_2 > 1$, then the residual Δ_n will satisfy $\Delta_n \leq 2/(2\kappa_1 + \kappa_1(\kappa_2 - 1)(n - n_0))^{1/(\kappa_2 - 1)}$ for all $n \geq n_0$ where $n_0 \equiv \lfloor \log_2 \Delta_0 + \frac{1}{\kappa_2 - 1} \log_2(2\kappa_1) \rfloor_+$.*

Proof. In each step, the worst possible reduction in bracketing error Δ_n happens when x_f is (very) close to one of the endpoints a or b and the reduction in the bracketing error Δ is all due to the perturbation δ towards the center. Notice that

when the perturbation δ is greater than $|x_{1/2} - x_f|$ then x_t is taken to be equal to $x_{1/2}$, and, if x_f is close to one of the endpoints a or b this will happen only when $\kappa_1 \Delta_n^{\kappa_2}$ is greater than or equal to $\Delta_n/2$. Thus, for as long as

$$\kappa_1 \Delta_n^{\kappa_2} \geq \Delta_n/2,$$

Algorithm 0 must behave identically to the bisection method. Hence, combining this with the fact that $\Delta_n = \Delta_0/2^n$ we find that if $\kappa_1 \Delta_0^{\kappa_2} \geq \Delta_0/2$ then $x_t = x_{1/2}$ for every n that satisfies:

$$n \leq \log_2 \Delta_0 + \tfrac{1}{\kappa_2 - 1} \log_2(2\kappa_1);$$

and if $\kappa_1 \Delta_0^{\kappa_2} < \Delta_0/2$ then x_t never coincides with $x_{1/2}$ for x_f near the extremities a or b. Thus, large values of κ_1 and small values of κ_2 might enforce several bisection-type steps, while sufficiently small values of Δ_0 can enforce that no bisection-type steps will take place. In either case, for $n \leq n_0$ where $n_0 \equiv \lfloor \log_2 \Delta_0 + \frac{1}{\kappa_2 - 1} \log_2(2\kappa_1) \rfloor_+$ we will have that $\Delta_n \leq \Delta_0/2^n$.

Now, when $n > n_0$ and x_f is near one of the extremities a or b, the perturbation δ will produce x_t that differs from $x_{1/2}$. Thus we find that in the worst case, for $n > n_0$, we have

$$\Delta_{n+1} = \Delta_n - \kappa_1 \Delta_n^{\kappa_2}. \tag{2.19}$$

Notice that the residual Δ_n can be seen as a function from $\mathbb{N}$ to $\mathbb{R}$, in other words $\Delta_n = \Delta(n)$. Consider now a continuation of Δ_n on the real line, i.e. a function $\Delta : \mathbb{R} \to \mathbb{R}$ such that $\Delta(n) = \Delta_n$. It is easy to see that $\Delta : \mathbb{R} \to \mathbb{R}$ can easily be made continuously differentiable. Also, if Δ_n satisfies (2.19) then Δ_n is a strictly decreasing sequence and so the function $\Delta(t)$ may also be constructed to be decreasing, and, since $\Delta_n \geq 0$ then we have $\Delta(t) \geq 0$ for all t. Notice that these two properties imply that $|\Delta'(t)|$ must vanish for increasing values of t; combining this with the fact that in (2.19) we have that $|\Delta_{n+1} - \Delta_n|$ is strictly decreasing, we may then guarantee that the continuation $\Delta(t)$ will also have $|\Delta'(t)|$ strictly decreasing. Hence we may derive that:

$$-\kappa_1 \Delta(t)^{\kappa_2} = \Delta(t+1) - \Delta(t) = \int_t^{t+1} \Delta'(\tau) d\tau \geq \min_{\tau \in (t, t+1)} \Delta'(\tau) = \Delta'(t).$$

This leads to the following differential inequality:

$$-\kappa_1 \Delta(t)^{\kappa_2} \geq \Delta'(t), \tag{2.20}$$

that holds for $t > t_0 \equiv n_0$. Defining $y = \Delta$ we have that:

$$-\kappa_1 y^{\kappa_2} \geq \frac{dy}{dt} \quad \Longrightarrow \quad -dt \geq \frac{dy}{\kappa_1 y^{\kappa_2}}$$

thus,

$$\int_{t_0}^{t} -dt \geq \int_{y_0}^{y_t} \frac{dy}{\kappa_1 y^{\kappa_2}} \quad \Longrightarrow \quad -t + t_0 \geq \frac{1}{-\kappa_1(\kappa_2 - 1) y_t^{(\kappa_2 - 1)}} + \frac{1}{\kappa_1(\kappa_2 - 1) y_0^{(\kappa_2 - 1)}}$$

from which we find by isolating y (and substituting $y = \Delta$ and $t = n$):

$$\Delta_n \leq \Delta_{n_0} \; / \; \Big(1 + \kappa_1(\kappa_2 - 1) \Delta_{n_0}^{(\kappa_2 - 1)} (n - n_0)\Big)^{1/(\kappa_2 - 1)}. \tag{2.21}$$

Notice now that Δ_{n_0} must be equal to $\Delta_0/2^{n_0}$ since in the worst case we will never have $f(\tilde{x}) = 0$ in any iteration, and so

$$\Delta_{n_0} = \frac{\Delta_0}{2^{\lfloor \log_2 \Delta_0 + (\log_2 2\kappa_1)/(\kappa_2 - 1) \rfloor}} \leq \frac{2\Delta_0}{2^{\log_2 \Delta_0 + (\log_2 2\kappa_1)/(\kappa_2 - 1)}} = \frac{2}{(2\kappa_1)^{1/(\kappa_2 - 1)}};$$

and also,

$$\Delta_{n_0} = \Delta_0/2^{\lfloor \log_2 \Delta_0 + (\log_2 2\kappa_1)/(\kappa_2 - 1) \rfloor} \geq 1/(2\kappa_1)^{1/(\kappa_2 - 1)}.$$

Combining these inequalities with (2.21) we have

$$\Delta_n \leq 2/(2\kappa_1)^{1/(\kappa_2 - 1)} \; / \; \Big(1 + \kappa_1(\kappa_2 - 1)(1/(2\kappa_1)^{1/(\kappa_2 - 1)})^{(\kappa_2 - 1)}(n - n_0)\Big)^{1/(\kappa_2 - 1)};$$

which simplifies to $\Delta_n \leq 2/(2\kappa_1 + \kappa_1(\kappa_2 - 1)(n - n_0))^{1/(\kappa_2 - 1)}$ as desired. □

*Proof (***Proof of Theorem 6:).**

The Lemma 1 has established the worst case behavior of Algorithm 0 when $\tilde{x} = x_t$. Now, we will analyze the asymptotic behavior of Algorithm 0. In order to do so we first recognize a simple property of Algorithm 0 that is a direct consequence of Lemma 1, namely that: if $\tilde{x} = x_t$, then neither of the end points a or b are retained indefinitely as the regula falsi. The proof of this claim follows immediately from Lemma 1 since if one of the endpoints were retained, then the residual would be lower bounded by either $x^* - a$ or $b - x^*$ depending on which endpoint is retained. We will see that for sufficiently large n, even if one of the endpoints is retained for several iterations, it must be updated eventually, and, for sufficiently large n the algorithm will enter a steady state in which the order of convergence becomes $\sqrt{\kappa_2}$.

It is well known that if $f \in \mathcal{C}^2$ and $f'(x^*) \neq 0$ then by linear interpolation bounds we can guarantee that $|x_f - x^*|$ satisfies

$$|x_f - x^*| = \left| \frac{f''(x^*)}{2f'(x^*)}(b - x^*)(x^* - a) \right| + O(b - a)^3;$$

which for simplicity we will represent as $|x_f - x^*| \sim |b - x^*||a - x^*|$. Since $|\tilde{x} - x_f| \leq \delta$ when $\tilde{x} = x_t$ we find that:

$$|\tilde{x} - x^*| \sim |b - x^*||a - x^*| + \kappa_1 |b - a|^{\kappa_2}.$$

Consider now a specific iteration $k = n - 1$ in which a and b are equal to (not necessarily in this order) $\tilde{x}_{n-1}$ and $\tilde{x}_{n-2}$, i.e. an iteration without retention. On that iteration we have:

$$\epsilon_n \sim \epsilon_{n-1}\epsilon_{n-2} + \kappa_1 \epsilon_{n-2}^{\kappa_2} \sim \epsilon_{n-1}\epsilon_{n-2} + \epsilon_{n-2}^{\kappa_2}. \tag{2.22}$$

If there exists a real number β such that $\epsilon_{n-1} \sim \epsilon_{n-2}^{\beta}$ then,

$$\epsilon_n \sim \epsilon_{n-1}^{\frac{\beta+1}{\beta}} + \epsilon_{n-1}^{\frac{\kappa_2}{\beta}}. \tag{2.23}$$

If $\kappa_2 \geq \beta + 1$ then $\epsilon_n \sim \epsilon_{n-1}^{\frac{\beta+1}{\beta}}$ which implies that $\beta = (\beta+1)/\beta$ and hence $\beta = \phi$. But from $\kappa_2 \geq \beta + 1$ we would find that $\kappa_2 \geq \phi + 1$, which leads to a contradiction since κ_2 was chosen to be in $[1, \phi + 1)$. Thus, it must be that $\kappa_2 < \beta + 1$ and so $\epsilon_n \sim \epsilon_{n-1}^{\frac{\kappa_2}{\beta}}$. This in turn implies that $\beta = \kappa_2/\beta$ and thus the order of convergence

must be $\beta = \sqrt{\kappa_2}$. Furthermore, plugging $\beta = \sqrt{\kappa_2}$ into the inequality $\kappa_2 < \beta + 1$ we find that $0 \leq \kappa_2 < \phi + 1$ which leads to no contradiction. Now, by combining the fact that $\epsilon_n \sim \kappa_1 \epsilon_{n-2}^{\kappa_2}$ and that $\epsilon_n \sim \kappa_3 \epsilon_{n-1}^{\sqrt{\kappa_2}}$ we then deduce that the asymptotic error constant κ_3 is equal to $\kappa_1^{1/(1+\sqrt{\kappa_2})}$ when $\kappa_2 > 1$. And, if $\kappa_2 = 1$ then, if (a, b) is initiated sufficiently close to x^*, from the definition of x_t we find that x_t will coincide with $x_{1/2}$, and for $x_{1/2}$ the asymptotic error constant is equal to $1/2$.

We emphasize that we just found that if (2.22) holds, then, the dominant term in (2.23) is the one associated with the perturbation δ instead of the estimation error of x_f, i.e. for sufficiently large n we will have that $\delta > |x_f - x^*|$. And, we also found that under this condition the only compatible value for the order of convergence is $\sqrt{\kappa_2}$. We now verify that if (2.22) holds for one step, then it will hold from that step onward. For this, since we have assumed that iteration $n-1$ was one in which neither of the endpoints is a "retained endpoint", assume without loss of generality that $a_{n-1} = \tilde{x}_{n-2}$ and $b_{n-1} = \tilde{x}_{n-1}$. Now, we will analyze three possible regions in which $\tilde{x}$ may fall: (i) If $\tilde{x} \in (x_{1/2}, b_{n-1})$ then a must be updated since $\delta > |x_f - x^*|$ implies that the bracketing update strategy will make $a = \tilde{x}$. Hence, no retention will occur in that iteration and (2.22) will hold for the next iteration; (ii) If $\tilde{x} \in (a_{n-1}, x_{1/2})$, then since the dominant term in (2.23) is the one associated to δ (and not the estimation error from x_f) we find that $x^* \leq x_f + \delta = \tilde{x} < x_{1/2}$ this shows that ϵ_{n-2} was in fact lesser than ϵ_{n-1}, and thus in the next iteration the error of $\tilde{x}_{n+1}$ will satisfy:

$$|\tilde{x}_{n+1} - x^*| = \epsilon_{n+1} \sim \epsilon_n \epsilon_{n-2} + \epsilon_{n-2}^{\kappa_2} \lesssim \epsilon_n \epsilon_{n-1} + \epsilon_{n-1}^{\kappa_2};$$

and so, once again (2.22) holds for the following step; (iii) Finally, if $\tilde{x} = x_{1/2}$ and a is updated then equation (2.22) must hold in the next iteration, and, if b is updated then that shows that $\epsilon_{n-2} < \epsilon_{n-1}$ and so as argued above, this implies that (2.22) holds in the next step, which completes the proof that $\tilde{x} = x_t$ produces a sequence $\tilde{x}_n$ that converges to x^* with an order of convergence of $\sqrt{\kappa_2}$. That the residuals Δ_n vanish with the same order follows from the fact that for sufficiently high values of n, since the truncation error δ is greater than the vanishing estimation error $|x_f - x^*|$, the upper and lower bounds a and b will be updated alternately; and so, the order of convergence of Δ_n is equal to the order of $\tilde{x}_{n-1}$, which is of course the same as the order of convergence of $\tilde{x}_n$.

What is left now is to verify that, under the super-linear convergence conditions of x_t, if (2.11) is wide enough then the projection of x_t onto the minmax disk will also converge with an order of convergence of $\sqrt{\kappa_2}$. To see this, we point out one fact about the steady state behavior of x_t, namely: that the truncation error δ will eventually become and remain larger than that of the estimation error $|x_f - x^*|$ and thus we are always left with the smallest amongst the intervals (a, x_t) or (x_t, b) in each iteration. Now, since the direction of the projection always agrees with that of the truncation step, then the same is also true of the projection of x_t onto (2.11). Hence, under the steady state conditions of x_t the residuals satisfy $\Delta_{k+1} \leq \Delta_k/2$ and, with a little algebra, we can show that the fraction of the interval (a_k, b_k) covered by (2.11), which we will denote by $F(k)$, increases from iteration k to iteration $k+1$, i.e. that

$$F(k+1) \equiv \frac{2(\epsilon 2^{n_{1/2}-(k+1)} - \Delta_{k+1}/2)}{\Delta_{k+1}} \geq \frac{2(\epsilon 2^{n_{1/2}-k} - \Delta_k/2)}{\Delta_k} \equiv F(k).$$

Furthermore, if $\tilde{x}$ is not equal to x_t, then it is the projection onto the minmax disk, i.e. $\tilde{x}_k = x_{1/2} \pm (\epsilon 2^{n_{1/2}-k} - \Delta_k/2)$. In which case since $\Delta_{k+1} \leq \Delta_k/2$ we must have that:

$$\Delta_{k+1} = \Delta_k - \epsilon 2^{n_{1/2}-k};$$

from which we derive that the fraction of the interval (a_{k+1}, b_{k+1}) covered by (2.11) is given by

$$F(k+1) \equiv \frac{2(\epsilon 2^{n_{1/2}-(k+1)} - \frac{\Delta_{k+1}}{2})}{\Delta_{k+1}} = \frac{2(\epsilon 2^{n_{1/2}-k} - \frac{\Delta_k}{2})}{\Delta_k} \times \frac{1}{1 - \frac{\epsilon 2^{n_{1/2}-k}}{\Delta_k}} > 2F(k).$$

Where the inequality follows from the fact that $\epsilon 2^{n_{1/2}-k}/\Delta_k$ is greater than $1/2$ given that $(b-a)/\epsilon$ is not a power of two. Thus, under the steady state conditions of x_t, the fraction of the interval (a_{k+1}, b_{k+1}) covered by (2.11) must at least double in each iteration if $\tilde{x} \neq x_t$. Hence, if the fraction of the interval (a_0, b_0) covered by (2.11) is not too small, it will take only a few iterations until $\tilde{x}$ can assume any value within (a, b), and will thus coincide with x_t from that iteration onward. □

2.A.2 A pseudocode for the ITP method

As recommended by a referee, we provide a "line-by-line" description of the ITP method algorithm for ease of read and implementation. This particular implementation does not receive y_a and y_b as inputs (and thus must query $f(a)$ and $f(b)$ to initiate), it adopts the relaxation of $n_{\max} = n_{1/2} + n_0$ (and hence no need for the "$\mathcal{I}_0$ not too small" condition if $n_0 > 0$), and returns the new values of a and b obtained with $b - a \leq 2\epsilon$ (instead of only one estimator $\hat{x}$).

Algorithm 1: ITP_Method$(a, b, \epsilon, n_0, \kappa_1, \kappa_2, f)$

Input: $a, b, \epsilon, n_0, \kappa_1, \kappa_2, f$

/* where: $a \leq b$, $0 < \epsilon$, $0 \leq n_0$, $0 < \kappa_1$, $1 \leq \kappa_2 < 1 + \phi$, $f(a) < 0 < f(b)$ */

$y_a = f(a)$;

$y_b = f(b)$;

$n_{1/2} = \lceil \log_2 \frac{b-a}{2\epsilon} \rceil$;

$n_{\max} = n_{1/2} + n_0$;

$k = 0$;

while $b - a > 2\epsilon$ **do**

  /* Interpolation: */

  $x_f = \frac{y_b a - y_a b}{y_b - y_a}$;

  /* Truncation: */

  $x_{1/2} = \frac{a+b}{2}$;

  $\sigma = \text{sign}(x_{1/2} - x_f)$;

  $\delta = \kappa_1 (b-a)^{\kappa_2}$;

  if $\delta \leq |x_{1/2} - x_f|$ **then**

    $x_t = x_f + \sigma\delta$;

  else

    $x_t = x_{1/2}$;

  /* Projection: */

  $r = \epsilon 2^{n_{\max} - k} - (b-a)/2$;

  if $|x_t - x_{1/2}| \leq r$ **then**

    $\tilde{x} = x_t$;

  else

    $\tilde{x} = x_{1/2} - \sigma r$;

  /* Updating: */

  $\tilde{y} = f(\tilde{x})$;

  if $\tilde{y} > 0$ **then**

    $b = \tilde{x}$;

    $y_b = \tilde{y}$;

  else if $\tilde{y} < 0$ **then**

    $a = \tilde{x}$;

    $y_a = \tilde{y}$;

  else

    $a = \tilde{x}$;

    $b = \tilde{x}$;

  $k = k + 1$;

Output: a, b

When implementing the ITP Algorithm, we point out that special care must be put in controlling and/or mitigating numerical errors if minmax optimality is to be attained, specifically in the computation of r in line 15 when $n_0 = 0$. For example, if r', the numerical representation of r, satisfies $r' > r$ then the projection step in line 19 may produce a non-minmax $\tilde{x}$ and may come at the cost of additional iterations. Similarly, if $r' < r$, then when $r = 0$ we will have a negative representation r' which in turn may produce unpredictable behaviours in the projection step. Although this goes beyond the scope of this chapter we point out that simple numerical error controlling strategies (such as taking $r = r - TOL$ for some small $TOL > 0$ after line 15 to avoid $r' > r$ and taking $r = \max\{r, 0\}$ to avoid negative values for r) seem to work fine in avoiding such instabilities. Similar issues can be pointed out in the computation of $x_{1/2}, \delta, x_f$ and x_t.

2.A.3 Additional numerical experiments

We perform another set of experiments in order to examine the theoretical prediction of the order of convergence of ITP method, as provided by Theorem 6. The specific results presented here are obtained for the following function:

$$f(x) = (x - (100 + 10\pi))(x - 10\pi) \tag{2.24}$$

The root $x^* = 10\pi$ is searched within the interval defined by $a = -100$ and $b = 100$, and we assume that $f(a)$ and $f(b)$ are known *a priori*. In order to allow the algorithm to achieve a steady-state in which the asymptotic behavior becomes clearly identifiable, we adopt $\epsilon = 10^{-50}$. With such a small tolerance interval, the usual numerical precision provided by most computational environments is not enough. For this reason, we employ the SageMath environment, which allows computation with a specified numerical precision. All the results reported here are obtained using constants and variables with numerical precision of 1000 bits.

The algorithm parameters are defined as:

$$\kappa_1 = 0.1 \quad \kappa_2 = 0.98\left(1 + \frac{1+\sqrt{5}}{2}\right) = 0.98(1+\phi) \tag{2.25}$$

This κ_2 is 0.98 times the value which leads to the instance of ITP algorithm with the highest possible order of convergence, as predicted by Theorem 6. We employ this value instead of $(1 + \phi)$ in the experiment because the exact maximum value leads to sequences that do not reach a steady-state regime.

The definition of order of convergence is given by expression (2.7). However, as shown in the proof of Theorem 6, the asymptotic behavior achieved by algorithm ITP is composed of alternate updates of the interval extremes, a and b, with each of them achieving an order of convergence $\beta = \sqrt{\kappa_2}$. This means that their asymptotic behavior should be measured considering the comparison of $a(k)$ with $a(k + 2)$ and $b(k)$ with $b(k + 2)$. Notice that, as the order of convergence measured on one iteration should be $\beta = \sqrt{\kappa_2}$, the order of convergence measured over two consecutive iterations must be $\beta^2 = \kappa_2$. Similarly, the asymptotic error constant $\alpha = \kappa_1^{\frac{1}{1+\sqrt{\kappa_2}}}$ refers to one iteration, which means that for the composition of two successive iterations, the composite constant should be $\alpha \cdot \alpha^{\beta} = \kappa_1$. Therefore, we define the indicators $D_a(k)$ and $D_b(k)$:

$$D_a(k) = \frac{|a_{k+2} - x^*|}{|a_k - x^*|^{\kappa_2}} \qquad D_b(k) = \frac{|b_{k+2} - x^*|}{|b_k - x^*|^{\kappa_2}} \tag{2.26}$$

The Theorem 6 predicts the following:

$$\lim_{k\to\infty} D_a(k) = \lim_{k\to\infty} D_b(k) = \kappa_1 \tag{2.27}$$

Figure 2.6 shows the sequence of $D_a(k)$ and $D_b(k)$ over the iterations of algorithm ITP. This figure is consistent with the predictions of Theorem 6: $D_a(k)$ and $D_k(b)$ are built using exponent κ_2 in the denominator, and this specific exponent leads the respective sequences to achieve a constant value in steady-state. Also, this constant is equal to κ_1.

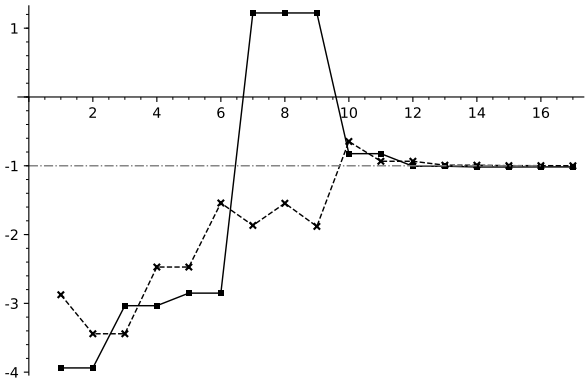


Fig. 2.6: Evolution of $\log(D_a(k))$, indicated by (■), and $\log(D_b(k))$, indicated by (×), in one run of algorithm ITP, considering function (2.24), $\epsilon = 10^{-50}$, $\kappa_1 = 0.1$ and $\kappa_2 = 0.98(1 + \phi)$. The value of $\log(\kappa_1)$ is indicated by the horizontal dash-dot line. The iteration count k is represented in the horizontal axis.

Figure 2.7 shows the evolution of indicators $rule(k)$ and $side(k)$, which are defined as:

$$rule(k) = \begin{cases} 1 & , \quad \text{if } x_t(k) = x_f + \sigma\delta \neq x_{1/2} \\ -1 & , \quad \text{if } x_t(k) = x_{1/2} \end{cases} \quad side(k) = \begin{cases} 0.5 & , \quad \text{if } a(k) > a(k-1) \\ -0.5 & , \quad \text{if } b(k) < b(k-1) \end{cases} \tag{2.28}$$

Figure 2.7 shows that, up to iteration 7, the value of x_t was given by the rule $x_t = x_{1/2}$. From iteration 8 on, the rule $x_t = x_f + \sigma\delta$ stops to degenerate into $x_{1/2}$, leading to non-bisection steps. From iteration 6 to 9, only the value of b was updated, while a was kept fixed. After iteration 9, the ITP algorithm started to alternate the updates in the values of a and b. Those are the conditions for reaching the steady-state regime in which the sequences $D_a(k)$ and $D_b(k)$ become asymptotically constant, as stated in the proof of Theorem 6. This is consistent with Figure 2.6,

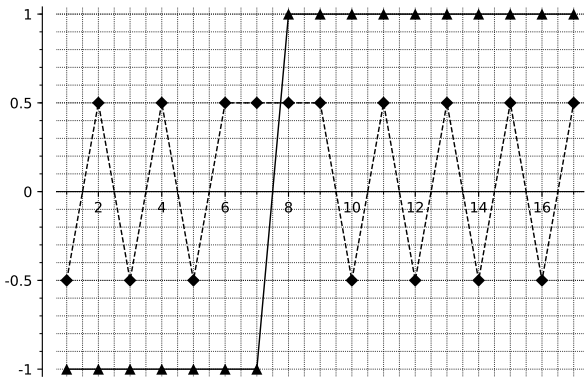


Fig. 2.7: $rule(k)$, indicated by (▲), is 1 in the steps when $x_t = x_f + \sigma\delta \neq x_{1/2}$ and -1 when $x_t = x_{1/2}$. $side(k)$, indicated by (⧫), is -0.5 in the iterations in which a is updated, and 0.5 in the iterations when b is updated. Results from one run of algorithm ITP, considering function (2.24), $\epsilon = 10^{-50}$, $\kappa_1 = 0.1$ and $\kappa_2 = 0.98(1 + \phi)$. The iteration count k is represented in the horizontal axis.

which shows the values of $D_a(k)$ and $D_k(b)$ starting to converge monotonically to their steady-state value after iteration 9, with alternate enhancements in a and b.

It is important to mention that we conducted several similar experiments, considering different functions $f(x)$, different values of a and b and different values of κ_1 and κ_2. The results of this set of experiments, when $f(x) \in \mathcal{C}^2$ and x^* is a root with multiplicity one, were consistent with the predictions of Theorem 6, in the sense that always existed a sufficiently small interval $[a, b]$ within which a steady-state regime arose as described in the former paragraph. In those cases, the values of β (the order of convergence) and α (the asymptotic error constant) were verified as predicted by Theorem 6. On the other hand, when $f(x)$ had a discontinuity on the root or when x^* had multiplicity greater than one, the algorithm ITP did not achieve a detectable steady-state regime.

References

Argyros, I. K., & Khattri, S. K. (2013). On the secant method. *Journal of Complexity*, *29*(6), 454-471.

Brent, R. P. (1971). An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, *14*(4), 422–425. doi: https://doi.org/10.1093/comjnl/14.4.422

Bus, J. C. P., & Dekker, T. J. (1975). Two efficient algorithms with guaranteed convergence for finding a zero of a function. *ACM Transactions on Mathematical Software*, *1*(4), 330–345. doi: https://doi.org/10.1145/355656.355659

Chapra, S. C., & Canale, R. P. (2010). Numerical methods for engineers. In (6th ed. ed., p. 202-220). New York, NY: McGraw-Hill Higher Education.

Dowell, M., & Jarratt, P. (1971, June). A modified regula falsi method for computing the root of an equation. *ACM Transactions on Mathematical Software*, *11*(2), 168-174. doi: https://doi.org/10.1007/BF01934364

Eiger, A., Sikorski, K., & Stenger, F. (1984). A bisection method for systems of nonlinear equations. *ACM Transactions on Mathematical Software*, *10*(4), 367-377. doi: https://doi.org/10.1145/2701.2705

Ford, J. A. (1995). *Improved algorithms of illinois-type for the numerical solution of nonlinear equations* (Department of Computer Science Report). University of Essex.

Gal, S., & Miranker, W. (1977). Optimal sequential and parallel search for finding a root. *Journal of Combinatorial Theory*, *23*(1), 1-14. doi: https://doi.org/10.1016/0097-3165(77)90074-7

Graf, S., Novak, E., & Papageorgiou, A. (1989). Bisection is not optimal on the average. *Numerische Mathematik*, *55*, 481-491. doi: https://doi.org/10.1007/BF01396051

Kearfott, R. B. (1987). Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, *13*(3), 197-220. doi: https://doi.org/10.1145/29380.29862

Kiefer, J. (1953). Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, *4*(3), 502-506.

Laber, E. S., Milidiú, R. L., & Pessoa, A. A. (2002). On binary searching with nonuniform costs. *SIAM Journal on Computing*, *31*(4), 1022-1047. doi: https://doi.org/10.1137/S0097539700381991

Le, D. (1985a). An efficient derivative-free method for solving nonlinear equations. *ACM Transactions on Mathematical Software*, *11*(3), 250-262. doi: https://doi.org/10.1145/214408.214416

Le, D. (1985b). Three new rapidly convergent algorithms for finding a zero of a function. *SIAM Journal on Scientific and Statistical Computing*, *6*(1), 193-208. doi: https://doi.org/10.1137/0906016

McNamee, J. M., & Pan, V. Y. (2012). Efficient polynomial root-refiners: A survey and new record efficiency estimates. *Computers & Mathematics with Applications*, *63*(1), 239-254. doi: https://doi.org/10.1016/j.camwa.2011.11.015

Muller, D. E. (1956). A method for solving algebraic equations using an automatic computer. *Mathematical Tables and Other Aids to Computation*, *10*(56), 208-215. doi: https://doi.org/10.2307/2001916

Nerinckx, D., & Haegemans, A. (1976). A comparison of non-linear equation solvers. *Journal of Computational and Applied Mathematics*, *2*(2), 145-148. doi: https://doi.org/10.1016/0771-050X(76)90017-6

Norton, V. (1985). Algorithm 631 finding a bracketed zero by larkin's method of rational interpolation. *ACM Transactions on Mathematical Software*, *11*(2), 120-134. doi: https://doi.org/10.1145/214392.214396

Novak, E. (1989). Average-case results for zero finding. *Journal of Complexity*, *5*(4), 489-501.

Novak, E., & Ritter, K. (1993). Some complexity results for zero finding for univariate functions. *Journal of Complexity*, *9*(1), 15-40. doi: https://doi.org/10.1006/jcom.1993.1003

Novak, E., Ritter, K., & Woźniakowski, H. (1995). Average-case optimality of a hybrid secant-bisection method. *Mathematics of Computation*, *64*(212), 1517–1539. doi: https://doi.org/10.2307/2153369

Oliveira, I. F. D., & Takahashi, R. H. C. (2020). An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software*.

Perl, Y., Itai, A., & Avni, H. (1978). Interpolation search—a log log n search. *Communications of the ACM*, *21*, 550-553. doi: https://doi.org/10.1145/359545.359557

Pownuk, A., & Kreinovich, V. (2017). Why linear interpolation? *Mathematical Structures and Modeling*, *3*(43), 43-49.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: the art of scientific computing. In (6th ed. ed., p. 442-486). Cambridge, UK: Cambridge University Press.

Rice, J. R. (1969). *A set of 74 test functions for nonlinear equation solvers* (Department of Computer Science Report No. 64-034). Purdue University.

Ridders, C. (1979). A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, *26*(11), 979–980. doi: https://doi.org/10.1109/TCS.1979.1084580

Ritter, K. (1994). Average errors for zero finding: lower bounds for smooth or monotone functions. *Aequationes Mathematicae*, *48*(2), 194-219. doi: https://doi.org/10.1007/BF01832985

Segura, J. (2010). Reliable computation of the zeros of solutions of second order linear odes using a fourth order method. *SIAM Journal on Numerical Analysis*, *48*(2), 452-469. doi: https://doi.org/10.1137/090747762

Shrager, R. I. (1985). A rapid robust rootfinder. *Mathematics of Computation*, *44*(169), 151-165. doi: https://doi.org/10.2307/2007799

Sikorski, K. (1982). Bisection is optimal. *Numerische Mathematik*, *40*(1), 111-117. doi: https://doi.org/10.1007/BF01459080

Sikorski, K. (1985). Optimal solution of nonlinear equations. *Journal of Complexity*, *1*(2), 197-209. doi: https://doi.org/10.1016/0885-064X(85)90011-1

Stage, S. A. (2013). Comments on an improvement to the brent's method. *International Journal of Experimental Algorithms*, *4*(1), 1-16.

Traub, J. F. (1963). Iterative methods for the solution of equations. *Bell Telephone Laboratories*, *8*(4), 550-551.

Vrahatis, M. N. (1988). Algorithm 666 chabis: A mathematical software package for locating and evaluating roots of systems of nonlinear equa-

tions. *ACM Transactions on Mathematical Software*, *14*(4), 330-336. doi: https://doi.org/10.1145/50063.51906

Wu, X. (2005). Improved muller method and bisection method with global and asymptotic superlinear convergence of both point and interval for solving nonlinear equations. *Applied Mathematics and Computation*, *166*(2), 299-311. doi: https://doi.org/10.1016/j.amc.2004.04.120

Yao, A. C., & Yao, F. F. (1976). The complexity of searching an ordered random table. *Proceedings of the Seventeenth Annual Symposium on Foundations of Computer Science*, 173-177. doi: https://doi.org/10.1109/SFCS.1976.32

Zhang, Z. (2011). An improvement to the brent's method. *International Journal of Experimental Algorithms*, *2*(1).

The list searching problem

Summary. This chapter, contained in pages 55 to 76, is based on the the paper *Minmax-optimal list searching with O(log2 log2 n) average cost* which was first submitted for consideration by a scientific journal on the 14th of September of 2020; and, as of the day of the submission of this thesis is currently under consideration for publication. The latest version of this paper (Oliveira & Takahashi, 2020b) can be found in the pre-print repository in the following address: `https://arxiv.org/abs/2105.11919`. In the current form presented below, minor changes are made to make for a more coherent read within the context of this thesis, however, it is kept self contained and an independent read of Chapter 3 is possible without need of reference to external material within the thesis. Any major content foreign to the original paper is highlighted with dark blue text, and, the remainder, where little or no alteration is made, we present the content in plain black text.

In this chapter we formally study the list searching problem introduced in Chapter 1. In particular we find a searching method on ordered lists that surprisingly outperforms binary searching with respect to average query complexity while retaining minmax optimality. The method is shown to require $O(\log_2 \log_2 n)$ queries on average while never exceeding $\lceil \log_2 n \rceil$ queries in the worst case, i.e. the minmax bound of binary searching. Our average results assume a uniform distribution hypothesis similar to those of previous authors under which the expected query complexity of interpolation search of $O(\log_2 \log_2 n)$ is known to be optimal. Hence our method turns out to be optimal with respect to both minmax and average performance. We further provide robustness guarantees and perform several numerical experiments with both artificial and real data. Our results suggest that time savings range roughly from a constant factor of 10% to 50% to a logarithmic factor spanning orders of magnitude when different metrics are considered.

3.1 Introduction

Given a sorted list $\boldsymbol{v}$ and a target value z, the problem of searching sorted lists is typically stated as:

$$\text{Find } k_* \text{ such that } v_{k_*} \leq z < v_{k_*+1}; \tag{3.1}$$

where $\boldsymbol{v}$ is of size $n+1$ with entries in $[0,1]$ and $v_0 = 0$, $v_n = 1$ and z a scalar in $(0,1)$. This problem is ubiquitous in computer science with applications spanning several different fields of computer programming, engineering and mathematics. Variations of (3.1) include searching unbounded lists (Bentley & Yao, 1976), tables (Knuth, 1998), searching continuous functions for a zero (Oliveira & Takahashi, 2020a), as well as the construction of insertions and deletion procedures in canonical data-structures (Bentley & Sedgewick, 1997).

The standard approach to solve (3.1), commonly known as *binary search* (Knuth, 1998), begins with and updates upper and lower bounds $a = 0$ and $b = n$ for the location of the desired index k_*. At each step this is done by recursively probing the index $k_{1/2}$ which is defined by rounding

$$x_{1/2} \equiv \frac{a+b}{2} \tag{3.2}$$

arbitrarily to the nearest integer, and, by comparing $v_{k_{1/2}}$ and z it updates a and b accordingly, i.e. if $v_{k_{1/2}} > z$ then b is updated to $b = k_{1/2}$ if $v_{k_{1/2}} < z$ then a is updated to $a = k_{1/2}$ and if $v_{k_{1/2}} = z$ then a and b are updated to $a = k_{1/2}$ and $b = k_{1/2} + 1$. The algorithm terminates when the tolerance $\Delta \equiv b - a$ is equal to one. For convenience we display below the general structure of the binary searching algorithm as a while-loop; however, binary searching also admits for-loop formulations and other formulations that exploit computer architecture (Schlegel, Gemulla, & Lehner, 2009; Cannizzo, 2018) to improve computational speed. Here, a and b are initiated at $a = 0, b = n$, and, in line (1) $\tilde{k}$ is taken to be equal to $k_{1/2}$.

Algorithm 0: The Bracketing Algorithm

Input: $\boldsymbol{v}$ and z
while $(b - a > 1)$, do:
(1). Choose $\tilde{k} \in (a, b)$ and evaluate $v_{\tilde{k}}$;
(2). Update a and b according to the values of $v_a, v_b, v_{\tilde{k}}$ and z;
Output: $k_* = a$

The key feature of binary search is it's *minmax optimality*. That is, it requires at most

$$N_{1/2} = \lceil \log_2 n \rceil \tag{3.3}$$

queries to locate k_* while no other method can provide the same guarantee in less than $N_{1/2}$ queries. This property is specifically of interest when the computational cost of one query is known to be much higher than the computation of the search procedure itself. This assumption is often made implicitly in the literature, and for the sake of clarity, it is assumed henceforth.

While binary search is optimal with respect to the worst case metric, *interpolation search* turns out to be a more efficient alternative with respect to expected query complexity if a uniform distribution is assumed, see (Yao & Yao, 1976; Perl & Reingold, 1977; Perl, Itai, & Avni, 1978). Interpolation search is a bracketing algorithm with $\tilde{k}$ in line (1) of Algorithm 0 defined as the linear interpolation between the points (a, v_a) and (b, v_b). More precisely $\tilde{k}$ is taken to be equal to k_f where k_f is defined as the integer closest to

$$x_f \equiv \frac{b \cdot (v_a - z) - a \cdot (v_b - z)}{v_a - v_b}, \tag{3.4}$$

that lies in between x_f and $x_{1/2}$. The key feature of interpolation search is that if the entries of $\boldsymbol{v}$ and z are sorted samples of a uniform distribution over $[0, 1]$, then, interpolation search is optimal with respect to expected query complexity (Yao & Yao, 1976) and the expected number of queries $\mathbb{E}(N)$ to solve (3.1) is

$$\mathbb{E}(N) = O(\log_2 \log_2 n), \tag{3.5}$$

which considerably improves on the expected query complexity of binary search.

Although interpolation search enjoys an improved average performance, the improvement comes at the cost of requiring up to n queries to terminate in the worst case. Furthermore, the guarantees on the expected query complexity of interpolation search do not hold if the distributional hypothesis is misspecified. Thus, choosing interpolation search over binary search may come at a high cost since interpolation search may also require a full series of n queries to terminate on the average under different distributions. Is it possible to simultaneously enjoy the benefits of interpolation search with no costs on the minmax peformance of binary search? And furthermore, is it possible to enjoy such benefits without trading off performance under misspecified conditions?

In this chapter we answer the above questions affirmatively. To answer the first question we begin by identifying the necessary and sufficient conditions for a searching method to be minmax optimal. Then, we pin-point one specific minmax method,

which we name the *Interpolation, Truncation and Projection Method*, or simply the *ITP Method*. We show that this method attains the expected query complexity of $O(\log_2 \log_2 n)$ queries under similar assumptions as those required by interpolation search; and, it requires no more iterations than the upperbound of $\lceil \log_2 n \rceil$ of binary searching. Hence, it is both optimal with respect to minmax and average performance at no cost other than the computation of the method itself. To answer the second question, we find lower bounds on the average performance of binary searching under very broad distributional hypothesis and show that the bisection method can never outperform the ITP method on the average performance by any significant margin. Hence, opting for the ITP method instead of binary searching comes at (almost) no cost even if the distribution is misspecified.

It is worth pointing out that our findings bear close resemblance with those of (Oliveira & Takahashi, 2020a) for the continuous version of Problem (3.1), i.e. searching the zero of a continuous function. However, despite the resemblance, our findings here are brand new and do not stem from previously known results. In fact, the methods for analysing the discrete searching problem in this chapter are much more closely related to those developed in (Perl & Reingold, 1977) than those developed in the literature of numerical analysis. Perhaps more importantly, we believe that our findings here might be of more significance and repercussion than previous results due to the fundamental role that Problem (3.1) and Algorithm 0 plays in the field of computer science, serving as a basis for a much of algorithmic theory and practice.

Chapter Outline

The following section, entitled *Main Results*, is divided into three parts. The section begins by characterizing necessary and sufficient conditions for Algorithm 0 to be minmax optimal, putting forward results analogous to Theorem 2.1 of (Oliveira & Takahashi, 2020a) which were previously unknown in the discrete case. Then, in Subsection 3.2.1 we describe our main contribution: the ITP method for searching sorted lists with minmax and expected query complexity results in Theorem 9. These results show that under mild conditions the ITP method can attain an expected query complexity of the same order of interpolation search while retaining the minmax optimality of binary search. In Subsection 3.2.2, we calculate lower bounds on the expected query complexity of binary searching under very broad distributional assumptions, and as a consequence we find that our method cannot be outperformed on the average by binary search by more than one or two iterations. Thus, we provide brand new robustness guarantees that cannot be provided by interpolation search. In Section 3.3 we perform extensive experiments on both artificial and real data from which we find that the expected query complexity of the ITP method can be orders of magnitude lower than interpolation search and binary search alike. Finally, in Section 3.4 we summarize and discuss the relevance of our findings and point out applications and future directions of research.

3.2 Main Results

Given a sorted list $\boldsymbol{v}$ and a target value z, at each iteration j of Algorithm 0 define Δ_j as $\Delta_j \equiv b_j - a_j$ and $x_{1/2} \equiv (a_j + b_j)/2$. Then,

Theorem 8. *Algorithm* 0 *requires at most* $N_{1/2}$ *iterations to terminate if and only if at each iteration* j *we have*

$$|\tilde{k}_j - x_{1/2}| \leq 2^{N_{1/2}-j-1} - \tfrac{1}{2}\Delta_j. \tag{3.6}$$

Proof. Given any instance of (3.1), we may calculate the maximum number of iterations $N_{1/2}$ required by any minmax strategy using equation (3.3). After the first iteration one is left with $N_{1/2}-1$ queries and so, from (3.3) we have that $\lceil \log_2(b_1 - a_1) \rceil$ must be at most $N_{1/2} - 1$, thus $b_1 - a_1$ can be at most $2^{N_{1/2}-1}$. Combining this with the fact that $b_1 - a_1$ is less than or equal to $b_0 - \tilde{k}_0$ and $\tilde{k}_0 - a_0$ it follows that as long as $\tilde{k}_0$ is chosen in such a way that both $b_0 - \tilde{k}_0$ and $\tilde{k}_0 - a_0$ are less than or equal to $2^{N_{1/2}-1}$, then, from that step onward, Algorithm 0 can still guarantee termination in $N_{1/2}$ iterations. Requiring that both $b_0 - \tilde{k}_0$ and $\tilde{k}_0 - a_0$ be less than or equal to $2^{N_{1/2}-1}$ is equivalent to enforcing $|\tilde{k}_0 - x_{1/2}| \leq 2^{N_{1/2}-1} - \frac{1}{2}\Delta_0$. This proves Theorem 8 for iteration $j = 0$.

For higher values of j the reasoning is very similar. After steps $0, 1, ..., j-1$, Algorithm 0 is left with $N_{1/2} - j$ iterations. Thus, on step j, as long as $b_{j+1} - a_{j+1}$ is less than or equal to $2^{N_{1/2}-j}$, Algorithm 0 can guarantee termination in at most $N_{1/2} - j$ iterations. Thus, we find similarly that $b_{j+1} - a_{j+1} \leq 2^{N_{1/2}-j}$ is guaranteed when $|\tilde{k}_j - x_{1/2}| \leq 2^{N_{1/2}-j-1} - \frac{1}{2}\Delta_j$, and, this completes the proof. □

Theorem 8 identifies the class of strategies that, similar to binary searching, enjoy minmax optimality. In most situations the set of strategies that satisfy the conditions of Theorem 8 can be quite large. However, when n is equal to 2^T for some $T \in \mathbb{N}$ then we will find that $2^{N_{1/2}-j-1} - \frac{1}{2}\Delta_j$ must be null for $j = 0, 1, 2...$ and thus the class naturally reduces to binary searching. In every other situation $\tilde{k}$ may be chosen by means of interpolation, randomization or any other technique as long as the distance of k_j to $x_{1/2}$ remains within the ranges established by Theorem 8. Figure 3.1 depicts two search trees with $n = 17$ that have depth $N_{1/2} = 5$. Both of these trees have the same minimal depth of binary searching, however they do not subdivide the nodes in half in each query as binary searching would. Given that $n = 17$ is not a power of 2 then several such trees with depth $N_{1/2} = 5$ exist.

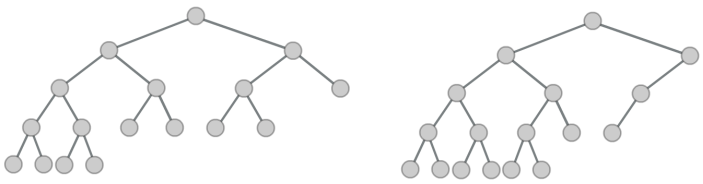


Fig. 3.1: Two searching trees with 17 nodes and minimal depth of 5, none of which correspond to binary searching.

Before we proceed in displaying our main algorithm we point out two variations of Theorem 8 that may be of interest in different circumstances, one less conservative and one more conservative. Both variations are motivated by the fact that minmax

optimality alone does not avoid certain types of inefficiencies. The first of type arises from the fact that (3.6) may, at times, be too restrictive and degenerate to bisection steps too early in a run. This may happen if (3.6) is initiated too small, or, if Algorithm 0 unluckily makes too many "bad guesses" with $\tilde{x}$ producing $\Delta_{j+1} > \Delta_j/2$ for several iterations. To avoid this and produce a variation of (3.6) that is more "forgiving" of bad iterations one may upperbound the maximum number of iterations by $N_{\max}$ instead of $N_{1/2}$ with $N_{\max} \geq N_{1/2} + 1$. This is attained if and only if in each iteration j we have

$$|\tilde{k}_j - x_{1/2}| \leq 2^{N_{\max}-j-1} - \tfrac{1}{2}\Delta_j. \tag{3.7}$$

The second type of inefficiency that may be present is of opposite nature. Minmax optimality may allow for too much freedom. For example, it is possible that Algorithm 0 after a few iterations reduces Δ to a sufficiently small size that it could be tackled with a few binary steps. However, minmax optimality allows Algorithm 0 to waste "spare iterations" produced in the beginning of the run. One way to avoid this is to require that after each iteration, the new subproblem with $b_j - a_j = \Delta_j$ would take no more iterations than binary search would, i.e. that at most $\lceil \log_2 \Delta_j \rceil$ queries would be used from step j onward. This is obtained by enforcing

$$|\tilde{k}_j - x_{1/2}| \leq 2^{\lceil \log_2 \Delta_j \rceil - 1} - \tfrac{1}{2}\Delta_j \tag{3.8}$$

in every iteration instead of equation (3.6).

All three versions of (3.6) may be of interest to software development. If problem (3.1) is generated by a known distribution that allows for the construction of reliable estimators for the location of k_*, as exemplified in the next section, then perhaps the original form (3.6) might be chosen. If (3.6) is too small, then the relaxation in (3.7) might be an appropriate alternative. In fact allowing for as little as one additional iteration with $N_{\max} = N_{1/2} + 1$ we have that equation (3.7) will encompass the entirety of (a, b) in the first iterations. Finally, equation (3.8) might be preferred if the underlying distribution does not allow for the construction of reliable estimators for the location of k_*, or, if the underlying distribution is unknown. In any case, the classes of methods here identified by (3.6) to (3.8) offer a rich collection of alternatives to traditional binary searching that simultaneously retain minmax optimality and allow for enough freedom to incorporate interpolation and/or randomized strategies. In the following subsection we will see that even the "unaltered" minmax optimality condition in (3.6) can allow for an improved average performance under standard uniform distribution hypothesis.

3.2.1 The ITP Method

Let $\kappa_1 \in \mathbb{R}_+$ and $\kappa_2 \in (\frac{1}{2}, 1)$ be two user provided constants[1]. Now define σ and δ as

$$\sigma \equiv \text{sign}(x_{1/2} - x_f) \qquad \text{and} \qquad \delta \equiv \kappa_1 |b - a|^{\kappa_2}, \tag{3.9}$$

[1] Notice that κ_2 is defined here to be between 1/2 and 1, whereas in (Oliveira & Takahashi, 2020a) it is defined to be between 1 and $(1+\sqrt{5})/2$. This difference is key and arises from the fact that in continuous settings one is typically interested in vanishing residuals Δ that are less than or equal to 1, whereas in discrete scenarios Δ is always greater than 1.

where $x_{1/2}$ and x_f are as in (3.2) and (3.4) respectively. Also, define x_t as

$$x_t \equiv x_f + \sigma\delta \tag{3.10}$$

if $\delta \leq |x_{1/2} - x_f|$ and $x_t = x_{1/2}$ otherwise. Now define the minmax radius r_k and interval $\mathcal{I}_k$ as

$$r_j \equiv 2^{N_{1/2}-j-1} - \tfrac{b_k - a_k}{2} \qquad \text{and} \qquad \mathcal{I}_j \equiv \left[x_{1/2} - r_j \,,\, x_{1/2} + r_j\right] \tag{3.11}$$

Now, in each step j define x_{ITP} as the projection of x_t onto $\mathcal{I}_j$, i.e.

$$x_{ITP} \equiv \begin{cases} x_t & \text{if } |x_t - x_{1/2}| \leq r_j; \\ x_{1/2} - \sigma r_j & \text{otherwise.} \end{cases} \tag{3.12}$$

The ITP method then takes $\tilde{k}$ to be equal to k_{ITP} defined as the closest integer to x_{ITP} that lies between x_{ITP} and $x_{1/2}$.

In the following theorem we will assume that $\boldsymbol{v}$ is constructed by sorting n independent samples of a uniformly distributed variable in $[0, 1]$. And, that the minmax interval $\mathcal{I}_0$ around $x_{1/2}$ in the first iteration $j = 0$ is "not too small", i.e. that $2r_0/n$ is not much smaller than one. This avoids the collapsing of $\mathcal{I}_k$ to $x_{1/2}$ (in which case the ITP method behaves identical to binary searching), and also, as shown in the proof of Theorem 9, in combination with the other conditions it guarantees that with high probability a steady state condition with super-linear convergence can be reached within a few iterations.

Theorem 9. *If n is sufficiently large and $\mathcal{I}_0$ is not too small, the number of iterations N for Algorithm 0 to terminate satisfies*

$$N \leq N_{1/2} \qquad \textit{and} \qquad \mathbb{E}(N) \leq \kappa_3 \log_2 \log_2 n \tag{3.13}$$

for some constant $\kappa_3 \in \mathbb{R}$ that depends on κ_1 and κ_2 but not on n.

Proof. The structure of the proof is as follows: We begin by analysing Algorithm 0 for $\tilde{k} = k_t$ where k_t is the closest integer to x_t that lies between x_t and $x_{1/2}$, i.e. without the projection onto $\mathcal{I}_j$. We will see that for sufficiently large n we have that k_t produces an expected query complexity of the order of $\log_2 \log_2 n$. Then, we include the projection step and verify that, if $\mathcal{I}_0$ is not too small, then with high probability the minmax range $\mathcal{I}_j$ will at least double in each iteration and in a few iterations the full interval (a, b) will be encompassed by $\mathcal{I}_j$. After that point, Algorithm 0 will behave as if there were no projection step, and thus, the same expected query complexity of k_t is attained.

Before we proceed with the proof, we point out that no attempt is made here to obtain the tightest bounds nor to optimize our choice of κ_1, κ_2 or any other meta-parameter. Instead, whenever possible we opted for the simplest and shortest path to obtain our results, and, overall aim for a proof that is accessible to a university level advanced algorithms course.

In order to calculate the expected query complexity of Algorithm 0 implemented with $\tilde{k} = k_t$ we first calculate the expected number of iterations $N_\delta \in \mathbb{R}$ that Algorithm 0 requires to reduce an interval of length Δ to a new interval with length less than or equal to $8\delta = 8\kappa_1 \Delta^{\kappa_2}$. For this we will use a few facts. First notice that the distance between k_t and k_* can be upper-bounded by:

$$|k_t - k_*| \leq |x_f - k_*| + \kappa_1 \Delta^{\kappa_2} + 1. \tag{3.14}$$

We refer to the first term $|x_f - k_*|$ as the estimation error, the second term $\kappa_1 \Delta^{\kappa_2}$ as the truncation error, and the third term "+1" is the round-off error.

We say that an iteration j is *successful* when $\Delta_{j+1} \leq \Delta_j/2$, and, it is *unsuccessful* when $\Delta_{j+1} > \Delta_j/2$. Notice that if $\boldsymbol{v}$ is built by sorting n independent samples from a uniform distribution over $[0, 1]$, then the probability of an iteration j with $\tilde{k} = k_t$ being successful is equal to the probability that k_t is between k_* and $x_{1/2}$. Without loss of generality we may assume that $k_t \leq x_{1/2}$, and thus the probability of a successful iteration is equal to the probability that $k_* \leq k_t$. Now the index k_* is equal to the number of entries v_j of $\boldsymbol{v}$ that satisfy $v_j \leq z$, and since each entry is sampled from a uniform distribution over $[0, 1]$, then, in problem (3.1) the variable k_* follows a binomial distribution with expected value of $\mu = n \cdot z$ and with variance $\varsigma^2 = nz(1 - z) \leq n/4$. During a run of Algorithm 0, given all the data collected up to iteration j, by using scaling arguments we find that the conditional distribution of k_* will also follow a translated binomial between a_j and b_j, with mean $\mu = x_f$ and with variance $\varsigma^2 \leq \Delta/4$. Thus, from (3.14) and Chebyshev's inequality $\mathbb{P}(|y - \mu| \geq t) \leq \varsigma^2/t^2$ we find that:

$$\mathbb{P}(\text{unsuccessful iteration}) \quad \leq \quad \mathbb{P}(|k_* - x_f| \geq \delta) \quad \leq \quad \frac{1}{4\kappa_1^2 \Delta^{2\kappa_2 - 1}}. \tag{3.15}$$

And thus, since $\kappa_2 > 1/2$, the probability of an unsuccessful iteration vanishes with larger values of Δ. We will denote by P_s and P_u the probabilities of successful and unsuccessful iterations respectively.

Now, from (3.15) we have that for large values of Δ the estimation error is smaller than the truncation error with high probability. The same is true of the round-off error. Thus we deduce that

$$|k_t - k_*| \leq 3\kappa_1 \Delta^{\kappa_2} \tag{3.16}$$

with high probability for large values of Δ.

Now let us analyse two different scenarios: (i) when k_* is near extremity a or b; and (ii) when it is somewhere in the middle. Or, formally: (i) when $k_* - a \leq \kappa_1 \Delta^{\kappa_2}$ or $b - k_* \leq \kappa_1 \Delta^{\kappa_2}$; (ii) every other case. It is easy to see that in case (i), with high probability, one successful iteration will suffice to reduce Δ to less than or equal to $4\kappa_1 \Delta^{\kappa_2}$. This is a direct consequence of equation (3.16) and (3.15). Similarly, notice that case (ii) after one iteration will produce $k_* - a \leq 4\kappa_1 \Delta^{\kappa_2}$ or $b - k_* \leq 4\kappa_1 \Delta^{\kappa_2}$ with high probability. Thus, after two successful iterations case (ii) will reduce Δ to less than or equal to $7\kappa_1 \Delta^{\kappa_2}$. Hence, with high probability, it suffices to obtain two successful iterations in order to reduce Δ to less than or equal to $8\kappa_1 \Delta^{\kappa_2}$, and, the expected number of iterations required to obtain two successes is given by

$$\mathbb{E}(\#\text{ iterations to obtain two successes}) = (2 \cdot 1)P_s^2 + (3 \cdot 2)P_u P_s^2 + (4 \cdot 3)P_u^2 P_s^2 ...$$

which, by using the relation $P_s + P_u = 1$ simplifies to

$$\mathbb{E}(\#\text{ iterations to obtain two successes}) = 2/P_s \xrightarrow{\Delta \to \infty} 2.$$

Thus we find that N_δ approaches 2 as Δ goes to infinity, and hence for Δ greater than or equal to some constant κ_4 (that depends on κ_1 and κ_2 alone) we have N_δ less than or equal to 3. This implies that for large Δ we have:

$$\mathbb{E}(N|\Delta) \leq 3 + \mathbb{E}(N|8\kappa_1\Delta^{\kappa_2}), \tag{3.17}$$

where $\mathbb{E}(N|Z)$ is the expected number of iterations given that there are Z elements in $[a, b]$. Thus applying (3.17) recursively we find that

$$\mathbb{E}(N|\Delta) \leq 3 + 3 + \mathbb{E}(N|8\kappa_1(8\kappa_1\Delta^{\kappa_2})^{\kappa_2})$$

and repeating this process m times we find

$$\mathbb{E}(N|\Delta) \leq m \times 3 + \mathbb{E}(N|(8\kappa_1)^{\frac{\kappa_2^m - 1}{\kappa_2 - 1}}\Delta^{\kappa_2^m}).$$

Thus, the value of m such that $(8\kappa_1)^{\frac{\kappa_2^m - 1}{\kappa_2 - 1}}\Delta^{\kappa_2^m}$ is less than a κ_4 will give us the expected query complexity of Algorithm 0 implemented with $\tilde{k} = k_t$. This, of course, reduces to

$$m \leq C_1 + C_2 \log_2 \log_2 \Delta$$

for some C_1 and C_2 that depend on κ_1 and on κ_2 but not on Δ. This completes the deduction of the expected query complexity of Algorithm 0 implemented with k_t.

What is left now is to verify the effect of the projection step on the behaviour of Algorithm 0. We start by pointing out that for high values of Δ, due to (3.15), Algorithm 0 implemented with k_t generates successful iterations with high probability. The same is true for the projection of k_t onto $\mathcal{I}_j$, since, if k_t lies between k_* and $x_{1/2}$ then so will the projection of k_t onto $\mathcal{I}_j$. Thus, with high probability we are left with the smallest amongst the intervals $(a, \tilde{k})$ and $(\tilde{k}, b)$ after each iteration. This implies that $\Delta_{j+1} \leq \Delta_j/2$ and, with a little algebra, we can show that the fraction of the interval (a_j, b_j) covered by $\mathcal{I}_j$, which we will denote by $F(j)$, increases from iteration j to iteration $j + 1$, i.e. that

$$F(j+1) \equiv \frac{2(2^{N_{1/2}-(j+1)-1} - \Delta_{j+1}/2)}{\Delta_{j+1}} \geq \frac{2(2^{N_{1/2}-j-1} - \Delta_j/2)}{\Delta_j} \equiv F(j).$$

Furthermore, if $\tilde{k}$ is not equal to k_t, then it is the projection onto the minmax disk. Thus, (ignoring rounding effects) we find that $\tilde{k}_j = x_{1/2} \pm (2^{N_{1/2}-j-1} - \Delta_j/2)$. In which case since $\Delta_{j+1} \leq \Delta_j/2$ we must have that:

$$\Delta_{j+1} = \Delta_j - 2^{N_{1/2}-j-1};$$

from which we derive that the fraction of the interval (a_{j+1}, b_{j+1}) covered by $\mathcal{I}_j$ is given by

$$F(j+1) \equiv$$
$$\frac{2(2^{N_{1/2}-(j+1)-1} - \Delta_{j+1}/2)}{\Delta_{j+1}} = \frac{2(2^{N_{1/2}-j-1} - \Delta_j/2)}{\Delta_j} \times \frac{1}{1 - 2^{N_{1/2}-j-1}/\Delta_j},$$

and thus $F(j+1) > 2F(j)$ since $2^{N_{1/2}-j-1}/\Delta_j$ is greater than $1/2$ given that Δ_j is not a power of two. Thus, with high probability the fraction of the interval (a_{j+1}, b_{j+1}) covered by $\mathcal{I}_j$ must at least double in each iteration if $\tilde{k} \neq k_t$. Hence, if the fraction of the interval (a_0, b_0) covered by $\mathcal{I}_0$ is not too small, it will take only a few iterations until $\tilde{k}$ can assume any value within (a, b), and will thus coincide with k_t from that iteration onward. □

Theorem 9 shows that the ITP method is both minmax optimal and can attain as low as $\log_2 \log_2 n$ expected query complexity given that $\mathcal{I}_0$ is not too small. This last condition, as mentioned earlier, can be dropped if minmax optimality is relaxed. In fact it suffices to allow for just one iteration more than $N_{1/2}$ and the "not too small" condition is satisfied. Also, it is worth mentioning that one may calculate the expected number of "gained" iterations per query and find that it is greater than or equal to one for sufficiently large n. Thus, even though $\mathcal{I}_j$ can collapse into binary searching after a few rounds of unsuccessful iterations, this will only happen with low probability since in an average run, the ITP method it will typically accumulate "spare iterations" and can afford a few misses quite early in the run. Also, from (3.15) we may deduce that the first iterations have the highest probability of being successful since these have the biggest intervals Δ, and hence, the iterations in which Algorithm 0 has less "spare iterations" are the ones less likely to blunder and produce unsuccessfull/wastefull iterations. By the time it has narrowed down the search to smaller intervals, several "spare" iterations will be available, and thus it will take many more unsuccessfully iterations for $\mathcal{I}_j$ to degenerate to binary searching.

3.2.2 Robustness and Limits

It is well known that the expected query complexity of interpolation search is of the order of $\log_2 \log_2 n$ and binary search is of the order of $\log_2 n$ under the uniform distribution assumption, i.e. interpolation search considerably outperforms binary search under the standard hypothesis. However, it is also well known that if the distribution is misspecified, then, the expected query complexity of interpolation search can reach up to n queries while binary search remains upper bounded by $\lceil \log_2 n \rceil$, i.e. interpolation search is considerably outperformed by binary search under misspecified conditions. In this section we will verify whether the ITP method suffers from the same drawback or whether it is robust to such changes, i.e. can the ITP method be outperformed by binary searching with respect to average performance[2]?

We will answer this question by analysing two large classes of distributions over instances of (3.1). The first class , which we will denote by $\mathcal{C}_1$, encompasses all distributions over instances of (3.1) that produce $z \neq v_{k_*}$ with no restriction on how z and $\boldsymbol{v}$ are generated. The second class, denoted $\mathcal{C}_2$, encompasses a large collection of distributions over instances of (3.1) that do produce $z = v_{k_*}$. In particular, the second class includes any distribution that does not limit k_* nor favours any particular j between 1 and n, i.e. it assumes only that k_* can assume any value from 1 to n with a uniform probability.

If Problem (3.1) is generated by a distribution from class $\mathcal{C}_1$, then since the distribution does not produce $z = v_{k_*}$, binary search must require at least $N_{1/2} - 1$ iterations to terminate. To see this, first notice that $\Delta_j \geq \frac{1}{2}(\Delta_{j-1} - 1)$, for any value of $\Delta_j = b_j - a_j$ (whether odd or even). By recursion, we find that $\Delta_j \geq$

[2] In the continuous setting this question was answered in Corollary 2.2 of (Oliveira & Takahashi, 2020a). There, since the bisection method has a fixed expected query complexity of $N_{1/2}$ for any continuous distribution, the worst-case guarantees of the ITP method already ensure that the expected query complexity of the ITP method cannot be outperformed by the bisection method. However, unlike the continuous setting, the expected query complexity of binary searching over lists does depend on the underlying distribution.

$\frac{1}{2^j}\Delta_0 - \frac{1}{2} - \frac{1}{4} - ... - \frac{1}{2^j}$ which in turn is greater than $\frac{1}{2^j}\Delta_0 - 1$. Hence, in order for Δ to be less than or equal to 1 the number of iterations N must satisfy: $N \geq \lceil \log_2 \Delta_0 \rceil - 1$. Thus

Corollary 2. *If the distribution over instances of (3.1) is such that $z \neq v_{k_*}$ then the expected query complexity of binary searching satisfies:*

$$\mathbb{E}(N) \geq N_{1/2} - 1. \tag{3.18}$$

The second class of distributions does allow for Problem (3.1) to admit a solution with $z = v_{k_*}$. The class $\mathcal{C}_2$ assumes nothing on how $\boldsymbol{v}$ or z is constructed other than the fact that the solution k_* can assume any value within the range from 1 to n with a uniform probability[3]. In this second case it is useful to consider the graph constructed by placing the first index visited on the root, and, successively branching left with the indices probed in case of $z < v_k$ and branching right when $v_k < z$. Figure 3.2 illustrates one such construction. The depth of the resulting tree

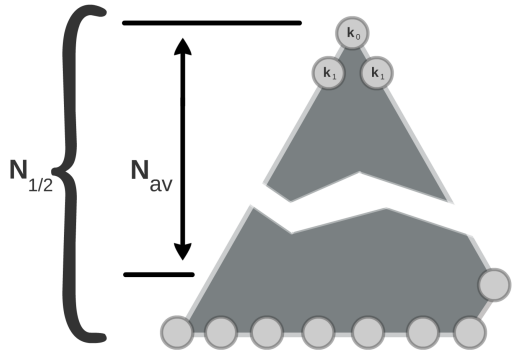


Fig. 3.2: The binary search tree associated with Algorithm 0. Each node of the tree represents an index k of vector $\boldsymbol{v}$ visited by Algorithm 0, the height of the tree represents the worst-case complexity of the searching strategy and the average depth of the tree N_{av} is represents the expected query complexity of the searching strategy.

measures the maximum number of iterations required for Algorithm 0 to terminate, and, the average depth of the graph measures the average number of iterations. We will denote the average depth by N_{av} and we decompose n into two factors as $n = 2^{N_{1/2}-1} + q$ for some $q \in \{1, .., 2^{N_{1/2}-1}\}$. This way we find that

[3] This second constraint is added since otherwise it is easy to construct distributions that can arbitrarily lower the expected query complexity of virtually any method. Taking binary search as an example, if the distribution trivially produces $v_{k_{1/2}} = z$ then the expected query complexity can be as low as one iteration. Thus to exclude trivial cases and arbitrary biases we assume that k_* is equally likely to assume any value between 1 to n.

Corollary 3. *If the distribution over instances of (3.1) is such that $z = v_{k_*}$ and k_* is equally likely to assume any value between 1 to n then the expected query complexity of binary searching is equal to $N_{av} = N_{1/2} - 1 - \delta$ where $\delta = \frac{n - N_{1/2} - 2q}{n-1}$, and satisfies*

$$\mathbb{E}(N) \geq N_{1/2} - 2. \tag{3.19}$$

Corollary 3 is well known and it's proof is thus omitted for simplicity [4]. Now combining the above corollaries with the fact that the ITP method requires no more than $N_{1/2}$ iterations to terminate we find that for the classes of distributions in $\mathcal{C}_1$ and $\mathcal{C}_2$ described above:

Theorem 10. *The expected query complexity of binary searching can outperform that of the ITP method by at most two iterations.*

Hence, unlike interpolation search, even under very broad misspecified conditions the ITP method cannot be outperformed by binary searching by any significant margin. Thus, Theorems 9 and 10 combined show that by choosing the ITP method over binary searching, not only will Algorithm 0 enjoy the benefits associated with interpolation search (the $\log_2 \log_2 n$ complexity over the uniform distribution assumption), but it will also not suffer the drawbacks associated with interpolation search (being outperformed by binary searching under misspecified conditions).

3.3 Experimental Results

In this section we empirically test the ITP strategy on three experiments and compare it with traditional binary searching and interpolation search. In the first experiment we test the minmax ITP method with varying values of κ_1 and κ_2 in order to find the values of κ_1 and κ_2 that minimize the expected number of queries under a uniform distribution assumption. The second and third experiments use the values of κ_1 and κ_2 found on the first experiment and compare the minmax ITP method with the relaxed version where $N_{\max} = N_{1/2} + 1$ and interpolation search over both artificial and real data.

Artificial Data 1

In our first experiment, we search for the values of κ_1 and κ_2 that minimize the expected number of iterations required by the minmax ITP method over lists of size $n = 2 \times 10^5$. As seen in the proof of Theorem 9, the behaviour of the ITP method quickly mimics the behaviour of k_t which depends solely on the values of κ_1 and κ_2. We performed 10^4 Monte Carlo simulations by generating the list $\boldsymbol{v}$ by sorting n independent samples from a uniform distribution over $[0, 1]$. The target value z was also sampled from a uniform distribution over $[0, 1]$. Table 3.1 shows the empirical average obtained by varying κ_1 between 0.01 and 0.78, and, varying κ_2 between 0.51 and 0.99.

[4] For completeness sake we point to Prof. PhD Steven Pigeon's proof an analysis of Corollary 3 in *Average node depth in a Full Tree* that can be found in `https://hbfs.wordpress.com/2013/05/14/average-node-depth-in-a-full-tree/`, published in 2013.

Table 3.1: Average number of iterations of 10^4 Monte Carlo simulations of searches in lists of size $n = 2 \times 10^5$ and z sampled from a uniform distribution between 0 and 1. Each column shows the performance of the ITP method with a given value of κ_1 and each line a fixed value of κ_2.

	κ_1 :							
	0.01	0.12	0.23	0.34	0.45	0.56	0.67	0.78
0.51	7.44	7.35	7.68	8.02	8.40	8.76	9.09	9.38
0.56	7.39	7.37	7.79	8.30	8.85	9.35	9.78	10.16
0.62	7.31	7.43	8.08	8.87	9.57	10.17	10.66	11.07
0.67	7.20	7.63	8.66	9.63	10.46	11.13	11.68	12.13
κ_2 : 0.72	7.08	8.05	9.38	10.54	11.51	12.25	12.81	13.32
0.78	6.95	8.64	10.26	11.66	12.73	13.52	14.09	14.51
0.83	6.87	9.35	11.40	13.00	14.09	14.73	15.19	15.57
0.88	6.93	10.27	12.78	14.42	15.23	15.60	15.93	16.30
0.94	7.21	11.45	14.39	14.95	15.94	16.77	17.24	17.51
0.99	7.51	13.03	14.55	16.53	17.49	17.69	17.69	17.69

As can be seen in Table 3.1, the empirical average was minimized at $\kappa_1 = 0.01$ and $\kappa_2 = 0.83$. We highlighted the cell located on the first column and on the seventh row to show the number of iterations attained with these values of κ_1 and κ_2 which are significantly lower than $N_{1/2} = \lceil \log_2 2 \times 10^5 \rceil = 18$. It should also be noted that the average number of iterations remains below $N_{1/2}$ for any value of κ_1 and κ_2 as predicted by Theorem 8.

Artificial Data 2

In our second experiment we compare the empirical performance of two versions of the ITP method against interpolation search on lists of various sizes. The first version of the ITP method used is the minmax version analysed in Theorem 9 and the second version is the one that makes use of the relaxation $N_{\max} = N_{1/2} + 1$. The average number of iterations required by each method was calculated by averaging the results of 500 Monte Carlo simulations on lists of sizes ranging from $n = 1$ to $n = 2^{18}$ generated by sorting n independent samples from predetermined distributions. The maximum number of iterations required by interpolation search is also reported for comparison with the worst case performance of the ITP method. In Figure 3.3 we plot the number of iterations as a function of the size of the list for lists generated from the uniform distribution and Figure 3.4 show the results under different distributions, namely: when the elements of $\boldsymbol{v}$ are samples of (i) a Gaussian distribution, (ii) an exponential distribution, (iii) a triangular distribution and (iv) a step function distribution (two overlapped uniform distributions over different intervals). The Gaussian in (i) was generated in each run with a random mean μ sampled from a uniform distribution over $[0, 1]$ and a fixed variance with $\sigma = 0.01$; the exponential in (ii) was constructed with a parameter of $\lambda = 1$; the triangular distribution was obtained by taking the square root of a uniformly distributed variable; and the step function distribution in (iv) was obtained by sampling from a distribution that is uniform over the intervals $A = [0, 0.75)$ and $B = [0.75, 1)$ with interval A concentrating half of the cumulative probability and B the other half.

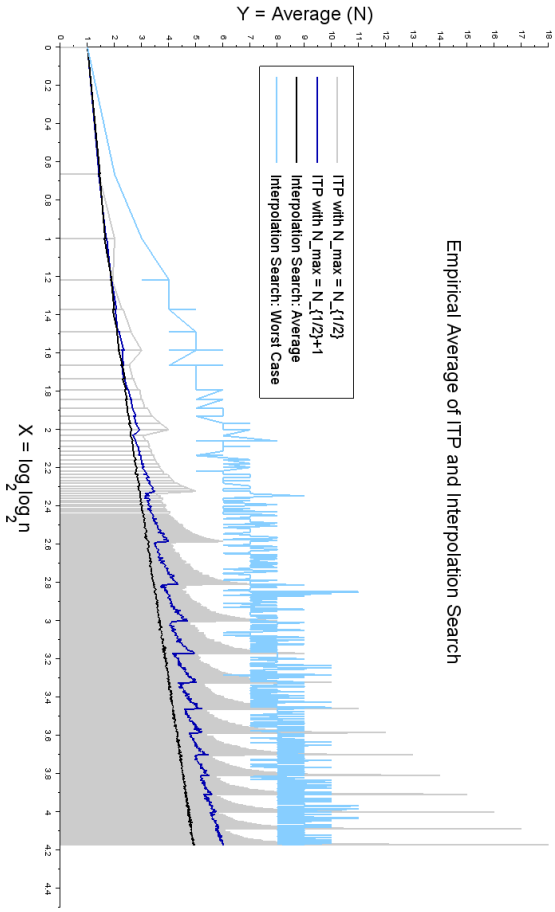


Fig. 3.3: The average of 500 Monte Carlo simulations comparing two versions of the ITP method and interpolation search for increasing values of n on data with uniform distribution. In the background, the bar plot in gray displays the average number of iterations required by the minmax version of the ITP method. The lower curve in black shows the average number of iterations required by interpolation search; the dark blue curve above it the average number of iterations required by the ITP method with $N_{\max} = N_{1/2} + 1$; and, the light blue curve shows the maximum number of iterations used by interpolation search over all 500 runs.

In Figure 3.3, the background bar plot shows the behaviour of the minmax version of the ITP method. As predicted by Theorems 8 and 9, for values of n in which (3.6) is not too small, i.e. most of the range, the growth of $E(N)$ is linear with respect to $\log_2 \log_2 n$ similar to interpolation search. The bar plot shows eighteen peaks which correspond to the values of n that are equal to 2^T for some $T \in \mathbb{N}$; and thus, for those values of n the number of iterations is linear with $\log_2 n$ and not $\log_2 \log_2 n$. The relaxation of the minmax ITP method with $N_{\max} = N_{1/2} + 1$ displayed in dark blue reduces the peaks and obtains a curve that grows linearly with $\log_2 \log_2 n$ in it's entirety just as interpolation search displayed right below it in black. The average performance of the ITP method with $N_{\max} = N_{1/2} + 1$ when compared with interpolation search, attains an almost identical linear growth with respect to $\log_2 \log_2 n$ that exceeds the number of iterations required by interpolation search by approximately one iteration throughout the range investigated, i.e. the ITP method with $N_{\max} = N_{1/2} + 1$ has a nearly identical expected query complexity as interpolation search under the uniform distribution hypothesis. However, the worst case behaviour of interpolation search is upper-bounded by n, i.e. both versions of the ITP method depicted have much better worst case guarantees than interpolation search. The light blue curve overarching the graph depicts the maximum number of iterations required by interpolation search in the 500 runs; and, as can be noticed it exceeded $\log_2 n$ for values of n less than or equal to $2^{2^{3.3}}$ which is approximately 10^3. Of course, with more runs, interpolation search will demand much more iterations in the worst case.

When different distributions are considered then the robustness of the ITP method becomes an interesting feature. As can be seen in Figure 3.4, the average number of iterations of the ITP method with $N_{\max} = N_{1/2} + 1$ remained below $\log_2 n$ under all distributions considered. Interpolation search performed much worse than $\log_2 n$ for both the Gaussian distribution and the exponential distribution, and displayed an average performance that seems to be close to $\log_2 n$ under the step function and the triangular distribution considered. The worst case behaviour of interpolation search showed to be much worse than $\log_2 n$ under the four distinct distributions. As depicted in Figure 3.4, under these distributions and others still, interpolation search may have both an average and a worst case performance that require much more iterations than the ITP method by several orders of magnitude. Thus, these experiments show that the ITP method seems to be a much better alternative than both binary searching and interpolation searching when both worst case and average performances are taken into account.

Real Data

In our third experiment we collect a wide variety of real data from publicly available lists of varying sizes and different origins which are specified in the appendix section. To name a few, we have included a list of full names of all public employees of the Brazilian government, a dataset of genome sizes of fungal species, atomic weights, zip codes and others. For each list we calculate the empirical average of the number of iterations required by both the ITP method with $N_{\max} = N_{1/2} + 1$ and interpolation search. In each run we sample z between v_0 and v_n with a uniform probability and perform the search with both methods. Four of the twelve lists considered were composed of names rather then numbers, specifically the NASDAQ Acronyms, the English Dictionary, the Family Names and Full Names. These were converted into

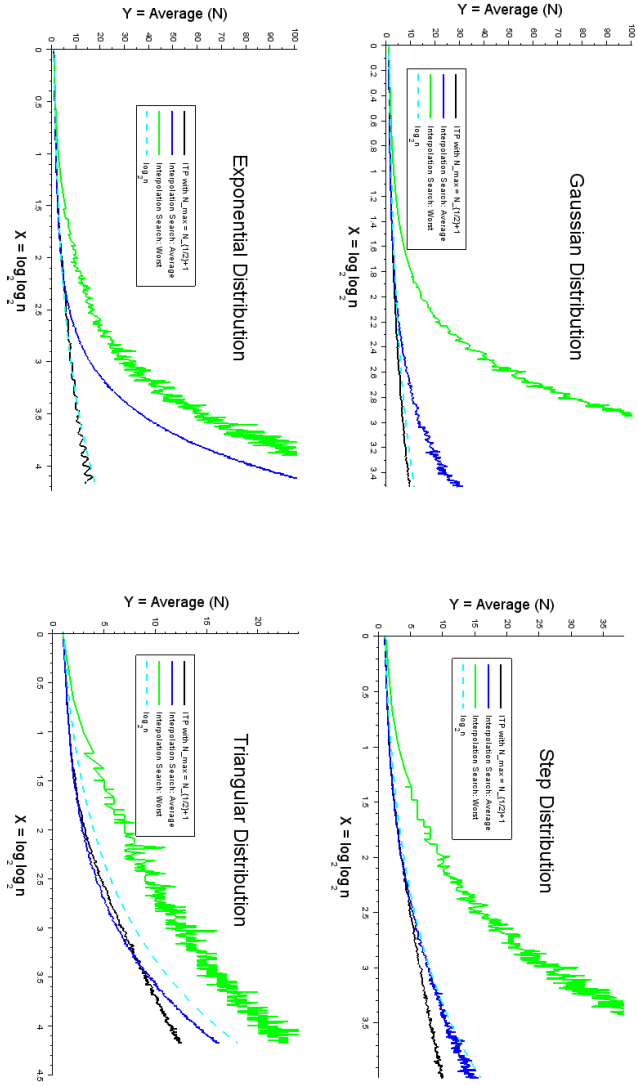


Fig. 3.4: The average of 500 Monte Carlo simulations comparing the ITP method and interpolation search for increasing values of n. The light blue dashed line provides for reference the value of $\log_2 n$. The lower curve in black shows the average number of iterations required by the ITP method with $N_{\max} = N_{1/2} + 1$; the dark blue curve the average number of iterations required by interpolation search; and, the green curve shows the maximum number of iterations used by interpolation search over all 500 runs.

numerical lists by taking a base-27 read of each digit and sorting them accordingly. Other natural approaches that could be used are the ASCII standard conversion or even a Morse code mapping onto binary numbers. Clearly, the average performance of the ITP method is sensitive to this mapping and hence there is space for improvement. However, this goes beyond the scope of this chapter and thus we opted to display only the results for the first approach considered, i.e. the base 27 conversion. Table 3.2 reports the empirical average of 10^3 runs of the described procedures.

Table 3.2: Average number of iterations required by the ITP method and Interpolation Search. The averages are taken over 10^3 searches for a target z sampled from a uniform distribution between v_0 and v_n. We also report the empirical maximum number of iterations required by Interpolation Search over this sample. The simulation capped out the count when more than 1000 iterations were required, and thus when 1000 iterations are reached we indicate with a sub-index the number of runs where this cap occurred. Also, since the ITP method was implemented with $N_{\max} = N_{1/2} + 1$, in the second column under the title "ITP Search" we provide the value of $N_{1/2}+1$. On the bottom lines, the estimates of the mean and the median are displayed in units of $N_{1/2}$.

	ITP Search		Interp Search			
	mean	max	mean	max	$N_{1/2}$	n
Thermodynamics Table	3.6	7	3.1	5	6	49
Atomic Weights	3.3	7	2.8	7	6	54
Fluid Dynamics Chart	5.8	11	5.8	11	10	600
Fibonacci Sequence	8.2	11	19.8	553	10	700
Genome Sizes	9.6	13	8.9	27	12	2352
NASDAQ Acronyms	10.6	15	28.8	1000_{18}	14	8203
Zip Codes	10.5	18	9.8	69	17	81831
Family Names	16.5	18	90.0	1000_{14}	17	88799
English Dictionary	19.0	20	247.4	1000_{103}	19	370103
Full Names	20.6	21	751.3	1000_{667}	20	660276
Prime Numbers	7.2	21	6.0	10	20	664579
Harmonic Series	22.3	25	79.7	189	24	10^7
Central Tendency Metrics:						
Mean	$0.75N_{1/2}$		$5.53N_{1/2}$			
Median	$0.78N_{1/2}$		$1.36N_{1/2}$			

Table 3.2 displays the average number of iterations required by the ITP method side by side with the number of iterations required by interpolation search. The ITP method seems to have a better performance when compared to interpolation search under both the average query complexity criteria and the worst case query complexity criteria. In all instances where interpolation search outperformed the ITP method on the average, it did so by less than 1.21 iterations, and when the ITP method outperformed interpolation search it did so by up to 730.75 iterations which

is more than 36 times the number of of iterations required by the ITP method. On the average the ITP method required 25% less iterations when compared to binary searching whereas interpolation search required on average more than five times the number of iterations as binary searching across all twelve lists. We point out that even if outliers were excluded from the list (the two most difficult cases for interpolation search for example) interpolation search still attains an empirical average worse than binary search, i.e. interpolation search does not seem to perform well in real data. One possible explanation for this might be the fact that real world data is not generated from uniformly distributed variables, and hence, the robustness guarantees provided by the ITP method seem to be vital for outperforming binary search in real world applications. By analysing the median metric a similar conclusion is reached, i.e. interpolation search performs poorly and the ITP method outperforms binary search.

Remark 7. The experiment reported in Table 3.2 by construction produces many unsuccessful searches. This in turn produces slightly longer iteration counts on all ITP, interpolation search and binary searching strategies because successful searches terminate before the exhaustive bracketing of the indices is complete. An analogous experiment where only successful searches are performed was also carried out in order to measure the comparative performance of the three methods considered under more favourable conditions. This was done by sampling the elements from the list with a uniform probability and performing a search for the sampled element. We opted however to omit this experiment due to the very similar findings obtained in the unsuccessful case. Furthermore, this similar behaviour is already somewhat to be expected due to the result in Corollary 3 which shows that binary searching does not (significantly) benefit from successful searches on the average; and, similar arguments can be made to show that both the ITP method and interpolation search should also not be expected significantly reduce iteration count.

When considering the worst case performances, since the ITP method in display made use of the relaxation $N_{\max} = N_{1/2} + 1$, then the ITP method never required more than one iteration above $N_{1/2}$, but due to the $O(\log_2 \log_2 n)$ expected query complexity, under favorable conditions it performed less than half the number of iterations of binary searching. On the other hand, interpolation search not only averaged higher iteration counts but it also maxed out the number of iterations with several unsuccessful searches, and hence, it seems to be the least interesting alternative amongst the three when both metrics are taken into consideration.

General Recommendations

Throughout our experiments (including an extensive number of experiments not reported here) the performance of the ITP method with the relaxation $N_{\max} > N_{1/2}$ seems to give the best results overall. With the relaxation, the ITP method is less sensitive to the value of n but also less sensitive to the choice of κ_1 and κ_2. As a rule of thumb we recommend the ITP method with $\kappa_1 = 0.01$ and $\kappa_2 = 0.83$ and with the relaxation[5] of $N_{\max} = N_{1/2} + 0.99$, however, if prior knowledge on the distribution

[5] By adopting a non integer value for $N_{\max}$, the maximum number of iterations of Algorithm 0 is of $\lceil N_{\max} \rceil$. Furthermore, the projection step of the ITP method

over instances of (3.1) is available, or, if there is availability of a training set, then the values of κ_1, κ_2 and $N_{\max}$ can be tested and chosen accordingly.

In Experiment 2, both interpolation search and the ITP method were implemented under four misspecified conditions when the non-uniform distributions were used to generate $\boldsymbol{v}$. If prior knowledge of the underlying distribution is available, then the behaviour of Algorithm 0 depicted in Figure 3.3 can be obtained for different distributions by implementing Algorithm 0 on the transformation of v_j by the cumulative distribution.

Remark 8. It is a difficult task to select a representative collection of instances that can cover the range of applications in which (3.1) emerges. Several lists tend to be labeled by (nearly) equally spaced points; specifically, engineering applications, physics and chemistry charts tend to fall into this category. We opted to hold out these examples since in these cases k_f is almost always "on-the-spot". In doing so we have biased our sample in a conservative way and so we expect that time savings should be closer to the upper quantile (about 30%) instead the reported 13.7% and 15.5%.

Remark 9. Also, it is worth noticing that even before searching is performed on a sorted list, other data manipulation operations can enjoy of immediate time savings when an improved searching strategy is employed. For example, in order to produce the sorted list, sorting algorithms are often used in order to organize data in ascending order; see chapters "Internal Sorting" and "Optimum Sorting" in (Knuth, 1998). Classical sorting strategies are known to require at most $n \log n$ comparisons in order to sort n elements; and, this is attained by *binary insertion sort*, *merge sort* amongst others. Notice that by substituting binary searching by the ITP method we may trivially construct an *ITP insertion sort* strategy improving on *binary insertion sort* thus obtaining an expected $n \log \log n$ comparisons while requiring no more than $n \log n$ in the worst case. Hence, even though we report the above time savings on the search operation, in practice much more time can be saved since the list being searched could, potentially, be produced by means of binary-search dependent sorting strategies.

3.4 Discussion

In this chapter we have identified a novel and yet simple searching method, which we refer to as the ITP method, that attains an expected query complexity of $O(\log_2 \log_2 n)$ iterations and a worst case query complexity of $\lceil \log_2 n \rceil$; i.e. it is optimal with respect to both average and worst case metrics. Furthermore, we also prove robustness guarantees that show that binary search cannot outperform the ITP method by more than a constant factor even if the distributional hypothesis is misspecified. Hence, the ITP method enjoys the benefits of interpolation search (the

---

projects to the interior of $\mathcal{I}_j$ instead of the border, avoiding numerical errors associated to edge cases. Our experiments were performed with $N_{\max} = N_{1/2} + 1$, however for practitioners we recommend a non integer $N_{\max}$ such as $N_{\max} = N_{1/2} + 0.99$ instead.

improved expected query complexity of $\log_2 \log_2 n$) without the drawbacks associated with it (a lower than binary search expected query complexity when distribution is misspecified). We perform extensive testing on artificial and real data and we find that the ITP method can considerably outperform both the classical binary search method as well as interpolation search. We reach time-savings that range roughly from 25% to 75%, depending on the experiment, when compared to binary searching; an overall much better performance than interpolation search when compared across experiments.

Binary searching is a fundamental tool in the field of computer science and has continually been the choice for applications, specifically due to its minmax optimality. Our results show that this preference for binary search, or alternatively for interpolation search, has often been an inefficient one. The improvements highlighted here have both practical and theoretical implications that directly translate to significant time savings, specifically when the cost of a query is much greater than the time to compute the procedure itself. In short, the ITP method is our recommended improvement to the traditional approach. However, the identified minmax class of methods, which is largely unexplored, is potentially a more significant contribution that may lead to further improvements and the identification of even more efficient methods.

Future work

The problem of searching sorted tables and/or other multidimensional variants are natural instances that may benefit if equivalent results as the ones developed here are found. Another relatively unexplored variation studied in (Bentley & Yao, 1976) is searching through infinite lists. Also, assuming multiple instances of (3.1) to be solved sequentially and generated under one common distribution, one may ask how to adapt and improve the solvers in between each resolution to obtain an adaptive/self-improving method. Finally, the cost of one query is typically assumed to be significantly greater than that of the computation of the searching procedure itself; several interesting questions arise when this assumption is modified.

3.A Appendix

3.A.1 Online material

Table 3.3 contains the sources of the twelve lists used in the second experiment. The texts were converted into numerals as explained in the end of Section 3.3 and any additional symbols such as "*.!;" and others were ignored. Several of the files found in the above links contain multiple columns, specifically the fluid dynamics chart, the genome sizes, the atomic weights and the thermodynamics table. When this is the case we selected one column arbitrarily and performed all simulations on the selected column.

Table 3.3: The source of the data collected on the fifth of June of 2019.

NASDAQ Acronyms
`ftp://ftp.nasdaqtrader.com/symboldirectory`
Prime Numbers
(self generated)
Atomic Weights
`https://www.qmul.ac.uk/sbcs/iupac/AtWt/`
Zip Codes
`http://federalgovernmentzipcodes.us`
Fluid Dynamics Chart
`https://engineering.purdue.edu/~wassgren/notes/CompressibleFlowTables.xls`
Genome Sizes
`http://www.zbi.ee/fungal-genomesize/index.php?q`
Fibonacci Sequence
(self generated)
Thermodynamics Table
`https://www.ohio.edu/mechanical/thermo/property_tables/H2O/H2O_TempSat.xls`
English Dictionary
`https://github.com/dwyl/english-words`
Family Names
`https://www.census.gov/topics/population/genealogy/data/2010_surnames.html`
Harmonic Series
(self generated)
Full Names
`http://www.portaltransparencia.gov.br/servidores`

References

Bentley, J. L., & Sedgewick, R. (1997, January). Fast algorithms for sorting and searching strings. *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 360–369.

Bentley, J. L., & Yao, A. C. (1976). An almost optimal algorithm for unbounded searching. *Information Processing Letters*, *5*(3), 82-87. doi: https://doi.org/10.1016/0020-0190(76)90071-5

Cannizzo, F. (2018). Fast and vectorizable alternative to binary search in O(1) applicable to a wide domain of sorted arrays of floating point numbers. *Journal of Parallel and Distributed Computing*, *113*(5), 37. doi: https://doi.org/10.1016/j.jpdc.2017.10.007

Knuth, D. E. (1998). The art of computer programming - sorting and searching. In (2nd ed., Vol. 3, chap. 6.2). Addison-Wesley.

Oliveira, I. F. D., & Takahashi, R. H. C. (2020a). An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software*.

Oliveira, I. F. D., & Takahashi, R. H. C. (2020b). Minmax-optimal list searching with $O(\log_2 \log_2 n)$ average cost. *Under consideration by the*

Journal of Computer and System Sciences, Elsevier. (Pre-print available at `https://arxiv.org/abs/2105.11919`)

Perl, Y., Itai, A., & Avni, H. (1978). Interpolation search—a log log n search. *Communications of the ACM*, *21*, 550-553. doi: https://doi.org/10.1145/359545.359557

Perl, Y., & Reingold, E. M. (1977). Understanding the complexity of interpolation search. *Information Processing Letters*, *6*(6), 219-222. doi: https://doi.org/10.1016/0020-0190(77)90072-2

Schlegel, B., Gemulla, R., & Lehner, W. (2009). k-ary search on modern processors. *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, 52-60.

Yao, A. C., & Yao, F. F. (1976). The complexity of searching an ordered random table. *Proceedings of the Seventeenth Annual Symposium on Foundations of Computer Science*, 173-177. doi: https://doi.org/10.1109/SFCS.1976.32

Armijo's back-tracking problem

Summary. This chapter, contained in pages 77 to 86, is based on the the paper *Efficient inexact line searching* first submitted for consideration by a scientific journal on the 28th of January of 2021; and, as of the day of the submission of this thesis, much work has been added to this paper (after recommendations by the Editor-in-Chief) and a re-submission should be under way within the following months. The latest version of this paper (Oliveira & Takahashi, 2021) can be found in the pre-print repository in the following address: `https://arxiv.org/abs/2110.14072`. In the current form presented below, minor changes are made to make for a more coherent read within the context of this thesis, however, it is kept self contained and an independent read of Chapter 4 is possible without need of reference to external material within the thesis. Any major content foreign to the original paper is highlighted with dark blue text, and, the remainder, where little or no alteration is made, we present the content in plain black text.

In this chapter we formally study the inexact Armijo-type searching problem introduced in Chapter 1 and in particular the ubiquitously employed backtracking procedure used in solving it. Backtracking is an inexact line search procedure that selects the first value in a sequence $x_0, x_0\beta, x_0\beta^2 ...$ that satisfies $g(x) \leq 0$ on $\mathbb{R}_+$ with $g(x) \leq 0$ iff $x \leq x^*$. This procedure is widely used in descent direction optimization algorithms with Armijo-type conditions. It both returns an estimate in $(\beta x^*, x^*]$ and enjoys an upper-bound $\lceil \log_\beta \epsilon/x_0 \rceil$ on the number of function evaluations to terminate, with ϵ a lower bound on x^*. The basic bracketing mechanism employed in several root-searching methods is adapted here for the purpose of performing inexact line searches, leading to a new class of inexact line search procedures. The traditional bisection algorithm for root-searching is transposed into a very simple method that completes the same inexact line search in at most $\lceil \log_2 \log_\beta \epsilon/x_0 \rceil$ function evaluations. A recent bracketing algorithm for root-searching which presents both minmax function evaluation cost (as the bisection algorithm) and superlinear convergence is also transposed, asymptotically requiring $\sim \log\log\log \epsilon/x_0$ function evaluations for sufficiently smooth functions. Other bracketing algorithms for root-searching can be adapted in the same way. Numerical experiments suggest time savings of 50% to 80% in each call to the inexact search procedure.

4.1 Introduction

Backtracking is an inexact line search technique typically used in the context of descent direction algorithms for solving non-linear optimization problems (Gill, Murray, & Wright, 1997; Boyd & Vandenberghe, 2009; Luenberger & Ye, 2018). After a descent direction is computed, a step size must be chosen by solving an inexact line searching problem that can be written as

$$\text{Find } \hat{x} \in \mathbb{R}_+ \text{ such that } g(\hat{x}) \leq 0; \tag{4.1}$$

for some $g : \mathbb{R}_+ \to \mathbb{R}$ such that $g(x) \leq 0$ for all x less than or equal to an unknown turning point $x^* \in \mathbb{R}_+$ and $g(x) > 0$ otherwise. The condition $g(x) \leq 0$ expresses some acceptable criteria for a descent method to attain desired convergence properties, such as the well known Armijo's condition (Armijo, 1966), Wolfe's condition (Wolfe, 1969), amongst others (Burachik, Drummond, Iusem, & Svaiter, 1995; Shi & Shen, 2005; Boukis, Mandic, Constantinides, & Polymenakos, 2010; Calatroni & Chambolle, 2017; Vaswani et al., 2019; Truong & Nguyen, 2021). The backtracking procedure, initiated with some pre-specified values of $x_0 \geq x^*$ and $\beta \in (0, 1)$, sequentially verifies and returns $\hat{x}$ as the first value of the sequence $x_0,\ x_0\beta,\ x_0\beta^2, ...$ that satisfies the inequality in (4.1), i.e. it usually takes no more than three lines (within a larger routine) as described in Algorithm 2.

Algorithm 2: Backtracking

```
1 x̃ ← x_0;
2 while g(x̃) > 0 do
3 |  x̃ ← βx̃;
```

Notwithstanding the relevance of Algorithm 2 as a component of a large variety of nonlinear optimization algorithms, the literature has not focused on its study yet. The working principles of the traditional backtracking algorithm are examined here, and a new class of methods for inexact line search with enhanced performance is proposed.

It is shown that the traditional backtracking delivers a $\lceil \log_\beta \epsilon/x_0 \rceil$ upper-bound on the number of function evaluations to terminate, where ϵ is a lower bound on x^*. The simplest method belonging to the class proposed here, which is based on the traditional bisection algorithm for root-searching, completes the same task with at most $\lceil \log_2 \log_\beta \epsilon/x_0 \rceil$ function evaluations. The same upper bound is provided by another method based on a recent bracketing algorithm for root-searching (Oliveira & Takahashi, 2020), which requires asymptotically only $\sim \log\log\log \epsilon/x_0$ function evaluations in the case of sufficiently smooth functions. Other root-searching bracketing algorithms can be adapted similarly for performing inexact line searches efficiently.

Numerical experiments are provided, suggesting 50% to 80% of function evaluation savings in each call to the inexact search procedure.

4.2 Analysis of Traditional Backtracking

The procedure described in Algorithm 2 enjoys the following guarantees:

Theorem 11. *Assume that $x^* > \epsilon > 0$, $\beta \in (0,1)$, $x_0 > x^*$, and $g(x) > 0$ iff $x > x^*$. Then, the backtracking algorithm finds a solution $\hat{x}$ such that $g(\hat{x}) \leq 0$ in at most $\lceil \log_\beta \epsilon/x_0 \rceil$ iterations, and the solution $\hat{x}$ satisfies $\beta x^* < \hat{x} \leq x^*$.*

Theorem 11 is often an unstated and implicit motivation to employ backtracking, since it both guarantees a finite termination in $\lceil \log_\beta \epsilon/x_0 \rceil$ iterations[1] and gives a guarantee on the location of $\hat{x}$. The more the value of β approximates 1.0 the closer $\hat{x}$ is guaranteed to be to x^*, which is the maximum possible step-size within the guarantees associated with $g(x) \leq 0$. The property that $\beta x^* < \hat{x} \leq x^*$ is often key in ensuring that the parent algorithm "makes the most out of" each descent direction expensively computed throughout its iterations. Of course, arbitrarily fast procedures could easily be devised that find $g(x) \leq 0$ by taking faster converging sequences to 0 if this requirement were to be dropped. Hence implicit to applications that make use of backtracking is the requirement that the solution to problem (4.1) must be "not too far from x^*".

Of a similar nature to the requirement that $\hat{x}$ is "not too far from x^*" is the requirement that x^* is "not too close to zero". Without this, the algorithm could take arbitrarily long to find $\hat{x}$ the closer x^* is to zero. This second requirement is, again, implicit in the formulation of backtracking procedures and, at times, it is even entailed by the construction of the parent algorithm. For example, assume the stopping criteria of the parent algorithm verifies stagnation in the domain of the objective function $f(\cdot)$. Then, by construction, when the parent algorithm finds one instance of (4.1) such that "x^* is too close to zero", it terminates. Thus, with the exception of the very last iteration, every other iteration will satisfy $x^* \geq \epsilon$.

In practice, any backtracking procedure should include a stopping condition that is activated when the iteration count i becomes greater than an allowed maximum i_{max}, in order to guarantee its termination. This is equivalent to the condition $x_0\beta^i < \epsilon$ for $\epsilon = x_0\beta^{i_{max}}$. Hence, the assumption that $x^* \geq \epsilon$ for some pre-specified ϵ seems to be a hypothesis on (4.1) that applications that make use of backtracking must assume, either explicitly or implicitly.

Both of these requirements, extracted from Theorem 11 and found implicitly or explicitly in the literature, are now stated formally for the sake of clarity. We require that:

Condition 1. *For some pre-specified β in $(0,1)$, the solution $\hat{x}$ to problem (4.1) must satisfy $\beta x^* < \hat{x}$.*

Condition 2. *For some pre-specified $\epsilon > 0$, the turning point x^* of problem (4.1) satisfies $\epsilon < x^*$.*

4.3 Bracketing-based inexact line search

The following general algorithm is proposed here:

[1] The exact number of iterations can be more precisely expressed as a function of x^* with the relation $n = \lceil \log_\beta x^*/x_0 \rceil$. The solution-independent bound requires x^* to be bounded away from zero, since otherwise, backtracking may require arbitrarily many iterations the closer x^* is to zero.

Algorithm 3: Fast-tracking

1 $a \leftarrow \epsilon$;
2 $b \leftarrow x_0$;
3 **while** $a \leq \beta b$ **do**
4 | chose $\tilde{x}$ in (a, b) and evaluate $g(\tilde{x})$;
5 | update (a, b) according to (4.2);
6 return $\hat{x} = a$;

The update rule in line 5 is defined by:

$$\begin{cases} a \leftarrow \tilde{x} \text{ if } g(\tilde{x}) < 0 \\ b \leftarrow \tilde{x} \text{ if } g(\tilde{x}) > 0 \\ a \leftarrow \tilde{x} \text{ and } b \leftarrow \tilde{x} \text{ if } g(\tilde{x}) = 0 \end{cases} \tag{4.2}$$

Algorithm 3 defines a class of bracketing-based methods for inexact line search because both the turning point x^* and the final solution $\hat{x}$ are kept inside the interval $[a, b]$ throughout the iterations. Different instances of this algorithm are defined by different choices of $\tilde{x}$ in line 4.

4.3.1 Geometric bisection fast tracking

Consider the instance of Algorithm 3 with the choice of $\tilde{x}$ in line 4 performed according to the choice rule (4.3):

$$\tilde{x} \equiv \sqrt{ab} \tag{4.3}$$

This procedure enjoys the following guarantees:

Theorem 12. *Fast-tracking with $\tilde{x}$ given by (4.3) finds a solution $\hat{x}$ such that $g(\hat{x}) \leq 0$ in at most $\lceil \log_2 \log_\beta \epsilon/x_0 \rceil$ iterations and the solution $\hat{x}$ satisfies $\beta x^* < \hat{x} \leq x^*$.*

Proof. The proof follows from the fact that the inequalities $\beta x^* < \hat{x} \leq x^*$ are equivalent to $\log_2 \beta < \log_2 \hat{x} - \log_2 x^* \leq 0$, which in turn implies that $|\log_2 \beta| > |\log_2 \hat{x} - \log_2 x^*|$. Therefore, to produce an estimate $\hat{x}$ to x^* with *relative* precision of at least β, is equivalent to searching for an estimate $\hat{X} = \log_2 \hat{x}$ of $X^* = \log_2 x^*$ with an *absolute* error of at most $-\log_2 \beta$. Under this logarithmic scale, the bisection method is guaranteed to perform the search task with minmax optimality guarantees. What remains is, quite simply, to translate the bisection method from the logarithmic to the standard scale. This is done as follows: Define $A = \log_2 a$ and $B = \log_2 b$; thus, if the bisection method takes the midpoint $X_{1/2} = (A + B)/2$ in each iteration on the logarithmic scale, then, in the standard scale this translates to $X_{1/2} = (\log_2 a + \log_2 b)/2 = (\log_2 ab)/2 = \log_2 (ab)^{1/2}$. Thus, we have that $\tilde{x}$ must be taken to be equal to $\sqrt{ab}$ in the standard scale.

We now verify that when $B - A \leq -\log_2 \beta$, the lower estimate produced by $A = \log_2 a$ satisfies Condition 1, i.e. that for any value of x^* in (a, b) we must have that $\beta x^* < a$. For this, notice that $B - A \leq -\log_2 \beta \implies \log_2 b/a \leq \log_2 \beta^{-1} \implies b/a \leq \beta^{-1}$ which in turn implies that $\beta b \leq a$. And, since x^* is less than b the inequality in Condition 1 holds. In fact, we express the condition $B - A \leq -\log_2 \beta$ as $a \leq \beta b$ in the standard scale. What is left now is to verify the number of iterations required by the bisection method over the logarithmic scale.

The bisection method requires at most $n_{1/2} \equiv \lceil \log_2(B_0 - A_0)/\delta \rceil$ iterations to reduce the interval (A, B) to one of length $B - A \leq \delta$. Thus, given that $A_0 = \log_2 \epsilon$ and that $B_0 = \log_2 x_0$ and $\delta = -\log_2 \beta$ we find that $n_{1/2}$ is equal to $\lceil (\log_2(B_0 - A_0)/\log_2 \beta) \rceil = \lceil \log_2 ((\log_2 x_0 - \log_2 \epsilon)/\log_2 \beta) \rceil = \lceil \log_2 \log_\beta x_0/\epsilon \rceil$. □

Thus, an immediate consequence of Theorem 12 is that naive backtracking procedures unnecessarily fall short in terms of worst case performance. They require exponentially more iterations on the worst case when compared to simple binary searching applied to the logarithmic scale. Of course, the $\lceil \log_\beta \epsilon/x_0 \rceil$ upper-bound of standard backtracking can, and often is, carefully minimized by choosing x_0 as "near as possible" to x^* by means of interpolation bounds. However, the same procedures that minimize $\lceil \log_\beta \epsilon/x_0 \rceil$ can also be used to minimize the tighter $\lceil \log \log_\beta \epsilon/x_0 \rceil$ upper-bound of *geometric bisection fast-tracking*. Notice that backtracking for an estimate with relative precision β is equivalent to grid searching with a fixed step size on the logarithmic scale: the relative inefficiencies of grid searching when compared to binary searching are well documented in the literature (Press, Teukolsky, Vetterling, & Flannery, 2007). Thus, this improvement is attained with no appeal to additional assumptions on the conditions of Problem (4.1), nor on the use of additional function or derivative evaluations. It is attained solely at the cost of computing the method itself, which for choice rule (4.3) is the additional computation of one square-root per iteration.

The application of the bisection method on the logarithmic scale seems to be an often forgotten technique within the different communities that make use of numerical solvers, and it is certainly under-represented in the literature. We surveyed popular numerical analysis and optimization textbooks, including (Gill et al., 1997; Press et al., 2007; Boyd & Vandenberghe, 2009; Chapra & Canale, 2010; Luenberger & Ye, 2018), and found no reference to this technique, despite the existence of scattered references in computational forums [2] and other isolated references to "geometric bisection" in the context of eigenvalue computation (Ralha, 2012, 2018). In fact, it is easy to find textbook examples that recommend the use of relative error stopping criteria in conjunction with bisection method on a linear scale (see pseudo-code in Figure 5.11 of (Chapra & Canale, 2010) and chapter 9.1 of (Press et al., 2007)). This gives rise to the same inefficiency as the one caused by the use of naive backtracking. Similar remarks can be made concerning the use of golden-section searching / Fibonacci-searching for a minimum using relative error stopping criteria. Of course, the underlying metric behind floating point arithmetic most certainly prioritizes relative over absolute error in numerical representations (Press et al., 2007; Chapra & Canale, 2010), hence it is natural to recommend upper-bounding relative errors and, for the same reasons, the proper logarithmic scaling should be recommended before the use of bisection type methods, specifically when the initial interval (a, b) can span several orders of magnitude.

A noticeable exception to the "inefficiency gap" between the use of arithmetic and geometric averages in the bisection method, is when the search is already ini-

[2] Some early external references to "geometric bisection" can be found in `codeforces.com/blog/entry/49189`, `math.stackexchange.com/questions/3877202/bisection-method-with-geometric-mean` and `github.com/SimpleArt/solver/wiki/Binary-Search`

tiated with a small interval (a, b) with[3] $a, b > 0$ and with $b - a \ll a$, and thus a value of β close to 1. However, standard conditions under which backtracking is used can hardly be expected to satisfy this condition since the further into the run of a descent direction algorithm, the closer x^* is expected to be to zero, and hence, quite the opposite is expected. That is, we find that throughout the run of a standard descent direction algorithm $b - a = x_0 - \epsilon$ tends to be much greater than $a = \epsilon$. Furthermore, the choice of β near one defeats the purpose of employing inexact searching, since it is often intended as a reduction to the computational cost of exact searching. Instead of choosing β near one, in this case one might as well employ exact one dimensional minimization techniques to dictate the step-size.

4.3.2 Fast tracking with multi-logarithmic speed-up

Asymptotic bounds are also improved when the proper scale is adopted. This is shown in the following by making explicit the estimated number of iterations when a hybrid technique for the construction of $\tilde{x}$ is used. The exact construction of $\tilde{x}$ is a straightforward application of the ITP root-searching method, described in (Oliveira & Takahashi, 2020), on the logarithmic scale, and is omitted for brevity.

Corollary 4. *If $\tilde{x}$ in line 4 of Algorithm 3 is taken as the ITP estimate on the logarithmic scale (instead of the bisection method), then, the same guarantees as Theorem 12 hold; and, if furthermore $g(x)$ is C^1 with x^* a simple root, then asymptotically the number of iterations is of the order of $\sim \log\log\log_\beta \epsilon/x_0$.*

Proof. Follows immediately from the properties of the ITP method (Oliveira & Takahashi, 2020). □

Corollary 4 makes use of standard assumptions on the smoothness of g, under which even faster convergence can be guaranteed. The ITP method mentioned therein is an efficient first order root-searching method that in the likes of Ridders' rule, Brent's method or Dekker's method, attains a superlinear order of convergence when employed to solve one dimensional root searching problems. However, unlike the aforementioned methods it is the only one known to retain the minmax optimal performance of the bisection method. The exact inner-workings of the ITP method are beyond the scope of this chapter. A reader more familiar with other hybrid methods (such as Ridders', Brent's or Dekker's method) may substitute the ITP method for the solver of preference, albeit with weaker worst case guarantees. The point being that *multi-logarithmic speed-ups can be attained* with interpolation based strategies while retaining the logarithmic speed-up on the worst case performance.

[3] This way, if the standard bisection method runs till $\beta b \leq a$ for some β near one, it will take a number of iterations $n_{1/2}$ of the order of $\sim \log_2(b_0 - a_0)/[a_0(1-\beta)] = \log_2(b_0 - a_0)/a_0 - \log_2(1-\beta)$, and since $\Delta/x \approx \log_2(x + \Delta) - \log x$, we find that $n_{1/2}$ is of the order of $\approx \log_2(\log_2 b_0 - \log_2 a_0) + \log_2(1-\beta)/\beta = \log_2 \log_2 b_0/a_0 + \log_2 \log_2 \beta$ which simplifies to $\approx \log_2 \log_\beta b_0/a_0$, the complexity of the bisection method applied to the logarithmic scale.

4.4 Experiments

Quick numerical comparisons between standard backtracking and fast-tracking are performed here under the optimization set-up in which inexact searching is typically employed. For this we implement a standard gradient descent algorithm with Armijo's condition, from which the corresponding function $g(x)$ is derived, to minimize ten different loss functions $f : \mathbb{R}^{10} \mapsto \mathbb{R}$ described in Table 4.1. Both methods were initiated at $\boldsymbol{x} = [1, 1, ...1]^T$ with $\beta = 0.8$, $\epsilon = 10^{-10}, x_0 = 1$ and were compared after twenty gradient descent iterations. All functions chosen contain at least one local minimum not too far from the initial guess, and thus both implementations produced approximately the same path, hence ensuring the comparison is made on as-similar-as-possible conditions. We report here the results using a fixed upper-bound step value for x_0 that does not depend on the size of the gradient, i.e. our standard backtracking sequentially searches for the first term in the sequence $\{\boldsymbol{x} + \beta^k \nabla f(\boldsymbol{x})/\|\nabla f(\boldsymbol{x})\| \text{ for } k = 1, 2, ...\}$, and fast-tracking calls an external root-searching solver on the logarithmic scale. We use the ITP method[4], however other non-linear solvers could have been used with slightly weaker guarantees. Under the conditions here considered the simple "geometric average" bisection method would require exactly 7 function evaluations in each iteration if exact arithmetic were used, hence we use this number as a reference point to which standard backtracking and fast-tracking are compared.

Figure 4.1 focuses on the first function considered, and displays the evolution of the number of iterations required by each inexact searching procedure as a function of the gradient-descent iteration. And, as can be seen, fast-tracking tends to reduce the number of iterations the further into the run while backtracking increases the number of iterations the further into the run. This is because interpolation guarantees of the ITP method are improved with the progression of the gradient run (since it is initialized closer to the final solution), while standard backtracking will require more iterations as the ratio of x^*/x_0 is reduced the further into the run. In fact, we observe this pattern of progression of both backtracking and of fast-tracking in most runs.

As can be seen in Table 4.1, fast-tracking vastly improves on standard backtracking under both average and worst-case performance. The global average of fast-tracking is roughly 50% that of the minmax guarantee of 7 iterations, and, since the ITP solver called made use of the 0.99 slack variable, the worst case performance over the test set is at most $\lceil 0.99 \rceil = 1$ iteration more than the minmax guarantee, i.e. at most $7 + 1 = 8$ iterations. Standard backtracking only attained a number of iterations equal to the minmax on one instance, and was outperformed by vanilla "geometric average" binary searching on every other instance.

Furthermore, by varying the values of β and the initial estimate $\boldsymbol{x}$, we verify that the differences in performance are affected too. Our preliminary estimates suggest that for values of β near 0.5 backtracking performs much worse than fast-tracking than what is reported in Table 4.1, multiplying by a factor of 10 the difference in average iteration count at it's peak value. For β near 0 or 1 the differences are kept roughly in the range of the ones reported in Table 4.1. Concerning the effect of

[4] The ITP parameters used were of $\kappa_1 = 0.1; \kappa_2 = 2$; and, a slack parameter of $N_0 = 0.99$ applied on the rescaled root-searching problem made to satisfy $b - a \leq 1$ in order to benefit of the guarantees of (Oliveira & Takahashi, 2020).

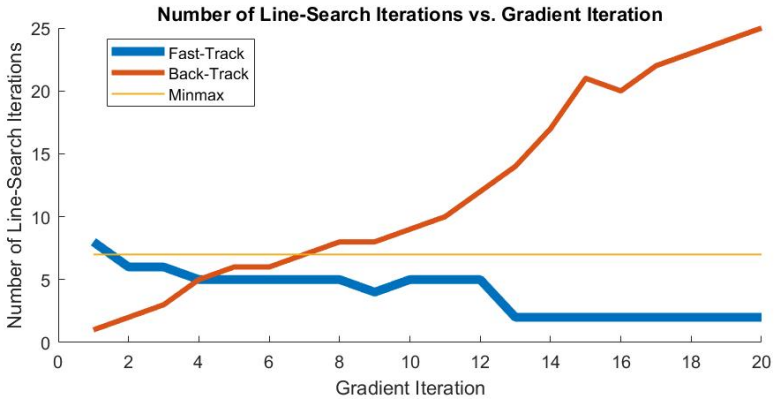


Fig. 4.1: Evolution of number of iterations after each gradient calclation

the initial estimate for $\boldsymbol{x}$, our experiments suggest that the the closer the initial estimate is to the stationary point $\boldsymbol{x}^*$ to which the gradient method converges, the greater the benefit of fast-tracking over backtracking, and when initiated far from $\boldsymbol{x}^*$ the difference in performance is reduced, but not reversed. Finally, analogous experiments were also performed providing the solvers with additional interpolation information and different values of x_0 and found no significant difference in the comparative performance reported above. Thus, these results have been kept out for brevity.

4.5 Discussion

The emphasis of "backtracking papers" does not typically lie on the three lines that construct and verify which point in the sequence $x_0, x_0\beta...$ first satisfies $g(x) \leq 0$. In fact, the construction of $g(\cdot)$, and the guarantees associated with the criteria $g(\cdot) \leq 0$, is typically where the contributions of those papers are found. Thus, perhaps justifiably so, it seems that not much research effort has been devoted to those three lines since they, informally speaking, "get the job done" and "some other paper can deal with it". Here, we do precisely this: we tackle the backtracking problem and find the most efficient way of completing this often overlooked task.

Here, we show a simple and proper construction of a procedure that finds $g(x) \leq 0$, and does so with optimal guarantees. A logarithmic speed-up is attained with respect to worst case, and a multi-logarithmic speed-up is attained with respect to asymptotic performance if hybrid interpolation based techniques are employed. These speed-ups are well reflected in experiments achieving roughly 50% to 80% time savings in each call to the inexact line-searching procedure.

References

Armijo, L. (1966). Minimization of functions having lipschitz continuous first

Table 4.1: Average number of function evaluations required to solve the inexact line-search problem in each iteration of a vanilla gradient descent for different loss functions. The numbers reported are the average obtained after 20 gradient steps under conditions where the minmax "geometric-average" binary-searching procedure would require exactly 7 iterations. Below, the symbol V stands for an identity matrix plus the Vandermonde matrix obtained in interpolation problems on n Chebyshev points; the vector $\boldsymbol{n}$ is defined as $[1, 2, ..., n]^T$, and every operation on $\boldsymbol{n}$ is done element-wise.

		Backtracking	Fast-tracking
Functions -			
Simple Quadratic	$\sum_i x_i^2$	12.2	4.0
High Degree Polynomial	$\sum_i x_i^{2i}$	10.8	4.8
Vandermonde Interpolation	$\boldsymbol{x}^T V \boldsymbol{x}$	16.2	3.9
Trigonometric 1	$\sum_i i \cos(x_i)$	7.0	4.8
Trigonometric 2	$\sum_i i \cos(\cos(x_i))$	12.0	4.0
Log-Poly	$2 \log \lvert\lvert \boldsymbol{x} - \boldsymbol{n}^{1/\boldsymbol{n}} \rvert\rvert_2$	27.2	2.7
Quartic	$\frac{1}{n}(\sum_i x_i)^4 + \lvert\sqrt{\boldsymbol{n}^T x}\rvert$	24.7	4.2
Interpolation w/ Regularizer	$\boldsymbol{x}^T V \boldsymbol{x} + \lvert\lvert \boldsymbol{x} - \sqrt{\boldsymbol{n}} \rvert\rvert_1$	49.5	3.5
Noisy Quadratic Hard	$\lvert\lvert \boldsymbol{x} \rvert\rvert_2^2 + 10^{-3} \sum_i \sin(i/x_i)$	26.8	3.0
Noisy Quadratic Easy	$\lvert\lvert \boldsymbol{x} \rvert\rvert_2^2 + 10^{-3} \sum_i \sin(10^3 i x_i)$	35.6	2.3
Global Average		**22.2**	**3.7**
Global Worst Case		**147**	**8**

partial derivatives. *Pacific Journal of Mathematics*, *16*(1), 1-3.

Boukis, C., Mandic, D. P., Constantinides, A. G., & Polymenakos, L. C. (2010, May). A modified armijo rule for the online selection of learning rate of the lms algorithm. *Digital Signal Processing*, *20*(3), 630–639. doi: https://doi.org/10.1016/j.dsp.2009.09.003

Boyd, S., & Vandenberghe, L. (2009). Convex optimization. In (7th ed.). Cambridge, UK: Cambridge University Press.

Burachik, R., Drummond, L. M. G., Iusem, A. N., & Svaiter, B. F. (1995, January). Full convergence of the steepest descent method with inexact line searches. *Optimization*, *32*, 137–146. doi: https://doi.org/10.1080/02331939508844042

Calatroni, L., & Chambolle, A. (2017, September). Backtracking strategies for accelerated descent methods with smooth composite objectives. *SIAM Journal on Optimization*, *29*, 1772–1798. doi: https://doi.org/10.1137/17M1149390

Chapra, S. C., & Canale, R. P. (2010). Numerical methods for engineers. In

(6th ed. ed., p. 202-220). New York, NY: McGraw-Hill Higher Education.

Gill, P. E., Murray, W., & Wright, M. H. (1997). *Practical optimization* (11th ed.). Academic Press.

Luenberger, D. G., & Ye, Y. (2018). Linear and nonlinear programming. In (4th ed.). Springer International Publishing.

Oliveira, I. F. D., & Takahashi, R. H. C. (2020). An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software*.

Oliveira, I. F. D., & Takahashi, R. H. C. (2021). Efficient solvers for armijo's backtracking problem.
(Pre-print available at `https://arxiv.org/abs/2110.14072`)

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: the art of scientific computing. In (6th ed. ed., p. 442-486). Cambridge, UK: Cambridge University Press.

Ralha, R. (2012). The geometric mean algorithm. *Applied Mathematics and Computation*, *219*(4), 1607-1615. doi: https://doi.org/10.1016/j.amc.2012.08.002

Ralha, R. (2018). Mixed precision bisection. *Mathematics in Computer Science*, *12*, 173–181. doi: https://doi.org/10.1007/s11786-018-0336-6

Shi, Z. J., & Shen, J. (2005). New inexact line search method for unconstrained optimization. *Journal of Optimization Theory and Applications*, *127*, 425–446. doi: 10.1007/s10957-005-6553-6

Truong, T. T., & Nguyen, H. T. (2021, September). Backtracking gradient descent method and some applications in large scale optimisation. part 2: Algorithms and experiments. *Applied Mathematics & Optimization*, *84*, 2557–2586. doi: https://doi.org/10.1007/s00245-020-09718-8

Vaswani, S., Mishkin, A., Laradji, I., Schmidt, M., Gidel, G., & Lacoste-Julieny, S. (2019). Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Proceedings of the 33rd conference on neural information processing systems - neurips).* Vancouver, Canada.

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM Review*, *11*(2), 226-235. doi: https://doi.org/10.1137/1011036

The multi-objective optimization problem

Summary. This chapter, contained in pages 87 to 117, is based on the the paper *An incremental descent method for multi-objective optimization* first submitted for consideration by a scientific journal on the 11th of September of 2021; and, as of the day of the submission of this thesis, this paper is under consideration for publication. The latest version of the paper (Oliveira & Takahashi, 2021b) can be found in the pre-print repository in the following address: `https://arxiv.org/abs/2105.11845`. In the current form presented below, minor changes are made to make for a more coherent read within the context of this thesis, however, it is kept self contained and an independent read of Chapter 5 is possible without need of reference to external material within the thesis. Any major content foreign to the original paper is highlighted with dark blue text, and, the remainder, where little or no alteration is made, we present the content in plain black text.

In this chapter we formally study the multi-objective optimization problem introduced in Chapter 1 and in particular the multi-objective equivalent of the steepest descent method. The multi-objective steepest descent method, under the assumption of lower-bounded objective functions with L-Lipschitz continuous gradients, requires $O(m/\epsilon^2)$ gradient and function computations to produce a measure of proximity to critical conditions akin to $||\nabla f(\boldsymbol{x})|| \leq \epsilon$ in the single-objective setting, where m is the number of objectives considered. We reduce this to $O(1/\epsilon^2)$ with a multi-objective incremental approach that has a computational cost that does not grow with the number of objective functions m.

5.1 Introduction

In this chapter we deal with the problem of finding *critical* points of multi-objective optimization problems. The multi-objective function $F : \mathbb{R}^n \to \mathbb{R}^m$ is defined as the concatenation of m single-objective functions as $F(\boldsymbol{x}) \equiv [f_1(\boldsymbol{x}), ..., f_m(\boldsymbol{x})]^T$ that are assumed to have L-Lipschitz continuous gradients, i.e. for some $L \geq 0$ and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and all $i = 1, ..., m$ the inequality $||\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})|| \leq L||\boldsymbol{x} - \boldsymbol{y}||$ holds.

A point $\boldsymbol{x} \in \mathbb{R}^n$ is[1] *critical* (also referred to as stationary) if and only if there does not exist a direction $\boldsymbol{v} \in \mathbb{R}^n$ such that $\nabla f_i(\boldsymbol{x})^T \boldsymbol{v} < 0$ for all $i = 1, ..., m$.

The search for stationary points is motivated by the fact that standard first order conditions for local optimality, akin to $\nabla f(\boldsymbol{x}) = 0$ in single-objective optimization, is that $\boldsymbol{x}$ be stationary (Luenberger & Ye, 2018). Descent direction algorithms, analogous to those employed in single-objective problems, have been devised and analysed extensively for the multi-objective setting, including: (i) *steepest descent* strategies (Fliege & Svaiter, 2000; Vieira, Takahashi, & Saldanha, 2012; Fliege, Vaz, & Vicente, 2019); (ii) *Newton* type methods (Fliege, Drummond, & Svaiter, 2009; Povalej, 2014); (iii) *projected gradient* strategies (Drummond & Iusem, 2004; Cruz, Pérez, & Melo, 2011); amongst others (Pérez & Prudente, 2019; Tanabe, Fukuda, & Yamashita, 2019; Gebken & Peitz, 2021; Moudden & Mouatasim, 2021). These generally follow the same structure as single-objective optimization strategies:

Algorithm 4: Descent direction method

```
1 initialize x̂ ∈ ℝ^n;
2 while not stop condition do
3 |   compute a descent direction d;
4 |   find an appropriate step-size α ∈ ℝ_+;
5 |   update the estimate x̂ ← x̂ + αd;
6 end
```

As in single-objective optimization, the multi-objective steepest descent method provides the standard performance benchmark to which other methods are typically compared to (Fliege & Svaiter, 2000; Fliege et al., 2009; Fukuda & Drummond, 2014; Fliege et al., 2019; Cocchi, Liuzzi, Lucidi, & Sciandrone, 2020). For any given estimate $\hat{\boldsymbol{x}}$, the steepest descent method defines $\boldsymbol{d}$ as $\boldsymbol{d} \equiv \boldsymbol{V}_s(\hat{\boldsymbol{x}})$ where

$$\boldsymbol{V}_s(\hat{x}) \equiv \arg\min_{\boldsymbol{V} \in \mathbb{R}^n} \max_{i=1,\dots,m} \nabla f_i(\hat{\boldsymbol{x}})^T \boldsymbol{V} + \frac{1}{2}||\boldsymbol{V}||^2. \tag{5.1}$$

If line 3 of Algorithm 4 is implemented with $\boldsymbol{d} \equiv \boldsymbol{V}_s(\hat{\boldsymbol{x}})$, and line 4 is implemented with step-size α produced by Armijo's backtracking procedure delineated in (Fliege & Svaiter, 2000) or the Fibonacci search of (Vieira et al., 2012), then, under mild technical assumptions, global convergence is guaranteed irrespective of the starting point, analogous to the guarantees of single-objective gradient descent. See Theorem 1 of (Fliege & Svaiter, 2000) and Theorem 3 of (Vieira et al., 2012) for the details. Furthermore, the computational complexity of the single-objective gradient descent method, which upperbounds the norm of the gradient of the objective function $||\nabla f(\boldsymbol{x})||$ by $O(1/\sqrt{k})$ after k steps, is also echoed by its multi-objective counterpart as the analogous measure of proximity to critical conditions $\min_{l=1,\dots,k} ||\boldsymbol{V}_s(\hat{\boldsymbol{x}}^l)||$ is also upper-bounded by $O(1/\sqrt{k})$ after k steps under the Lipschitz continuous gradient assumption; see Theorem 3.1 of (Fliege et al., 2019).

As mentioned in Section 1.2, to the best of our knowledge, this worst case query complexity, in the single-objective setting, is the best known derived under such assumptions. And, it is not improved on even with the addition of second order information, such as with the use of Newton's method; see Section 3 of (Cartis, Gould, & Toint, 2010) (though the use of a regularizer in addition to the second order information does improve the bound). Analogous convergence guarantees to those of

[1] Following the definition of (Fliege & Svaiter, 2000).

gradient descent in single-objective optimization are also known for convex functions as well as strong convex functions when the multi-objective steepest descent method is employed (Fukuda & Drummond, 2014; Fliege et al., 2019). Furthermore, similar results have been developed for Newton-type methods as well as projected gradient-type methods in the multi-objective optimization literature (Drummond & Iusem, 2004; Fliege et al., 2009; Cruz et al., 2011; Povalej, 2014).

Problem statement.

While multi-objective steepest descent recovers the $O(1/\sqrt{k})$ rate of convergence of single-objective problem solving, since the computation of $\boldsymbol{d}$ in each iteration requires querying all m gradients, and the production of α in both exact and inexact line searches requires querying all m functions per iteration, the overall query complexity of multi-objective steepest descent is still m times that of single-objective optimization; i.e. when we take into consideration the dependence on m the number of calls to gradient and function evaluations to upper-bound the measure of proximity to critical conditions by $||\boldsymbol{V}_s(\hat{\boldsymbol{x}})|| \leq \epsilon$ is more precisely of the order of $O(m/\epsilon^2)$. That is, with the same query complexity budget, gradient descent can produce solutions to each of the m single-objective minimization problems separately with similar ϵ worst case bounds as one run of the multi-objective steepest descent method. Thus, standard multi-objective steepest descent methods turns out to be extremely inefficient choices when the complexity with respect to the number of objective functions m is taken into consideration. Furthermore, notice that stationarity with respect to multi-objective optimization is a relaxation of stationarity with respect to single-objective optimization, thus the query complexity of multi-objective optimization should actually be no greater than that of single-objective optimization. Hence, the motivation of this chapter is to address this complexity gap, as well as to investigate alternative descent direction definitions that might provide improved properties in terms of rates of convergence, robustness and overall simplicity.

For this, we propose a brand new descent method that attains the $O(1/\sqrt{k})$ rate of convergence of single objective optimization that does not have an increasing cost with respect to increasing values of m; i.e. we produce measures of proximity to critical conditions akin to $||\nabla f(\boldsymbol{x})|| \leq \epsilon$ with only $O(1/\epsilon^2)$ queries irrespective of the value of m. This is attained by means of two advancements: first, a brand new generalization of the steepest-descent direction is proposed, which we name the *central descent direction* and denote by $\boldsymbol{V}_c$; and second, a brand new incremental approach to the canonical structure of Algorithm 1 is proposed which is shown to substantially reduce the cost of each iteration. It is the combination of these two improvements that produce the optimal $O(1/\sqrt{k})$ rate of convergence. To motivate our new descent direction, we show that $\boldsymbol{V}_c$ has multiple advantages over the classical definition of $\boldsymbol{V}_s$, as stated in (5.1), including robustness guarantees that cannot be ensured by (5.1) as well as natural metric results that provide measures of proximity to critical conditions, akin to $||\nabla f(\boldsymbol{x})|| \leq \epsilon$, obtained by computation of the norm of $\boldsymbol{V}_c$ alone. And, to motivate our incremental approach, we provide our main result: a global convergence guarantee for lower-bounded objective functions with Lipschitz continuous gradients.

Chapter layout.

The remainder of this chapter is divided into five parts: Sections 5.2 through 5.6. Section 5.2 covers the definition and properties of the central descent direction. There, robustness guarantees are given to motivate the use of $\boldsymbol{V}_c$ as well as metric results that provide measures of proximity to critical conditions derived from $||\boldsymbol{V}_c||$ which are akin to classical single-objective measures. Section 5.3 defines the incremental central descent method and contains our main result. There we provide the main query complexity guarantee as well as discuss the implications of our result. In Sections 5.4 and 5.5, respectively, we illustrate the method proposed with a toy example and then provide a self contained analysis of line-searching with the techniques here delineated; both of which provides an illustration of the convergence (from different angles) of the techniques here proposed. Finally, Section 5.6 summarizes our findings, situates it within the larger framework of multi-objective optimization literature and points to future work.

5.2 The central descent direction

Given a non-critical point $\boldsymbol{x} \in \mathbb{R}^n$, any $\boldsymbol{v}$ that satisfies $\nabla f_i(\boldsymbol{x})^T \boldsymbol{v} < 0$ for all $i = 1, ..., m$ is called a descent direction. In this chapter we propose one specific uniquely defined descent direction, which we call the *central descent direction* and denote by $\boldsymbol{V}_c(\boldsymbol{x})$. It is defined as

$$\boldsymbol{V}_c(\boldsymbol{x}) \equiv \left\{ \begin{array}{ll} \text{argmin} & \frac{1}{2}||\boldsymbol{V}||^2 \\ \text{s.t.} & \nabla f_i(\boldsymbol{x})^T \boldsymbol{V} \leq -||\nabla f_i(\boldsymbol{x})|| \text{ for all i = 1, ..., m} \end{array} \right\}; \tag{5.2}$$

i.e. it is the smallest vector $\boldsymbol{V} \in \mathbb{R}^n$, as measured by the $\mathcal{L}_2$ norm, with a projected component in the direction of the negative normalized gradients of (at least) a unit: $\frac{\nabla f_i(\boldsymbol{x})^T}{||\nabla f_i(\boldsymbol{x})||} \boldsymbol{V} \leq -1$. And, the following property, which uniquely identifies $\boldsymbol{V}_c/||\boldsymbol{V}_c||$, provides the first robustness guarantee associated with the descent direction $\boldsymbol{V}_c$.

Theorem 13 (Robustness to Perturbation). *The descent direction* $\boldsymbol{V}_c(\boldsymbol{x})/||\boldsymbol{V}_c(\boldsymbol{x})||$ *is the unit vector maximally distant from the set of non-descent directions* $\mathbb{R}^n \setminus \{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x})^T \boldsymbol{v} \leq 0 \text{ for } i = 1,...,m\}$.

Proof. Given a unit vector $\boldsymbol{u}$ in $\{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x})^T \boldsymbol{v} \leq 0 \text{ for i = 1,...,m}\}$, the $\mathcal{L}_2$ distance of $\boldsymbol{u}$ to $\mathbb{R}^n \setminus \{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x})^T \boldsymbol{v} \leq 0 \text{ for i = 1,...,m}\}$ is given by $d(\boldsymbol{u}) = \min_{i=1,...,m} \nabla f_i(\boldsymbol{x})^T \boldsymbol{u}/||\nabla f_i(\boldsymbol{x})||$. Therefore, the maximization problem that defines the unit vector maximally distant from non-descent directions can be expressed as

$$\begin{array}{ll} \max_{z,\boldsymbol{u}} & z \\ \text{s.t.} & z||\nabla f_i(\boldsymbol{x})|| \leq \nabla f_i(\boldsymbol{x})^T \boldsymbol{u} \text{ for } i = 1, ..., m; \\ & ||\boldsymbol{u}|| = 1; \end{array}$$

which, by defining $\boldsymbol{V} \equiv \boldsymbol{u}/-z$ the objective $\max z$ turns out to be equivalent to $\min ||\boldsymbol{V}||_2$, and, the constraint $z||\nabla f_i(\boldsymbol{x})|| \leq \nabla f_i(\boldsymbol{x})^T \boldsymbol{u}$ for $i = 1, ..., m$; turns out to be equivalent to $-||\nabla f_i(\boldsymbol{x})|| \geq \nabla f_i(\boldsymbol{x})^T \boldsymbol{V}$ for $i = 1, ..., m$. The remaining constraint of $||\boldsymbol{u}|| = 1$ can be dropped by recognizing that $|z| = 1/||\boldsymbol{V}||$ is non-restrictive on the remaining variables. □

Theorem 13 ensures that the central descent direction is maximally distant from non-descent directions as measured by the $\mathcal{L}_2$ norm. And, as a consequence, the unavoidable numerical errors and/or intentional approximations in the calculation of $\boldsymbol{V}_c/||\boldsymbol{V}_c||$ are less likely to produce a non-descent direction when compared to any other directions in the cone of descent directions $\left\{\boldsymbol{v} \mid \nabla f_i(\boldsymbol{x})^T\boldsymbol{v} < 0 \;\forall\; i = 1, ..., m\right\}$. Hence, the direction $\boldsymbol{V}_c/||\boldsymbol{V}_c||$ seems to be a natural choice for the descent direction $\boldsymbol{d}$ in line 3 of Algorithm 4 when numerical errors and approximations are taken into consideration. In contrast, the unit vector $\boldsymbol{V}_s/||\boldsymbol{V}_s||$ not only does not enjoy any such type of guarantee, but it can also be made arbitrarily close to the boundaries of the cone of descent directions with a simple rescaling of the objectives with potentially devastating effects (a property that $\boldsymbol{V}_c/||\boldsymbol{V}_c||$ is also immune to as is discussed further ahead following Theorem 14).

Remark 10. A similar claim can be done if only orthogonal perturbations to the descent direction are considered, that is, the direction $\boldsymbol{V}_c/||\boldsymbol{V}_c||$ also happens to be maximally distant from non-descent directions if the distance of $\boldsymbol{u}$ in $\{\boldsymbol{v}$ s.t. $\nabla f_i(\boldsymbol{x})^T\boldsymbol{v} \leq 0$ for i = 1, ..., m$\}$ to non descent directions $\mathbb{R}^n \backslash \{\boldsymbol{v}$ s.t. $\nabla f_i(\boldsymbol{x})^T\boldsymbol{v} \leq 0$ for i = 1,...,m$\}$ is measured only over the orthogonal plane defined by $\{\boldsymbol{x}$ s.t. $(\boldsymbol{x} - \boldsymbol{u})^T\boldsymbol{u} = 0\}$. And hence, one way or another, the central descent direction $\boldsymbol{V}_c$ seems to be a robust choice when numerical or estimation errors are taken into consideration.

The following result further ensures that both the direction $\boldsymbol{V}_c/||\boldsymbol{V}_c||$ as well as the norm $||\boldsymbol{V}_c||$ are robust to non-linear rescaling of the objective functions. This property, as discussed ahead, ensures that both $\boldsymbol{V}_c$ as well as the metrics we derive from $||\boldsymbol{V}_c||$ (to be defined ahead) are a stable choice across different natural formulations of any given multi-objective optimization problem. This property is a key difference between $\boldsymbol{V}_c$ and $\boldsymbol{V}_s$ as well as previous directions proposed in the literature thus far.

Theorem 14 (Robustness to Rescaling). *Given a multi-objective problem with* $F(\boldsymbol{x}) \equiv [f_1(\boldsymbol{x}), ..., f_m(\boldsymbol{x})]$ *and a collection of strictly increasing and differentiable transformations* $g_i : \mathbb{R} \to \mathbb{R}$ *for* $i = 1, ..., m$, *define* $G(\boldsymbol{x}) \equiv [g_1(f_1(\boldsymbol{x})), ..., g_m(f_m(\boldsymbol{x}))]$. *Then, the central descent direction* $\boldsymbol{V}_c(\boldsymbol{x})$ *calculated with respect to* $F(\boldsymbol{x})$ *is equal to the central descent direction calculated with respect to* $G(\boldsymbol{x})$.

Proof. The proof follows immediately from the chain rule: $\frac{\partial}{\partial x_j}\left(g_i(f_i(\boldsymbol{x}))\right) = g_i'(f_i(\boldsymbol{x}))\cdot \frac{\partial}{\partial x_j} f_i(\boldsymbol{x})$. Applying the chain rule to the central descent direction on the monotonically transformed problem we find that $\text{argmin}\{\frac{1}{2}||\boldsymbol{V}||^2$ s.t. $g_i'(f_i(\boldsymbol{x}))\nabla f_i(\boldsymbol{x})^T\boldsymbol{V} \leq -||g_i'(f_i(\boldsymbol{x}))\nabla f_i(\boldsymbol{x})||\}$ is equal to $\text{argmin}\left\{\frac{1}{2}||\boldsymbol{V}||^2 \quad \text{s.t. } \nabla f_i(\boldsymbol{x})^T\boldsymbol{V} \leq -||\nabla f_i(\boldsymbol{x})||\right\}$ since the terms $g_i'(f_i(\boldsymbol{x}))$ cancel out. □

Thus, Theorem 14 ensures that the central descent direction is invariant to changes on the scales of the objective functions. In contrast, the steepest descent direction in (5.1) is not only sensitive to monotone transformations, but even, sensitive to simple linear transformations of the objectives; i.e. the direction $\boldsymbol{V}_s(\boldsymbol{x})$ obtained by considering the objective functions in $F(\boldsymbol{x}) \equiv [f_1(\boldsymbol{x}), f_2(\boldsymbol{x})]^T$ is not the same as the one obtained by considering $F(\boldsymbol{x}) \equiv [f_1(\boldsymbol{x}), \kappa f_2(\boldsymbol{x})]^T$ for $\kappa > 0$ with $\kappa \neq 1$. This property can have two devastating effects: it can (i) drastically warp the path taken by the descent direction algorithm if the scales are not a-priori fine tuned; and

it can (ii) make $\boldsymbol{V}_s/||\boldsymbol{V}_s||$ arbitrarily close to the boundary of the cone of descent directions making small numerical or estimation errors sufficient to produce a non-descent direction $\boldsymbol{d}$. These effects might be difficult to avoid since multi-objective optimization techniques are typically adopted precisely when the relative weights of the objectives are unknown.

We illustrate the contrast between $\boldsymbol{V}_s$ and $\boldsymbol{V}_c$ in Figures 5.1 to 5.4, which refer to a bi-objective problem with objective functions given by:

$$f_1(\boldsymbol{x}) \equiv (x_1+2)^2 + 3x_2^2 \qquad f_2(\boldsymbol{x}) \equiv 3x_1^2 + (x_2+2)^2 \tag{5.3}$$

Figure 5.1 depicts the level sets of $||\boldsymbol{V}_s||$ (the standard parameter derived from $\boldsymbol{V}_s$ used to measure of proximity to critical conditions) as well as the stream-lines (the curves produced by "releasing a particle to flow in the direction of $\boldsymbol{V}_s$") of the problem when the objective functions are multiplied by different constants. The warping effect can over-weigh one objective over the other producing contour-lines and stream-lines that can even parallel the efficient set, making the descent direction algorithm unnecessarily "go around" close-by solutions to converge at distant ones. This effect can be dramatically intensified with different monotone transformations. Figure 5.2 shows both the contour-lines as well as the stream-lines induced by $\boldsymbol{V}_c$ and, as can be seen, the curves follow a more natural path towards a close-by efficient solution irrespective of the scale adopted in the representation of the objective functions. Finally, Figure 5.3 depicts $\boldsymbol{V}_s$ and $\boldsymbol{V}_c$ for varying values of $||\nabla f_i(\boldsymbol{x})||$ for $i = 1, 2$, while Figure 5.4 shows the variation of $\boldsymbol{V}_c$ for different angles between those gradients. We can now provide our metric results.

One of the key properties of the central descent direction is that the conditions for a point in $\mathbb{R}^n$ to be critical can be restated in terms of the central descent direction:

Lemma 2. *A point $\boldsymbol{x} \in \mathbb{R}^n$ is critical if and only if (i) at least one gradient $\nabla f_i(\boldsymbol{x})$ is null; or (ii) the central descent direction $\boldsymbol{V}_c(\boldsymbol{x})$ defined in (5.2) is empty; or (iii) both.*

Proof. Follows immediately from the definition of critical points. □

And, furthermore:

Lemma 3. *Let $\{\boldsymbol{x}_k\}_{k=1,\ldots,\infty}$ be a sequence of non-critical points converging to $\boldsymbol{x}_\infty \in \mathbb{R}^n$. Then, if neither the gradients $\nabla f_i(\boldsymbol{x}_k)$ goes to zero with $k \to \infty$, we have that either (i) the accumulation point $\boldsymbol{x}_\infty$ is critical and $||\boldsymbol{V}_c(\boldsymbol{x}_k)|| \to \infty$; or (ii) the accumulation point $\boldsymbol{x}_\infty$ is not critical and $\boldsymbol{V}_c(\boldsymbol{x}_k)$ converges to $\boldsymbol{V}_c(\boldsymbol{x}_\infty)$.*

Proof. Refer to the appendix section 5.A.1. □

Thus, while in single-objective optimization convergence to critical conditions is typically measured in terms of upper-bounding the norm of the gradient, from Lemmas 2 and 3, in the multi-objective setting proximity to critical conditions is perhaps more naturally measured in terms of two orthogonal conditions. A point $\boldsymbol{x} \in \mathbb{R}^n$ is near-critical if either:

(a) $\min_{i=1,\ldots m}\{||\nabla f_i(\boldsymbol{x})||\}$ is small; or
(b) $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ is small.

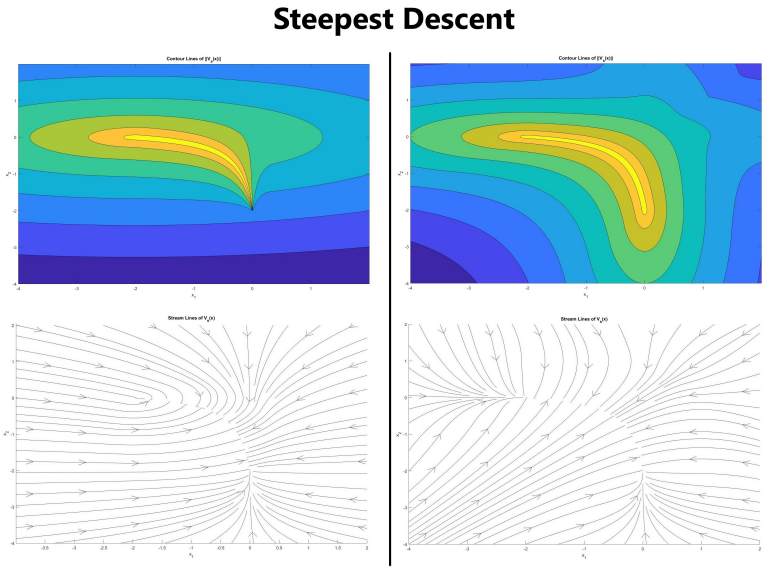


Fig. 5.1: The top two images depict the level sets of the measure of proximity induced by the steepest descent (5.1), i.e the level sets of $||\boldsymbol{V}_s||^2$. Since the steepest descent direction is not metric-independent, the level sets of the measure of proximity can be warped (as it is on the left side) when the objectives are not measured with some comparable scale (as on the right side). The path taken by the steepest descent method is affected by the warping and can produce curves that are arbitrarily close to the efficient set and yet parallel to it. The path of the steepest descent method (when the steps are sufficiently small) will follow the stream lines depicted on the two lower images.

Condition (a) indicates that the current solution is near-optimal with respect to at least one of the single-objective sub-problems and (b) indicates that it is near-optimal in the Pareto sense. Of course, these can be easily mixed into one combined metric $\min_{i=1,...m}\{||\nabla f_i(\boldsymbol{x})||\}/||\boldsymbol{V}_c(\boldsymbol{x})||$; however, the separation of these conditions allows for a better understanding of candidate solutions $\boldsymbol{x} \in \mathbb{R}^n$. In Figure 5.5 we illustrate a multi-objective optimization problem and the regions that approximate the efficient set with the measures here delineated.

While condition (a) is well understood, what is left now is to provide metric results to better understand (b). That is, in the following two results we characterize the the measure of proximity to critical conditions given by $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ in terms of more familiar well understood mathematical tools.

Theorem 15 (Proximity measure in gradient space). *For any non-critical point* $\boldsymbol{x} \in \mathbb{R}^n$ *define* $\mathcal{S}^+ \equiv \{\nabla f_i(\boldsymbol{x})/||\nabla f_i(\boldsymbol{x})||$ *for* $i = 1, ..., m\}$ *and* $\mathcal{S}^- \equiv \{-\nabla f_i(\boldsymbol{x})/||\nabla f_i(\boldsymbol{x})||$ *for* $i = 1, ..., m\}$. *We have that*

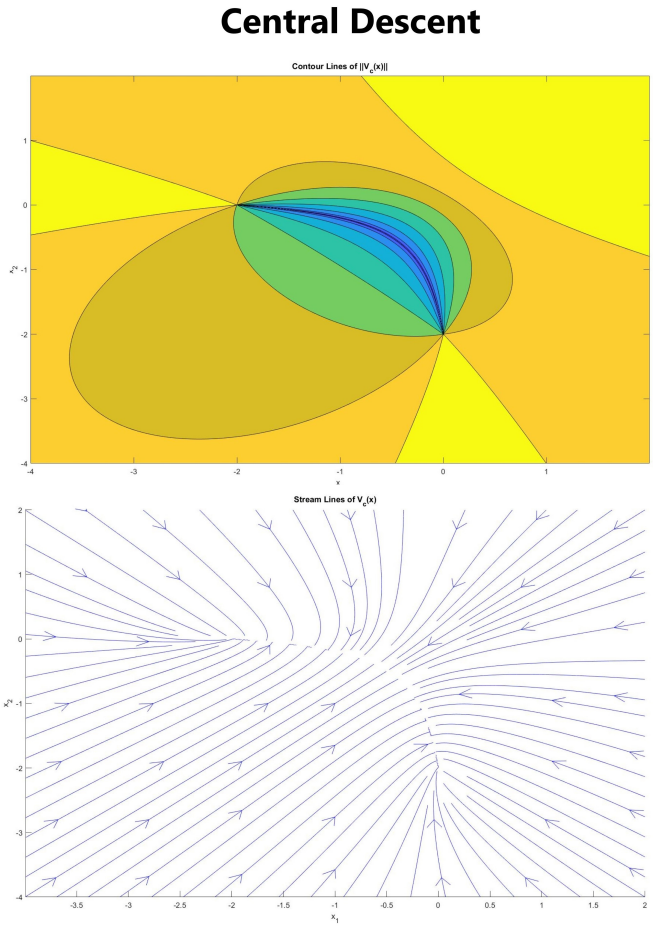


Fig. 5.2: The top image depicts the level sets of $||\boldsymbol{V}_c(\boldsymbol{x})||$ (the same as the bottom right of Figure 5.5) and the bottom image depicts the stream lines induced by the central descent direction. Notice that since the central descent direction is unaffected by the rescaling of the objective functions, these level sets are not warped by different scales/representations of the same collection of objectives. Also, notice that proximity to the minima of the single-objective functions is not measured by this metric, as the level curves meet at a sharp angle at each individual minimum; instead it measures the angle between the gradients independently of their norms. This produces (on the bottom figure) stream lines that are metric independent and are not warped by arbitrary scaling choices.

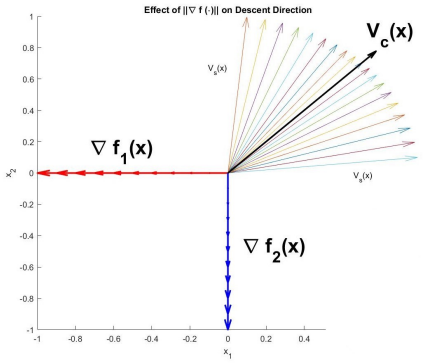


Fig. 5.3: The background image depicts the effect of changing the scales of the objective functions. The adoption of different scales on the objective functions alter the sizes of the gradients $\boldsymbol{g}_i \equiv \nabla f_i(\boldsymbol{x})$ for $i = 1, 2$, but not the directions of the gradients. Both size and direction of the central descent are unaffected by these changes in scale, however the direction of $\boldsymbol{V}_s$ described in (5.1) is affected by these changes in scale. The size of $\boldsymbol{V}_s$ is also affected by such changes, however, here we only depict the projections of the vectors $\boldsymbol{V}_s$ and $\boldsymbol{V}_c$ to the unit ball for ease of visualization.

The maximum margin of separation between $\mathcal{S}^+$ *and* $\mathcal{S}^-$ *is equal to* $2/||\boldsymbol{V}_c(\boldsymbol{x})||$. (5.4)

Proof. It is well known that, given vectors $\boldsymbol{u}_i^+$ and $\boldsymbol{u}_i^-$ for $i = 1, ..., m$, the separating hyperplane $\{\boldsymbol{x} \mid \boldsymbol{w}^T\boldsymbol{x} + b = 0\}$ with maximum margin is obtained by solving the following problem (refer to the Support Vector Machine literature initiated in (Cortes & Vapnik, 1995) and subsequent literature):

$$(\boldsymbol{w}^*, b^*) \equiv \left\{ \begin{array}{lll} \text{argmin} & \frac{1}{2}||\boldsymbol{w}||^2 & \\ \text{s.t.} & \boldsymbol{w}^T\boldsymbol{u}_i^+ + b \geq 1 & \text{for } i = 1, ..., m \\ & \boldsymbol{w}^T\boldsymbol{u}_i^- + b \leq -1 & \text{for } i = 1, ..., m \end{array} \right. \tag{5.5}$$

It is also known that the maximum margin δ produced by (5.5) is equal to $\delta = 2/||\boldsymbol{w}^*||$. Now since in the separation of $\mathcal{S}^+$ and $\mathcal{S}^+$ we have $\boldsymbol{u}_i^+ \equiv \nabla f_i(\boldsymbol{x})/||\nabla f_i(\boldsymbol{x})|| = -\boldsymbol{u}_i^-$ the constraint $\boldsymbol{w}^T\boldsymbol{u}_i^- + b \leq -1$ is equivalent to $\boldsymbol{w}^T\boldsymbol{u}_i^+ - b \geq 1$ which in turn when combined with the constraint $\boldsymbol{w}^T\boldsymbol{u}_i^+ + b \geq 1$ implies that the variable b can be dropped from the formulation thus simplifying (5.5) to

$$\boldsymbol{w}^* = \left\{ \begin{array}{lll} \text{argmin} & \frac{1}{2}||\boldsymbol{w}||^2 & \\ \text{s.t.} & \boldsymbol{w}^T\boldsymbol{u}_i^+ \geq 1 & \\ & i = 1, ..., m & \end{array} \right\} = \left\{ \begin{array}{lll} \text{argmin} & \frac{1}{2}||\boldsymbol{w}||^2 & \\ \text{s.t.} & \nabla f_i(\boldsymbol{x})^T\boldsymbol{w} \geq ||\nabla f_i(\boldsymbol{x})|| & \\ & i = 1, ..., m & \end{array} \right\} \tag{5.6}$$

and by defining $\boldsymbol{V} \equiv -\boldsymbol{w}$ and plugging it back into (5.6) we recover the definition of $\boldsymbol{V}_c(\boldsymbol{x})$. Thus, the maximum margin δ is equal to $2/||\boldsymbol{V}_c(\boldsymbol{x})||$. □

Hence, Theorem 15 provides a natural interpretation of the measure of proximity to critical conditions $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ in terms of the domain of gradients: it measures

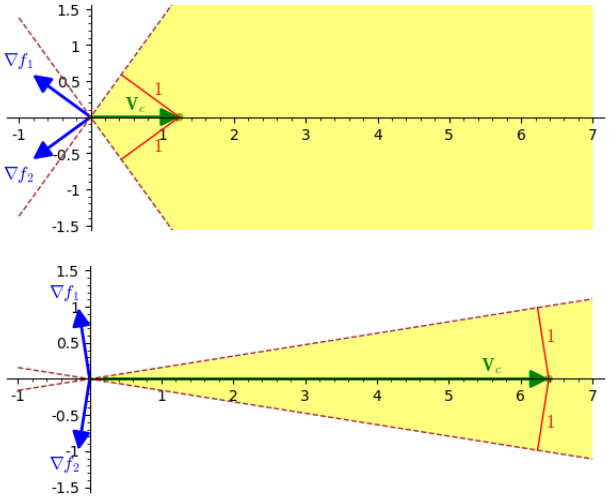


Fig. 5.4: The images depict the geometric construction of the vector $\boldsymbol{V}_c$ for different angles between $\nabla f_1(\boldsymbol{x})$ and $\nabla f_2(\boldsymbol{x})$, considering a point $\boldsymbol{x} = \boldsymbol{0}$. The cone of descent directions is shaded in yellow. The central descent direction can be obtained by bisecting the angle between $-\nabla f_1(\boldsymbol{x})$ and $-\nabla f_2(\boldsymbol{x})$. The $\boldsymbol{V}_c(\boldsymbol{x})$ is obtained as the point which is equidistant to the edges of the cone of descent directions and is situated at a distance of 1 to those edges. The longer the vector depicting $\boldsymbol{V}_c(\cdot)$ the closer $\nabla f_1(\boldsymbol{x})$ and $\nabla f_2(\boldsymbol{x})$ are from pointing in opposite directions, and the nearer is $\boldsymbol{x}$ to become a critical point.

the maximum margin of separation between the normalized gradients and their negatives. Naturally, when separation of the gradients and the negative of the gradients cannot be attained with a hyper-plane, then the point $\boldsymbol{x}$ must be critical, and, if it can, the larger the separation the more the directions are "in agreement" as to one direction that can decrease all objectives.

In the following we provide a similar result that ties in the measure $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ directly with distance of $\boldsymbol{x}$ to the set of critical points in the solution space. For this, more regularity must be assumed on the objective functions; in this case μ-strong convexity. We find that:

Theorem 16 (Proximity measure in decision space). *If the objective functions are assumed to be μ-strong convex then the distance δ of any estimate $\boldsymbol{x} \in \mathbb{R}^n$ to the set of critical points is upper-bounded by*

$$\delta \leq \left(\tfrac{2}{\mu} \max_{i=1,\ldots,m} ||\nabla f_i(\boldsymbol{x})|| \right) \frac{1}{||\boldsymbol{V}_c(\boldsymbol{x})||}. \tag{5.7}$$

Proof. From the definition of μ-strong convexity we have that for some $\mu > 0$

$$f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y}) + \nabla f_i(\boldsymbol{x})^T(\boldsymbol{x} - \boldsymbol{y}) - \frac{\mu}{2}||\boldsymbol{x} - \boldsymbol{y}||^2 \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n; \tag{5.8}$$

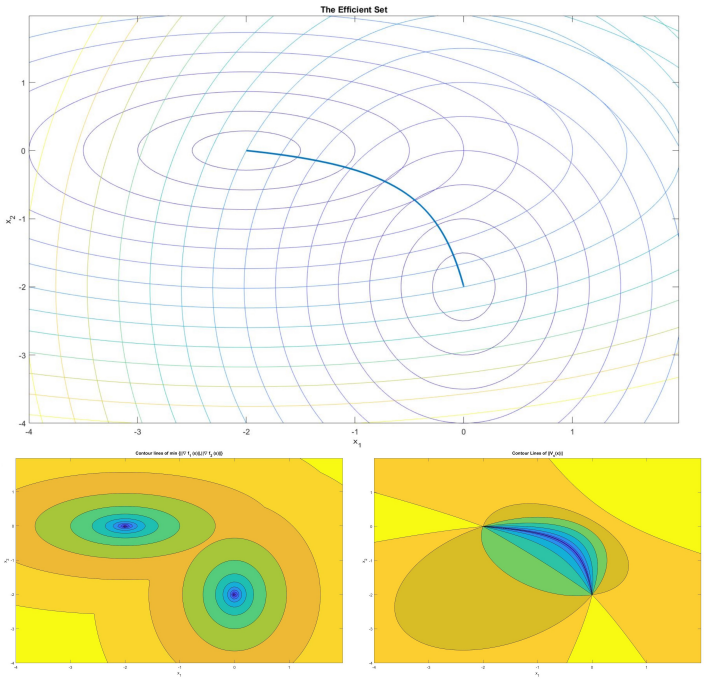


Fig. 5.5: The top figure depicts the level curves and the efficient set of the multi-objective optimization problem with objective functions $f_1(\boldsymbol{x}) \equiv (x_1 + 2)^2 + 3x_2^2$ and $f_2(\boldsymbol{x}) \equiv 3x_1^2 + (x_2 + 2)^2$. The bottom left depicts the level curves of $\min\{||\nabla f_1(\boldsymbol{x})||; ||\nabla f_1(\boldsymbol{x})||\}$ and the bottom right depicts the level curves of $||\boldsymbol{V}_c(\boldsymbol{x})||$. Both measures are complementary to quantify proximity to critical conditions; the $\min\{||\nabla f_1(\boldsymbol{x})||; ||\nabla f_1(\boldsymbol{x})||\}$ measures proximity to local minima of the single-objective sub-problems, and, $||\boldsymbol{V}_c(\boldsymbol{x})||$ measures proximity to intermediate solutions of the multi-objective problem.

Now given any $\boldsymbol{x} \in \mathbb{R}^n$ consider the collection of points $\boldsymbol{y}$ such that $f_i(\boldsymbol{y}) \leq f_i(\boldsymbol{x})$; for such points we may rearrange the terms in equation (5.8) to find that

$$0 \leq f_i(\boldsymbol{x}) - f_i(\boldsymbol{y}) \leq \nabla f_i(\boldsymbol{x})^T(\boldsymbol{x} - \boldsymbol{y}) - \frac{\mu}{2}||\boldsymbol{x} - \boldsymbol{y}||^2;$$

and therefore every $\boldsymbol{y}$ such that $f_i(\boldsymbol{y}) \leq f_i(\boldsymbol{x})$ must satisfy $||\boldsymbol{x}-\boldsymbol{y}||^2 \leq \frac{2}{\mu}\nabla f_i(\boldsymbol{x})^T(\boldsymbol{x}-\boldsymbol{y})$. Furthermore, since there must exist at least one critical point $\boldsymbol{y}^*$ such that $f_i(\boldsymbol{y}^*) \leq f_i(\boldsymbol{x})$ we may therefore upper-bound the distance δ of $\boldsymbol{x}$ to the critical set by

$$\delta \leq \left\{ \begin{array}{ll} \max_{\boldsymbol{y}\in\mathbb{R}^n} & ||\boldsymbol{y} - \boldsymbol{x}|| \\ \text{s.t.} & ||\boldsymbol{y} - \boldsymbol{x}||^2 \leq \frac{2}{\mu}\nabla f_i(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) \text{ for each } i = 1, ..., m \end{array} \right\}$$

which is then upper-bounded by

$$\leq \left\{ \begin{array}{ll} \max_{\boldsymbol{y}\in\mathbb{R}^n} & ||\boldsymbol{y}-\boldsymbol{x}|| \\ \text{s.t.} & ||\boldsymbol{y}-\boldsymbol{x}||^2 \leq \left(\frac{2}{\mu}\max_i ||\nabla f_i(\boldsymbol{x})||\right)\frac{\nabla f_i(\boldsymbol{x})^T}{||\nabla f_i(\boldsymbol{x})||}(\boldsymbol{y}-\boldsymbol{x}) \text{ for } i=1,...,m \end{array} \right\}$$

and by defining $K \equiv \frac{2}{\mu}\max_{i=1,...,m}||\nabla f_i(\boldsymbol{x})||$, and defining $\boldsymbol{d} \equiv \boldsymbol{y}-\boldsymbol{x}$ and $t \equiv ||\boldsymbol{d}||$ we obtain

$$\leq \left\{ \begin{array}{ll} \max_{\boldsymbol{d}\in\mathbb{R}^n, t\in\mathbb{R}} & t \\ \text{s.t.} & t^2 \leq tK\frac{\nabla f_i(\boldsymbol{x})^T}{||\nabla f_i(\boldsymbol{x})||}\frac{\boldsymbol{d}}{||\boldsymbol{d}||} \text{ for each } i=1,...,m \\ & t = ||\boldsymbol{d}|| \end{array} \right\}$$

which can be further simplified to

$$= \left\{ \begin{array}{ll} \max_{\boldsymbol{v}\in\mathbb{R}^n, t\in\mathbb{R}} & t \\ \text{s.t.} & 1 \leq \frac{\nabla f_i(\boldsymbol{x})^T}{||\nabla f_i(\boldsymbol{x})||}\left(\frac{K}{t}\boldsymbol{v}\right) \text{ for each } i=1,...,m \\ & ||\boldsymbol{v}|| = 1 \end{array} \right\}$$

and by defining $\boldsymbol{V} \equiv -\frac{K}{t}\boldsymbol{v}$ we have

$$= \left\{ \begin{array}{ll} \max_{\boldsymbol{V}\in\mathbb{R}^n} & K/||\boldsymbol{V}|| \\ \text{s.t.} & -||\nabla f_i(\boldsymbol{x})|| \geq \nabla f_i(\boldsymbol{x})^T\boldsymbol{V} \text{ for each } i=1,...,m \end{array} \right\}$$

which, of course, has $\boldsymbol{V}_c(\boldsymbol{x})$ as it's unique solution. □

Theorem 13, under the assumption of strong convexity, provides an upper-bound on the distance to the set of critical points as a function of $1/||\boldsymbol{V}_c(\boldsymbol{x})||$. In single-objective optimization, a similar inequality plays a role in motivating the use of $||\nabla f(\boldsymbol{x})||$ as a measure of proximity to critical conditions since the distance δ can be shown to be upper-bounded by[2]

$$\delta \leq \frac{1}{\mu}||\nabla f(\boldsymbol{x})||.$$

And thus, for sufficiently well behaved functions, small values of $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ ensure proximity to critical conditions in a much similar fashion that small values of $||\nabla f(\boldsymbol{x})||$ does in single-objective optimization.

In the absence of strong convexity $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ still provides a very clear measure of proximity when considering the domain of the gradients as ensured by Theorem 15. And, in the remainder of this chapter we no longer make use of this assumption of strong convexity, but instead only that the gradients are L-Lipschitz continuous, and, further ahead, we will also assume that the functions are lower-bounded by some unknown constant.

5.3 Incremental central descent method

In this section we will analyse the following algorithm which requires an initial estimate $\hat{\boldsymbol{x}} \in \mathbb{R}^n$ provided by the user, as well as specifications for functions SELECT_INDICES(·) and STEP_SIZE(·) discussed ahead.

[2] See (ii) of Lemma 3 in (Zhou, 2018).

Algorithm 5: The central descent algorithm

initialize $\hat{\boldsymbol{x}} \in \mathbb{R}^n$ and set $k \leftarrow 0$ and $\hat{\boldsymbol{g}}_1, ..., \hat{\boldsymbol{g}}_m \leftarrow \boldsymbol{0} \in \mathbb{R}^n$;
while *not stop condition* **do**
  $\mathcal{S} \leftarrow$SELECT_INDICES$(\cdot)$ and update $\hat{\boldsymbol{g}}_i \leftarrow \nabla f_i(\hat{x})$ for each $i \in \mathcal{S}$;
  $\hat{\boldsymbol{V}} \leftarrow \text{argmin}\{||\boldsymbol{V}||^2 \text{ s.t. } \hat{\boldsymbol{g}}_i^T \boldsymbol{V} \leq -||\hat{\boldsymbol{g}}_i|| \text{ for } i = 1, ..., m\}$;
  $\hat{\alpha} \leftarrow$STEP_SIZE$(\cdot)$;
  $\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} + \hat{\alpha}\hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}|| \quad \text{and} \quad k \leftarrow k + 1$;
end

The *central descent algorithm* is a generalization of the classical descent direction algorithm which encompasses both single-objective as well as multi-objective patterns. If function SELECT_INDICES$(\cdot)$ defines $\mathcal{S}$ as $\mathcal{S} \equiv \{1\}$ and STEP_SIZE$(\cdot)$ is taken as a classical line searching technique then Algorithm 5 reduces to traditional single-objective optimization on $f_1(\boldsymbol{x})$. Alternatively, if SELECT_INDICES$(\cdot)$ defines $\mathcal{S}$ as the collection of all indices $\mathcal{S} \equiv \{1, ..., m\}$, then, at a cost of computing the gradients of all m objective functions per iteration, we produce a full multi-objective descent direction method similar to the steepest descent methods of (Fliege & Svaiter, 2000) and (Vieira et al., 2012). Here, we are not interested in either of these extreme cases, but rather, in incremental approaches which produce sparse samplings of the gradients $\nabla f_1(\cdot), ..., \nabla f_m(\cdot)$ in order to estimate $\boldsymbol{V}_c$ instead of computing it exactly.

In our main result ahead SELECT_INDICES$(\cdot)$ simultaneously produces a sparse sampling of the gradients $\nabla f_1(\cdot), ..., \nabla f_m(\cdot)$ while also keeping track of one *anchor* function for the sake of performing inexact line searches which, as we show, is sufficient to ensure convergence to the critical set. Further ahead we discuss these decisions and point to other alternatives. In the following, $\beta \in (0, 1)$ is a user provided sufficient decrease parameter and SELECT_INDICES$(\cdot)$ and STEP_SIZE$(\cdot)$ are defined as

Incremental Anchored Sampling with Inexact Search:

$$\begin{aligned}
&\text{SELECT_INDICES}(k) \equiv \{j(k), t(k)\};\\
&\text{STEP_SIZE}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{V}}, \beta, k) \equiv \max_{l=0,1,...,\infty} \alpha = (1/2)^l\\
&\qquad\qquad \text{s.t. } f_{j(k)}(\hat{\boldsymbol{x}} + \alpha\hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}||) - f_{j(k)}(\hat{\boldsymbol{x}}) \leq \beta\alpha\hat{\boldsymbol{g}}_{\boldsymbol{j(k)}}{}^T\hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}||;
\end{aligned} \tag{5.9}$$

where $j(k)$ and $t(k)$ are defined as $j(k) = 1$ and $t(0) = 2$ with $t(k+1) = t(k) + 1$ if $t(k) < m$ and $t(k+1) = 2$ otherwise. Index j holds the anchor function identity and t produces a step-wise sampling of the gradients.

Theorem 17. *Suppose all functions f_i are bounded from below and let $f^{\min}$ be a lower bound on all f_i. The central descent method with incremental anchored sampling with inexact searching as defined in (5.9) generates a sequence such that:*

$$\min_{0 \leq l \leq k-1} \frac{\min_{t=1,...,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||}{||\hat{\boldsymbol{V}}^l||} \leq \sqrt{\frac{1}{k}\left(\frac{f_1(\hat{\boldsymbol{x}}^0) - f^{\min}}{\min\left\{\frac{\beta(1-\beta)}{2L}, \beta\right\}}\right)}. \tag{5.10}$$

Proof. The proof will make use of the following lemma:

Lemma 4. *Assuming that the gradients of objective functions are L-Lipschitz continuous, then, in each iteration of Algorithm 5 the step-size α satisfies:*

$$\alpha \geq \alpha_{\min} \equiv \min\left\{\tfrac{1-\beta}{2L}, 1\right\} ||\hat{\boldsymbol{g}}_j||/||\hat{\boldsymbol{V}}||. \tag{5.11}$$

where $j = j(k)$ is the index of the function searched in STEP_SIZE($\hat{\boldsymbol{x}}, \hat{\boldsymbol{V}}, \beta, k$).

Proof. When 2α does not satisfy sufficient decrease condition we have

$$f_j(\hat{\boldsymbol{x}} + 2\alpha\hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}||) - f_j(\hat{\boldsymbol{x}}) > \beta 2\alpha \nabla f_j(\hat{\boldsymbol{x}})^T \hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}||.$$

From the Lipschitz condition we find:

$$f_j(\hat{\boldsymbol{x}} + 2\alpha\hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}||) - f_j(\hat{\boldsymbol{x}}) \leq 2\alpha \nabla f_j(\hat{\boldsymbol{x}})^T \hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}|| + \tfrac{L}{2}||2\alpha \frac{\hat{\boldsymbol{V}}}{||\hat{\boldsymbol{V}}||}||^2.$$

Therefore:

$$\begin{aligned} & 2\alpha(1-\beta)\nabla f_j(\hat{\boldsymbol{x}})^T \hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}|| + 2L\alpha^2 \geq 0; \\ \implies & -L\alpha \leq (1-\beta)\nabla f_j(\hat{\boldsymbol{x}})^T \hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}|| \leq -(1-\beta)||\nabla f_j(\hat{\boldsymbol{x}})||/||\hat{\boldsymbol{V}}||; \\ \implies & \alpha \geq \frac{1-\beta}{L}||\nabla f_j(\hat{\boldsymbol{x}})||/||\hat{\boldsymbol{V}}|| = \tfrac{1-\beta}{2L}||\hat{\boldsymbol{g}}_j||/||\hat{\boldsymbol{V}}||. \end{aligned}$$

□

In each iteration l the following inequality holds for the objective function indexed by $j = j(l)$

$$f_j(\hat{\boldsymbol{x}}^{l+1}) - f_j(\hat{\boldsymbol{x}}^l) \leq \beta\alpha^l \nabla f_j(\hat{\boldsymbol{x}}^l)^T \hat{\boldsymbol{V}}^l/||\hat{\boldsymbol{V}}^l|| \leq -\beta\alpha^l ||\nabla f_j(\hat{\boldsymbol{x}}^l)||/||\hat{\boldsymbol{V}}^l||$$

and therefore by the above stated lemma

$$f_j(\hat{\boldsymbol{x}}^l) - f_j(\hat{\boldsymbol{x}}^{l+1}) \geq \beta\alpha^l ||\nabla f_j(\hat{\boldsymbol{x}}^l)||/||\hat{\boldsymbol{V}}^l|| \geq \beta\alpha_{\min}||\nabla f_j(\hat{\boldsymbol{x}}^l)||/||\hat{\boldsymbol{V}}^l||$$

$= \beta \min\left\{\tfrac{1-\beta}{2L}, 1\right\} ||\nabla f_j(\hat{\boldsymbol{x}}^l)||^2/||\hat{\boldsymbol{V}}^l||^2 \geq \min\left\{\tfrac{\beta(1-\beta)}{2L}, \beta\right\} \min_{t=1,\ldots,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||^2/||\hat{\boldsymbol{V}}^l||^2.$

Therefore a decrease of

$$f_j(\hat{\boldsymbol{x}}^l) - f_j(\hat{\boldsymbol{x}}^{l+1}) \geq \min\left\{\frac{\beta(1-\beta)}{2L}, \beta\right\} \min_{t=1,\ldots,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||^2/||\hat{\boldsymbol{V}}^l||^2 \tag{5.12}$$

is attained in iteration l. By summing the terms in equation (5.12) for varying values of l between 0 and $k-1$ we find:

$$f_j(\hat{\boldsymbol{x}}^0) - f_j(\hat{\boldsymbol{x}}^{k-1}) \geq \min\left\{\frac{\beta(1-\beta)}{2L}, \beta\right\} \sum_{l=0}^{k-1} \min_{t=1,\ldots,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||^2/||\hat{\boldsymbol{V}}^l||^2.$$

And therefore

$$f_j(\hat{\boldsymbol{x}}^0) - f^{\min} \geq k \min\left\{\frac{\beta(1-\beta)}{2L}, \beta\right\} \min_{0\leq l\leq k-1} \min_{t=1,\ldots,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||^2/||\hat{\boldsymbol{V}}^l||^2$$

$$\implies \frac{1}{k}\left(\frac{f_1(\hat{\boldsymbol{x}}^0) - f^{\min}}{\min\left\{\frac{\beta(1-\beta)}{2L}, \beta\right\}}\right) \geq \min_{0\leq l\leq k-1} \min_{t=1,\ldots,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||^2/||\hat{\boldsymbol{V}}^l||^2.$$

Which completes our proof. □

Theorem 17 provides, for the first time, an $O(1/\sqrt{k})$ convergence to critical conditions with an iteration cost that is unaffected by increasing values of m. The method delineated makes use of two gradients per iteration and one single-objective inexact line search. The above instantiation of the incremental central descent method is perhaps the simplest form in which Theorem 17 ensures convergence of $\min_{t=1,\ldots,m} ||\nabla f_t(\hat{\boldsymbol{x}}^l)||/||\hat{\boldsymbol{V}}^l||$ at the $1/\sqrt{k}$ rate of single-objective optimization. In each step an Armijo line-search is performed on one anchor function indexed by $j = j(k)$ which ensures that the sequence of estimates $\hat{\boldsymbol{x}}$ monotonically approaches critical conditions.

An intuitive step-by-step reasoning behind the proposed method is described below with the number between square brackets representing the iteration count:

- In the first iteration, only the gradients $\hat{\boldsymbol{g}}_1[1] = \nabla f_1(\hat{\boldsymbol{x}}[0])$ and $\hat{\boldsymbol{g}}_2[1] = \nabla f_2(\hat{\boldsymbol{x}}[0])$ are evaluated and employed in the computation of the direction $\hat{\boldsymbol{V}}[1]$; thus, it is guaranteed to be a descent direction with respect to the first two objectives, but not the remaining objectives. Then, the Armijo search generates a new point $\hat{\boldsymbol{x}}[1]$ that satisfies $f_1(\hat{\boldsymbol{x}}[1]) < f_1(\hat{\boldsymbol{x}}[0])$; and, if the step-size is sufficiently small, then also $f_2(\hat{\boldsymbol{x}}[1]) < f_2(\hat{\boldsymbol{x}}[0])$.
- In the second iteration, the gradients $\hat{\boldsymbol{g}}_1[2] = \nabla f_1(\hat{\boldsymbol{x}}[1])$ and $\hat{\boldsymbol{g}}_3[2] = \nabla f_3(\hat{\boldsymbol{x}}[1])$ are evaluated in the current point $\hat{\boldsymbol{x}}[1]$, and the former gradient of f_2 is kept, with $\hat{\boldsymbol{g}}_2[2] = \hat{\boldsymbol{g}}_2[1] = \nabla f_1(\hat{\boldsymbol{x}}[0])$. Those three vectors $\hat{\boldsymbol{g}}_1[2]$, $\hat{\boldsymbol{g}}_2[2]$ and $\hat{\boldsymbol{g}}_3[2]$ are used in the computation of a new direction $\hat{\boldsymbol{V}}[2]$. Notice that it is guaranteed that this $\hat{\boldsymbol{V}}[2]$ constitutes a descent direction for functions f_1 and f_3; and, if the step-size in the first iteration was sufficiently small, then due to the Lipschitz continuity of the gradients, it should also be a descent direction with respect to f_2, though it need not be (since we do not impose a small step-size). Thus, after the Armijo search in this iteration, the new point $\hat{\boldsymbol{x}}[2]$ must be such that $f_1(\hat{\boldsymbol{x}}[2]) < f_1(\hat{\boldsymbol{x}}[1])$ and, again, if the step-sizes were sufficiently small then $f_3(\hat{\boldsymbol{x}}[2]) < f_3(\hat{\boldsymbol{x}}[1])$ and possibly $f_2(\hat{\boldsymbol{x}}[2]) < f_2(\hat{\boldsymbol{x}}[1])$.
- In the third iteration, the gradients $\hat{\boldsymbol{g}}_1[3] = \nabla f_1(\hat{\boldsymbol{x}}[2])$ and $\hat{\boldsymbol{g}}_4[3] = \nabla f_4(\hat{\boldsymbol{x}}[2])$ are evaluated in the current point $\hat{\boldsymbol{x}}[2]$, and the old evaluations of gradients $\hat{\boldsymbol{g}}_2[3] = \hat{\boldsymbol{g}}_2[2] = \nabla f_1(\hat{\boldsymbol{x}}[0])$ and $\hat{\boldsymbol{g}}_3[3] = \hat{\boldsymbol{g}}_3[2] = \nabla f_3(\hat{\boldsymbol{x}}[1])$ are kept fixed. Those four vectors $\hat{\boldsymbol{g}}_1[3]$, $\hat{\boldsymbol{g}}_2[3]$, $\hat{\boldsymbol{g}}_3[3]$ and $\hat{\boldsymbol{g}}_4[3]$ are used in the computation of a new direction $\hat{\boldsymbol{V}}[3]$. Notice that it is guaranteed that $\hat{\boldsymbol{V}}[3]$ now constitutes a descent direction for functions f_1 and f_4. Again, if the previous step-sizes were sufficiently small then by Lipschitz continuity of the gradients $\hat{\boldsymbol{V}}[3]$ should also be a descent direction with respect to f_2 and f_3, however it need not be. And, after the Armijo step, the new point $\hat{\boldsymbol{x}}[3]$ must be such that $f_1(\hat{\boldsymbol{x}}[3]) < f_1(\hat{\boldsymbol{x}}[2])$, and, if the step-sizes are sufficiently small, we may also ensure $f_4(\hat{\boldsymbol{x}}[3]) < f_4(\hat{\boldsymbol{x}}[2])$, $f_2(\hat{\boldsymbol{x}}[3]) < f_2(\hat{\boldsymbol{x}}[2])$ and $f_3(\hat{\boldsymbol{x}}[3]) < f_3(\hat{\boldsymbol{x}}[2])$; though these last three inequalities need not hold.

  ...
- This process of increasing one new gradient vector on each iteration goes on up to iteration $k = m$. After this iteration, all computations of the descent direction $\hat{\boldsymbol{V}}[k]$ are performed using m non-null vectors $\hat{\boldsymbol{g}}_1, \ldots, \hat{\boldsymbol{g}}_m$. Notice that on each iteration k the vector $\hat{\boldsymbol{g}}_1$ is updated to the current value of the gradient $\nabla f_1(\hat{\boldsymbol{x}}[k-1])$ and one other vector $\hat{\boldsymbol{g}}_t$ is also updated to $\nabla f_t(\hat{\boldsymbol{x}}[k-1])$, which means that each vector $\hat{\boldsymbol{g}}_i$ for $i = 2, \ldots, m$ is updated at least once every $m-1$ iterations. Armijo Searching ensures that $f_1(\hat{\boldsymbol{x}}[k]) < f_1(\hat{\boldsymbol{x}}[k-1])$ and

$f_t(\hat{\boldsymbol{x}}[k]) < f_t(\hat{\boldsymbol{x}}[k-1])$ and, if the step-sizes are sufficiently small, we ensure $f_i(\hat{\boldsymbol{x}}[k]) < f_i(\hat{\boldsymbol{x}}[k-1])$ for every $i = 1, ..., m$.

It should be noticed that any function f_i might be used instead of f_1 for the purpose of controlling the step size with Armijo line-searching. Of course, different choices of the anchor function will lead to different step sizes, causing different trajectories of the sequence $\hat{\boldsymbol{x}}[k]$ towards the set of Pareto-critical points (different choices for anchor functions can be used and are discussed ahead). In this version of Algorithm 5, although the only function which is guaranteed to decrease on each step is the anchor function f_1, since the step sizes tend to decrease from iteration to iteration, every other non-anchor function also tends to decrease the further into the run. This is because decreasing step-sizes combined with Lipschitz continuity of the gradients ensures that the difference between the estimates $\hat{\boldsymbol{g}}_i$ and the gradients $\nabla f_i(\hat{\boldsymbol{x}})$ also tends to decrease from iteration to iteration. Furthermore, the robustness guarantees of $\boldsymbol{V}_c/||\boldsymbol{V}_c||$ in Theorem 13 provides maximal resilience to estimation errors, and thus, it is reasonable to expect $\hat{\boldsymbol{V}}/||\hat{\boldsymbol{V}}||$ to quickly fall within the cone of descent directions despite the added errors that come with the incremental approach in updating $\hat{\boldsymbol{g}}_i$'s. However, it is also worth noticing that the convergence to Pareto-critical conditions does not require this, i.e. that all searches be performed along true descent directions; instead, as demonstrated in our main result, monotonic decrease of the (lower-bounded) anchor function is sufficient.

5.3.1 Even weaker conditions for convergence

In the light of Theorem 17 one may ask what are the minimal conditions to ensure the convergence of Algorithm 5. In the following we provide yet another convergence guarantee for Algorithm 5, where the assumption of lower-boundedness is dropped. Furthermore, the anchoring strategy is dropped and the step-sizes $\{\alpha_k\}_{k=1,2,...}$ are only assumed to be non-negative, vanishing and non-summable; i.e. rather than assuming a specific backtracking strategy, we make use of an assumption that can be ensured by pre-specifying the step-sizes (e.g. $\alpha_k = 1/k$) or, for example, by performing a search in vanishing intervals. More precisely, we assume that

Vanishing non-summable steps with uniform sampling:
The functions STEP_SIZE and SELECT_INDICES satisfy

$$\begin{aligned} &\text{STEP\_SIZE produces a sequence of } \alpha_k > 0 \text{ for all } k \text{ and that } \lim_{k\to\infty} \alpha_k = 0 \\ &\qquad \text{and } \textstyle\sum_{k\in\mathbb{N}} \alpha_k = \infty; \\ &\text{SELECT\_INDICES}(k) \equiv \{t(k)\}; \end{aligned} \tag{5.13}$$

where $t(k)$ is defined as $t(1) = 1$ with $t(k+1) = t(k)+1$ if $t(k) < m$ and $t(k+1) = 1$ otherwise.

Theorem 18. *Algorithm 5 constructed with (5.13) produces a subsequence of estimates $\{\hat{\boldsymbol{x}}_{k_j}\}_{j=1,2,...}$ such that as $j \to \infty$ either (i) the gradient $\nabla f_i(\hat{\boldsymbol{x}}_{k_j})$ vanishes for some $i = 1, ..., m$; or (ii) the the central descent direction $\boldsymbol{V}_c(\hat{\boldsymbol{x}}_{k_j})$ is unbounded; or (iii) all functions are unbounded from below and decrease indefinitely:*

$$\begin{array}{lll} (i)\ \lim_{j\to\infty} ||\nabla f_i(\hat{\boldsymbol{x}}_{k_j})|| = 0 & (ii)\ ||\boldsymbol{V}_c(\hat{\boldsymbol{x}}_{k_j})|| \to \infty & (iii)\ \lim_{j\to\infty} f_i(\hat{\boldsymbol{x}}_{k_j}) = -\infty. \\ \textit{for some } i = 1,...,m & \textit{or} & \textit{for all } i = 1,...,m \end{array} \tag{5.14}$$

Before we provide the proof of Theorem 18 we point out a few key differences between this result and the one in Theorem 17. Firstly, it makes fewer and weaker assumptions on both the objective functions as well as the construction of Algorithm 5. As a consequence, the guarantees are also weaker in the sense that no speed or rate of decay/convergence is given. However, in one specific sense Theorem 18 provides one additional insurance that is not given by Theorem 17 alone: namely that the result is stated with respect to $||\boldsymbol{V}_c(\cdot)||$ and not simply with respect to $||\hat{\boldsymbol{V}}_c(\cdot)||$. Thus, Theorem 18 provides one (amongst others that are discussed in the following subsection) way to ensure a finer control over the convergence of $1/||\boldsymbol{V}_c(\cdot))||$ rather than $1/||\hat{\boldsymbol{V}}_c(\cdot)||$ if needed.

Proof. In this proof we will show that if (i) does not occur, then either (ii) or (iii) must occur. The negation of (i) implies that $||\nabla f_i(\hat{\boldsymbol{x}}_k)||$ is lower-bounded by some positive real value c_1 for all $i = 1, ..., m$ and all $k \in \mathbb{N}$ is thus assumed henceforth.

Notice that for any iteration $k \geq m + 1$ and any $i = 1, ..., m$ we have that $||\nabla f_i(\hat{\boldsymbol{x}}_k) - \hat{\boldsymbol{g}}_{i,k}|| = ||\nabla f_i(\hat{\boldsymbol{x}}_k) - \nabla f_i(\hat{\boldsymbol{x}}_\tau)|| \leq L||\hat{\boldsymbol{x}}_k - \hat{\boldsymbol{x}}_\tau||$ for some τ between $k - m$ and k; and furthermore $||\hat{\boldsymbol{x}}_k - \hat{\boldsymbol{x}}_\tau|| \leq ||\hat{\boldsymbol{x}}_k - \hat{\boldsymbol{x}}_{k-1}|| + ||\hat{\boldsymbol{x}}_{k-1} - \hat{\boldsymbol{x}}_{k-2}|| + ... + ||\hat{\boldsymbol{x}}_{k-m+1} - \hat{\boldsymbol{x}}_{k-m}|| = \sum_{k-m}^{k} \alpha_j$; and therefore

$$||\nabla f_i(\hat{\boldsymbol{x}}_k) - \hat{\boldsymbol{g}}_{i,k}|| \leq L \sum_{k-m}^{k} \alpha_j \quad \text{for any } k \geq m + 1 \text{ and } i = 1, ...m; \tag{5.15}$$

and thus $||\nabla f_i(\hat{\boldsymbol{x}}_k) - \hat{\boldsymbol{g}}_{i,k}||$ goes to zero as k increases. With this fact established we will now analyse two complementary cases:
Case 1. There exists a subsequence $\{k_j\}_{j=1,...,\infty}$ for which $\lim_{j\to\infty} ||\hat{\boldsymbol{V}}_{k_j}|| = \infty$.
Case 2. There exists an upper-bound $c_2 > 0$ for which $||\hat{\boldsymbol{V}}_k|| \leq c_2$ for every $k \in \mathbb{N}$.

Analysis of Case 1.

Under the conditions of Case 1 all that is needed is to show that when $||\hat{\boldsymbol{V}}_{k_j}|| \to \infty$ for $j \to \infty$, then the points $\hat{\boldsymbol{x}}_k$ produce a subsequence of central descent directions $\boldsymbol{V}_k \equiv \text{argmin}\{||\boldsymbol{V}||^2 \text{ st. } f_i(\hat{\boldsymbol{x}}_{\boldsymbol{k}})^T V \leq -||\nabla f_i(\hat{\boldsymbol{x}}_k)||\}$ that diverge. For this, notice that over the subsequence in which $||\hat{\boldsymbol{V}}_{k_j}||$ diverges we have: (A) the vectors $\hat{\boldsymbol{g}}_{i,k}/||\hat{\boldsymbol{g}}_{i,k}||$ are contained in a unit ball, and thus, there must exist a converging subsequence where $\lim_{k\to\infty} \hat{\boldsymbol{g}}_{i,k}/||\hat{\boldsymbol{g}}_{i,k}|| = \boldsymbol{u}_i$; and (B) $||\nabla f_i(\hat{\boldsymbol{x}}_k)||$ is lower-bounded by some positive real value c_1 for all $i = 1, ..., m$ and all $k \in \mathbb{N}$; and (C) the value of $||\nabla f_i(\hat{\boldsymbol{x}}_k) - \hat{\boldsymbol{g}}_{i,k}||$ goes to zero as k increases. The combination of (A), (B) and (C) imply that $\nabla f_i(\hat{\boldsymbol{x}}_k)/||\nabla f_i(\hat{\boldsymbol{x}}_k)||$ also converges to $\boldsymbol{u}_i$, and thus, by Lemma A.1 part 1 we conclude that the points $\hat{\boldsymbol{x}}_k$ produce a subsequence of central descent directions $\boldsymbol{V}_k \equiv \text{argmin}\{||\boldsymbol{V}||^2 \text{ st. } f_i(\hat{\boldsymbol{x}}_k)^T V \leq -||\nabla f_i(\hat{\boldsymbol{x}}_k)||\}$ that diverge.

Analysis of Case 2.

Lipschitz continuity of the gradients ensures that for each i = 1,...,m we have

$$f_i(\hat{\boldsymbol{x}}_{k+1}) - f_i(\hat{\boldsymbol{x}}_k) \leq \nabla f_i(\hat{\boldsymbol{x}}_k)^T(\hat{\boldsymbol{x}}_{k+1} - \hat{\boldsymbol{x}}_k) + \tfrac{1}{2}L||\hat{\boldsymbol{x}}_{k+1} - \hat{\boldsymbol{x}}_k||^2; \tag{5.16}$$

and thus

$$f_i(\hat{\boldsymbol{x}}_{k+1}) - f_i(\hat{\boldsymbol{x}}_k) \leq \alpha_k \nabla f_i(\hat{\boldsymbol{x}}_k)^T \hat{\boldsymbol{V}}_k/||\hat{\boldsymbol{V}}_k|| + \tfrac{1}{2}L\alpha_k^2 = \alpha_k[\hat{\boldsymbol{g}}_{i,k} + (\nabla f_i(\hat{\boldsymbol{x}}_k) - \hat{\boldsymbol{g}}_{i,k})]^T \hat{\boldsymbol{V}}_k/||\hat{\boldsymbol{V}}_k|| + \tfrac{1}{2}L\alpha_k^2$$

$$\leq \alpha_k \hat{\boldsymbol{g}}_{i,k}^T \hat{\boldsymbol{V}}_k/||\hat{\boldsymbol{V}}_k|| + \alpha_k L \sum_{k-m}^{k} \alpha_j + \tfrac{1}{2}L\alpha_k^2 \leq -\alpha_k||\hat{\boldsymbol{g}}_{i,k}||/||\hat{\boldsymbol{V}}_k|| + \alpha_k L \sum_{k-m}^{k} \alpha_j + \tfrac{1}{2}L\alpha_k^2;$$

where the first inequality of the second line is a consequence of (5.15) and the second inequality is a consequence of the definition of $\hat{\boldsymbol{V}}_k$. Now, using (5.15) a second time on the first term we obtain $-||\hat{\boldsymbol{g}}_{i,k}|| \leq -||\nabla f_i(\hat{\boldsymbol{x}}_k)|| + L\sum_{k-m}^{k}\alpha_j$ and thus $-\alpha_k||\hat{\boldsymbol{g}}_{i,k}||/||\hat{\boldsymbol{V}}_k|| \leq -\alpha_k||\nabla f_i(\hat{\boldsymbol{x}}_k)||/||\hat{\boldsymbol{V}}_k|| + \alpha_k L(\sum_{k-m}^{k}\alpha_j)/||\hat{\boldsymbol{V}}_k||$, and since by construction $||\hat{\boldsymbol{V}}_k|| \geq 1$ (when it exists) and by assumption $||\nabla f_i(\hat{\boldsymbol{x}}_k)|| \geq c_1$, then we conclude that the first term is upper-bounded by $-\alpha_k c_1/||\hat{\boldsymbol{V}}_k|| + \alpha_k L\sum_{k-m}^{k}\alpha_j$. Hence for every $i = 1, ..., m$ and every $k \geq m+1$ we have

$$f_i(\hat{\boldsymbol{x}}_{k+1}) - f_i(\hat{\boldsymbol{x}}_k) \leq \alpha_k \left(2L\sum_{k-m}^{k}\alpha_j + \tfrac{1}{2}L\alpha_k - \frac{c_1}{||\hat{\boldsymbol{V}}_k||} \right). \tag{5.17}$$

Furthermore, under the conditions of Case 2, there exists a constant $c_2 > 0$ such that $||\hat{\boldsymbol{V}}_k|| \leq c_2$ and thus $-c_1/||\hat{\boldsymbol{V}}_k|| \leq -c_1/c_2$. Inserting this back into (5.17) we obtain

$$f_i(\hat{\boldsymbol{x}}_{k+1}) - f_i(\hat{\boldsymbol{x}}_k) \leq \alpha_k \left(2L\sum_{k-m}^{k}\alpha_j + \tfrac{1}{2}L\alpha_k - c_1/c_2 \right).$$

Now notice that the first term within the brackets vanishes with increasing values of k, and thus for sufficiently large k we have $2L\sum_{k-m}^{k}\alpha_j + \frac{1}{2}L\alpha_k \leq \frac{1}{2}c_1/c_2$. Hence, for all $i = 1, ..., m$ and for all $k \geq \bar{k}$, for some $\bar{k} \in \mathbb{N}$, we have

$$f_i(\hat{\boldsymbol{x}}_{k+1}) - f_i(\hat{\boldsymbol{x}}_k) \leq -\tfrac{1}{2}\alpha_k c_1/c_2. \tag{5.18}$$

Summing up the terms for $k \geq \bar{k}$ in equation (5.18) we obtain:

$$\left[\lim_{k\to\infty} f_i(\hat{\boldsymbol{x}}_k)\right] - f_i(\hat{\boldsymbol{x}}_{\bar{k}}) \leq -\tfrac{1}{2}\frac{c_1}{c_2}\sum_{k\geq\bar{k}}\alpha_k = -\infty.$$

In this case all functions are unbounded and the sequence produces a subsequence of points in which all functions are simultaneously decreased indefinitely. This concludes our proof. □

5.3.2 Alternative sampling and searching strategies

We chose to display our main result in Theorem 17 with the simplest anchoring technique which keeps $j = 1$ as well as with the simplest Armijo-type line-search technique. However, other sound alternatives which more generally have the form of

$$\text{SELECT\_INDICES}(k) \equiv \mathcal{J}(k) \cup \mathcal{T}(k)$$

also ensure a convergence of the type found in Theorem 17 and are discussed in this subsection. First, the fixed anchor $j(k) = 1$ can be substituted for a set of fixed anchors of size $O(1)$ while still preserving the same query complexity cost:

Fixed anchor(s). *Select at random a fixed set of indices $\mathcal{J} \subset \{1, ..., m\}$ of size $\#\{\mathcal{J}\} = O(1)$ prior to initialization and perform the inexact line search in each function $f_{j'}(\cdot)$ for $j' \in \mathcal{J}$.*

Alternatively, the anchor functions over which the line-search is performed need not be fixed. It can be shown that instead of fixing the index of the anchor, one

might define j as the index of the function with lowest value visited so far:

Lowest anchor(s). *Function SELECT_INDICES*$(\cdot)$ *verifies if* $f_t(\boldsymbol{x}) < f_j(\boldsymbol{x})$ *and, if so, then it assigns* $j \leftarrow t$. *If multiple anchors are used, then, SELECT_INDICES*$(\cdot)$ *verifies and keeps the indices of functions with the lowest function values.*

Under such a rule, the inexact line search needs only to be performed on the lowest function observed. Similarly, the sampling strategy analyzed in Theorem 17 to estimate $\boldsymbol{V}_c$ had $t(k)$ incrementally visiting each index of the multi-objective function $\mathcal{F}(\boldsymbol{x}) \equiv [f_1(\boldsymbol{x}), ..., f_m(\boldsymbol{x})]^T$. We chose this rule due to its simplicity, however, if a finer monitoring of $1/||\boldsymbol{V}_c(\boldsymbol{x})||$ is desired rather than the estimate $1/||\hat{\boldsymbol{V}}^l||$, then we point out two alternatives. The first alternative is to update all $\hat{g}_i \leftarrow \nabla f_i(\hat{\boldsymbol{x}})$ for $i = 1, ..., m$ once in every m steps:

Full updating. *If* $k = m, 2m, 3m, ...$ *then* $\mathcal{T}(k) = \{1, ..., m\}$ *otherwise* $\mathcal{T}(k) = \{t(k)\}$.

And the second alternative is to reduce the step-sizes by means of:

Small step-sizes. *Select a high value for* $\beta \in (0, 1)$.

The *full updating* strategy ensures that the exact value of $1/||\hat{\boldsymbol{V}}(\boldsymbol{x})||$ can be known after every m steps while still attaining an average computational cost per iteration of only three gradient computations and a line search. And *small step-sizes* cautiously ensures small estimation errors of the older estimates $\hat{g}_i$ of $\nabla f_i(\hat{\boldsymbol{x}})$, which also amounts to finer estimation of $\boldsymbol{V}_c$. This causes an increase in the cost of the Armijo-type line-searching steps since $\sim \log \epsilon$ iterations are required in each call of the LINE_SEARCH$(\cdot)$ procedure, where $\epsilon = \epsilon(\beta)$ is a lower-bound on α^*. However this increased cost can be mitigated with the implementation of modern sub-logarithmic techniques to tackle the sufficient decrease condition that require only $\sim \log\log\log \epsilon$ iterations asymptotically while taking no more than $\sim \log\log \epsilon$ in the worst case: see *geometric bisection method* and the implementation of the *ITP method* on the logarithmic scale as discussed in (Oliveira & Takahashi, 2020, 2021a).

Finally, we point out that there are situations in which a smaller sampling might be desired. For example, when the number of objectives m is less than the dimension of the decision space n the (seldom updated) estimates $\hat{g}_i$ of the gradients are unlikely to produce an estimate $\hat{\boldsymbol{V}}(\boldsymbol{x})$ with a large value of $||\hat{\boldsymbol{V}}(\boldsymbol{x})||$ because that would require a near co-planarity of a subset of $\hat{g}_i$. However, when the number of objectives m is greater than the dimension of the decision space n, then it is sufficient that the complementarity condition $\sum_{i=1}^{m} \lambda_i \hat{g}_i = 0$ holds for $\lambda_i > 0$, which might occur with non-null probability. Thus, in such situations a smaller sampling might be desired to avoid early stopping. This can be addressed by either dropping older gradient estimates or by making $t(k)$ in (5.9) only visit a subset of indices $\{1, ..., m\}$ of size less than n.

5.4 A toy experiment

In this section we perform one small experiment to visualize Algorithm 5. For this we construct a multi-objective function which consists of five quadratic functions defined on the plane:

$$F(x,y) \equiv \begin{pmatrix} f_1(x,y) \\ f_2(x,y) \\ f_3(x,y) \\ f_4(x,y) \\ f_5(x,y) \end{pmatrix} \equiv \begin{pmatrix} (x+2)^2 + 3y^2 \\ 3x^2 + (y+2)^2 \\ (x-5)^2 + 3y^2 \\ 7(x-4)^2 + (y+\frac{1}{2})^2 \\ x^2 + (y-\frac{1}{2})^2 \end{pmatrix}. \tag{5.19}$$

These five functions have the following distinct minima : $P_1 \equiv (x_1^*, y_1^*) = (-2, 0)$, $P_2 \equiv (x_2^*, y_2^*) = (0, -2)$, $P_3 \equiv (x_3^*, y_3^*) = (5, 0)$, $P_4 \equiv (x_4^*, y_4^*) = (4, -\frac{1}{2})$, $P_5 \equiv (x_5^*, y_5^*) = (0, \frac{1}{2})$. And, the Pareto optimal region, depicted in Figure 5.6, has a non-linear optimal front composed of curves between points $\widehat{P_1P_5}$, $\widehat{P_5P_3}$, $\widehat{P_3P_4}$, $\widehat{P_4P_2}$ followed by $\widehat{P_2P_1}$. The first, third and fifth curves produce concave frontiers, the fourth curve produces a convex frontier and the second curve produces a frontier that is neither convex nor concave.

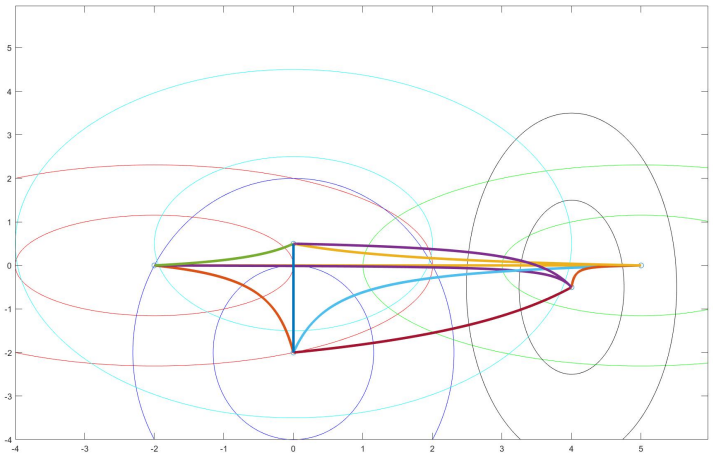
In this experiment we produced ten initial points equidistant from the origin with a radius of $r = 10$. Then, by performing ten steps of Algorithm 5 with the anchored sampling and Armijo line searching strategies outlined in (5.9) as in Theorem 17, we produced updated estimates. We depict the paths (in red) taken by the different starting points in Figure 5.7 for $\beta = 0.95$ in the final steps.

In this example we find that the path taken by the updated estimates (in red) closely approximates the stream lines of $\boldsymbol{V}_c(\boldsymbol{x})$ (in blue). This is so because we took β near 1. For smaller value of β (such as $\beta = 0.5$ not depicted in the figure) although our experiments suggest a faster convergence, the estimates take a less restrictive approximation path to the solution set. Also, we point out that this example illustrates that the method in Algorithm 5 with anchored sampling and Armijo searching (5.9) seems to be indiscriminate with respect to convexity/concavity of the Pareto-front; i.e. we observe convergence to points in both the convex and the concave fronts as well as the edges of the efficient set.

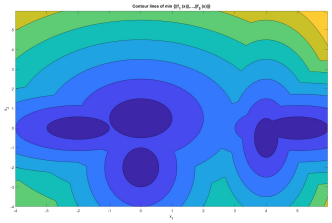
One caveat must however be pointed out. Since the anchored sampling of (5.9) only approximately estimates to $\boldsymbol{V}_c$ in each step, it is perfectly possible that the Algorithm has an early stop due to a "false alarm" with a small value of $1/||\hat{\boldsymbol{V}}_c||$ when the actual value of $1/||\boldsymbol{V}_c||$ is not small. However, isolating the effects of the stopping criteria the method will behave as expected; in our example this is ensured by taking large value of β since the value of $1/||\hat{\boldsymbol{V}}_c||$ is thus ensured to be close to the actual value of $1/||\hat{\boldsymbol{V}}_c||$ for large β.

5.5 An improvement on single objective optimization?

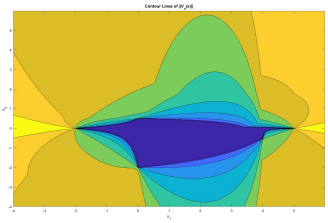
While the focus of this chapter has been on producing an improved steepest-descent type method for multi-objective optimization, one of the chapter's starting point was the fact that critical conditions for multi-objective optimization can be seen as a relaxation to critical conditions with respect to single objective optimization. And, this implies that the computational cost of searching for critical points of



(a) Efficient fronts & level curves.



(b) $\min_{j=1,\ldots,m} ||\nabla f_j(\boldsymbol{x})||$



(c) $1/||\boldsymbol{V}_c(\boldsymbol{x})||$

Fig. 5.6: The top-most figure depicts the level curves and the efficient fronts for each pair of objectives. The bottom-left figure depicts the level curves of the minimum of the norm of the gradients. And, the figure to the bottom-right depicts the multi-objective metric induced by $||\boldsymbol{V}_c(\boldsymbol{x})||$ on the plain.

multi-objective problems must be less than or equal to the cost of searching for critical points of single-objective optimization problems. One compelling question that remains is: by how much is the computational cost reduced with the addition of objectives?

Here we partially answer this question by quantifying the reduction in worst case query complexity of one dimensional multi-objective optimization when compared to single-objective optimization. To the best of our knowledge, that one dimensional line-searching in multi-objective optimization can enjoy of a reduced computational cost was first identified in (Vieira et al., 2012) where a multi-objective golden-section procedure was proposed and argued to (i) find iterations where the shrinking constant of the bracketing interval was reduced with the tracking of the many objectives and (ii) have a weaker stopping criteria when compared to single objective optimiza-

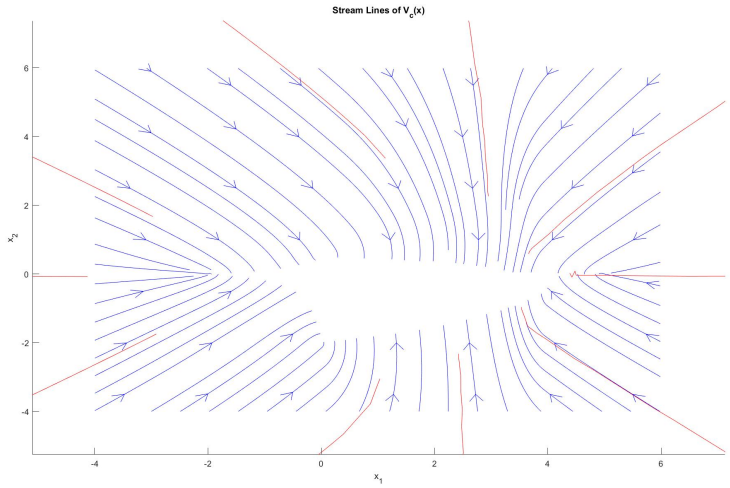


Fig. 5.7: The stream lines of $\boldsymbol{V}_c(\boldsymbol{x})$ (in blue) overlapped with the path (in red) taken by the ten iterations of Algorithm 5 with anchored sampling and Armijo line searching (5.9) for ten different starting points equidistant from the origin.

tion. In (Vieira et al., 2012) the minmax optimal $\sim \log 1/\epsilon$ lower-bound (where ϵ is the target precision) on query complexity for one dimensional searching is attained by the multi-objective line-search method proposed, and thus, the multi-objective problem complexity is shown to remain below the single-objective computational cost. However, as in the previous literature, in (Vieira et al., 2012) one query accounts for the computation of *all m objective functions* simultaneously, which, in our setting is equivalent to m queries. Here, we further argue that even with each objective function accounting for one query, the computational cost of a multi-objective line search must be no greater than that of a single-objective line search.

Problem definition.

Let $f_i : [0,1] \to \mathbb{R}$ for $i = 1, ..., m$ be m distinct unimodal functions with unique and distinct *minima* x_i for $i = 1, ..., m$; and, define $F : [0,1] \to \mathbb{R}^m$ as $F(x) = [f_1(x), f_2(x), ..., f_m(x)]$. Given a target precision ϵ, we are interested in finding an estimate $\hat{x} \in [0,1]$ such that $|\hat{x} - x^*| \leq \epsilon$ where x^* is critical with respect to $F(x)$. All functions considered are assumed to be continuously differentiable and the derivatives $f_i'(x)$ for $i = 1, ..., m$ are available. Thus a point x^* is critical if and only if there exists $i, j = 1, ..., m$ such that $f_i'(x^*) f_j'(x^*) \leq 0$. Furthermore, we make use of the following definitions:

$$\begin{gathered} \Delta x_* \equiv \max\{|x_i - x_j| \text{ st. } i, j = 1, ..., m\}, \\ \text{and} \\ \delta x_* \equiv \min\{|x_i - x_j| \text{ st. } i, j = 1, ..., m \text{ and } i \neq j\}. \end{gathered} \tag{5.20}$$

Remark 11. The assumption that the derivatives $f_i'(x)$ are available is made only to simplify the analysis while capturing the theoretical limits that can be derived for the non-differentiable case (and/or when function derivatives are not available). For example, derivatives used in our results may instead be substituted for numerical derivatives with a step-size smaller than ϵ and the result in Theorem 19 would hold, albeit, with weaker constants.

For the problem considered, the minmax optimal $\sim \log 1/\epsilon$ query complexity of one dimensional minimization is attained by Fibonacci-searching on functions $f : [a, b] \to \mathbb{R}$, and, if the derivative $f'(x)$ is available then the bisection method may be employed to search for the zero of $f'(x)$ instead at a slightly improved speed (Kiefer, 1953). In this section, we describe a one dimensional bisection-type method for multi-objective minimization that makes use of derivative values and produces a critical point with a query complexity of less than $\sim \log 1/\epsilon$. More specifically, we obtain a worst case query complexity of $\min\{3\lceil\log_2 1/\epsilon\rceil, 3\lceil\log_2 1/\delta x_*\rceil, 3\lceil\log_2 1/\Delta x_*\rceil + 3m\}$, where Δx_* is the maximum distance between the minima of each individual objective function, and, δx_* is the minimum distance between the minima of each individual objective function and m is the number of objective functions considered. Hence we obtain a less than $\sim \log 1/\epsilon$ minmax query complexity when the objective functions do not degenerate all minima to a neighbourhood of size less than ϵ and we match $3\lceil\log 1/\epsilon\rceil$ otherwise. Thus, we provide here for the first time an explicit characterization of the *reduction* in the worst-case query complexity of single-objective optimization obtained via multi-objective optimization. It is a subtle but meaningful result that may open doors to exploiting multi-objective relaxations to single-objective optimization problems. The analysis will rely on the following algorithm:

Algorithm 6: Alternating Bisection

Input: $F(x)$ and ϵ

/* where $F(x) = [f_1(x), ..., f_m(x)]$ with $m \geq 2$ is to be minimized and $\epsilon > 0$ is the target precision for the estimate $\hat{x}$ */

Initialize $a = 0, b = 1, k = 0$ and $i = 2$;

Calculate $l = f_1'(a), r = f_1'(b)$ and $\tilde{l} = f_i'(a), \tilde{r} = f_i'(b)$;

/* verifying if initial conditions already provide solution to minimization problem */

if $l \geq 0$ **then**

  $b = 0$;

else if $r \leq 0$ **then**

  $a = 1$;

/* if not, we then search for an ϵ solution */

while $(b - a > 2\epsilon)$ & $(l\tilde{l} > 0$ *and* $r\tilde{r} > 0)$ **do**

  /* storing previously visited points */

  $A_k = a, B_k = b$ and $k = k + 1$;

  /* performing a bisection step: */

  Calculate $x_{1/2} = \frac{a+b}{2}$ and $d = f_1'(x_{1/2})$;

  if $d < 0$ **then**

    $a = x_{1/2}$ and $l = d$;

  else if $d > 0$ **then**

    $b = x_{1/2}$ and $r = d$;

  else

    $a = x_{1/2}$ and $b = x_{1/2}$;

  /* rotating the objective function and updating information on the extremities: */

  if $i < m$ **then**

    $i = i + 1$;

  else

    $i = 2$;

  Calculate $\tilde{l} = f_i'(a)$ and $\tilde{r} = f_i'(b)$;

/* generating the estimate $\hat{x}$ */

if $l\tilde{l} \leq 0$ **then**

  $\hat{x} = a$;

else if $r\tilde{r} \leq 0$ **then**

  $\hat{x} = b$;

else

  Calculate $x_{1/2} = \frac{a+b}{2}$ and make $\hat{x} = x_{1/2}$;

Output: Estimate $\hat{x}$

The *alternating bisection method* makes use of a very similar technology as the *anchored incremental central descent method*. In each iteration it brackets the minima of one anchor function while incrementally querying the derivative of one other

function on the extremities of the bracketing interval. If the derivatives on the extremities have the same sign, then they are not critical with respect to the pair of objectives being evaluated. But, if the derivatives disagree in sign on at least one of the extremities of the interval then the method found a critical point with respect to that pair of objectives. In this way, the alternating bisection method enjoys the following guarantee:

Theorem 19. *The number of queries required by the alternating bisection method to return an estimate $\hat{x}$ to the multi-objective optimization problem is upper-bounded by*

$$3\lceil \log_2 1/\epsilon \rceil \quad \textit{and} \quad 3\lceil \log_2 1/\delta x_* \rceil \quad \textit{and} \quad 3\lceil \log_2 1/\Delta x_* \rceil + 3m. \tag{5.21}$$

Proof. In order to prove that the number of function evaluations is upper-bounded by $3\lceil \log_2 1/\epsilon \rceil$, we first recognize that for each function evaluation performed for the sake of the bisection step on the anchor function $f_1(x)$ in line 9, the Algorithm performs two evaluations of the derivative $f_i'(x)$ for some i in line 20. Also, notice that the while-loop contains the standard bisection stopping criteria of $(b - a > 2\epsilon)$ and thus, the upperbound $3\lceil \log_2 1/\epsilon \rceil$ is guaranteed by the standard bisection guarantees on the minimization of the anchor function f_1. It will terminate under this criteria if it has bracketed an epsilon solution to the single-objective problem, which in turn must also be an epsilon solution to the multi-objective problem as well.

We now proceed to prove that the number of function evaluations is upper-bounded by $3\lceil \log_2 1/\delta x_* \rceil$. For this, assume that Algorithm 6 has performed $\lceil \log_2 1/\delta x_* \rceil$ bisection steps and has produced an interval of length $b - a \leq \delta x_*$. We will see that in this iteration either the condition $l\tilde{l} \leq 0$ or the condition $r\tilde{r} \leq 0$ must hold enabling the algorithm to break loose from the while-loop. For this, we recall that by the definition of δx_* it follows that for any pair of objective functions i, j the distance between the minima x_i and x_j must be greater than or equal to δx_*; in particular this is true for the objectives $f_1(x)$ and the objective $f_i(x)$ in which that Algorithm 6 probes in line 20. Hence, interval $[x_1, x_i]$ (or $[x_i, x_1]$ depending on the order of x_i and x_1) is of length greater than or equal to that of $[a, b]$, and also, since Algorithm 6 has maintained x_1 bracketed in every iteration, it follows that $[a, b] \cap [x_1, x_i]$ is not empty (x_1 belongs to both intervals). Combining these two facts implies that either a or b or both must fall within interval $[x_1, x_i]$, i.e. at least one of them is a non-dominated solution to the problem of minimizing the pair of functions f_1 and f_i. Now, we recall that the first-order condition for a point $x^* \in (0, 1)$ to be non-dominated, given a pair of differentiable objective functions f_i and f_j, is that $f_i'(x^*)f_j'(x_*) \leq 0$; and, under the hypothesis of unimodality, this condition is also sufficient. Thus, we conclude that either $l\tilde{l} \leq 0$ or $r\tilde{r} \leq 0$ or both, which proves the second upper-bound.

Now, in order to prove our third upper-bound we will use the fact that $\max\{|x_1 - x_i| : \text{ for } i = 2, ..., m\}$ is greater than or equal to $\frac{1}{2}\Delta x_*$. Now, assume that Algorithm 6 has performed $\lceil \log_2 2/\Delta x_* \rceil$ iterations reducing the interval to a length of $b - a \leq \Delta x_*/2$. If the function f_i whose derivative is to be probed in line 20 happened to (luckily) be the one which maximizes $|x_1 - x_i|$, by analogous arguments to those provided above (for the upper bound of $3\lceil \log_2 1/\delta x_* \rceil$) Algorithm 6 would break loose from the while-loop for satisfying either $l\tilde{l} \leq 0$ or $r\tilde{r} \geq 0$ or both. Now, in any other case it takes at most m iterations for the index i that maximizes $|x_1 - x_i|$ to be probed after iteration number $\lceil \log_2 2/\Delta x_* \rceil$. Hence, at most $\lceil \log_2 2/\Delta x_* \rceil + m - 1 =$

$\lceil \log_2 1/\Delta x_* \rceil + m$ iterations (which correspond to $3\lceil \log_2 1/\Delta x_* \rceil + 3m$ derivative evaluations) are required to hit the desired index after the interval has reached the desired length.

□

Theorem 19 provides three upper-bounds on the cost of finding a non-dominated solution to the one-dimensional multi-objective problem. The first is a function of the desired precision ϵ, the second is a function of the geometry of the problem alone, and, the third is a function of the dimensions of the problem (in addition to the geometry of the problem). An immediate consequence of these upper-bounds is that, for small ϵ, the computational cost of finding a solution to the multi-objective problem turns out to be much less than that of finding a solution to the single-objective problem. Of course, if ϵ is not small then the single-objective problem might be less costly, however, even in such cases, the cost of the multi-objective problem is at most three times the cost of it's single-objective counterpart. Thus, for the first time, we can characterize (for the one dimensional problem) the reduction in computational cost associated with the addition of objective functions.

Finally, it is worth mentioning that, while we have focused here on the worst case bounds, improvements on asymptotic and average performances of Algorithm 6 can also be attained by performing *ITP steps* (Oliveira & Takahashi, 2020) in line 9 instead of bisection steps. This would yield an asymptotic function evaluation count of the order of $\min\{\log\log 1/\epsilon, \log\log 1/\delta x_*, \log\log 1/\Delta x_* + m\}$ given that x_1 is a simple root of $f_1'(x)$, and, that the ITP method is initiated near steady state conditions (as discussed in (Oliveira & Takahashi, 2020)), with no cost on the minmax performance established in Theorem 19.

5.6 Discussions

Current state-of-the-art solvers for multi-objective optimization problems require computing the gradients of all m objective functions per iteration and one m-objective line search to produce convergence to critical conditions at the worst case rate of $O(1/\sqrt{k})$, where k is the iteration count. Here, we propose an incremental descent method that achieves the same rate of $O(1/\sqrt{k})$ with at most two gradient computations per iteration and one single-objective line search; i.e. a reduction in the computational complexity by a factor of m.

The methods developed here make use of a brand new descent direction similar to the Cauchy's steepest descent, which we term the *central descent* direction. The central descent is shown to have improved robustness and geometric guarantees which are not shared by other directions considered so far in the literature. And, when approximated incrementally, it allows the construction of the method here proposed with a computational complexity that is unaffected by increasing values of m.

Future work.

The results attained here produce a convergence with a query complexity matching that of single-objective optimization. However, as mentioned in Section 5.1, since

multi-objective optimization can be seen as a relaxation to single-objective optimization, it is natural to expect that increasing values of m should reduce the overall computational cost of the search, specifically when the dependence on the stopping criteria is made explicit. This is so because a larger region of the solution space is considered near-critical with the increase in the number of objectives. In this chapter we focused solely on the iteration cost and, thus, it may be possible to show that, when the dependence on the stopping criteria is made explicit, the incremental central descent methods here considered have a diminishing cost with increasing values of m in a symilar fashion as the alternating bisection method of Section 5.5. Furthermore, despite our formulation having attained a reduced query complexity, it still requires a memory cost that is dependent and increasing with m. It might be possible to exploit the fact that each gradient only shows up as a restriction in the formulation of (5.2), and formulate an update scheme in the computation of $\hat{\boldsymbol{V}}$ that might mitigate and even eliminate the increasing memory cost with increasing values of m. Another compelling direction of research is to investigate if it is possible to exploit the "opposite direction" of scalarization; i.e. to formulate multi-objective relaxations of single-objective problems with the intent of reducing the overall query complexity of any given single-objective problem. We are unaware of any research done in this direction and we believe a mapping of the trade-offs associated with such a relaxation might prove to open brand new methods of solving classical problems.

5.A Appendix

5.A.1 Proof of Lemma 3

This proof will use make use of continuity-type results on the dependence of the central descent direction with respect to the gradients, described in Lemma 5 whose proof is found in the appendix Section 5.A.2.

Lemma 5. *Given a collection of vectors $\boldsymbol{g}_i$ for $i = 1, ..., m$ define $\mathcal{S} \equiv \{\boldsymbol{v}$ s.t. $\boldsymbol{g}_i^T \boldsymbol{v} \leq -||\boldsymbol{g}_i||$ for all i = 1, ...,m $\}$. Then:*

1. *If for some $R > 0$ the intersection of $\{\boldsymbol{v}$ s.t. $||\boldsymbol{v}|| \leq R\}$ with $\mathcal{S}$ is empty, then, there exists an $\epsilon > 0$ such that for every collection of $\hat{\boldsymbol{g}}_i$'s such that $||\hat{\boldsymbol{g}}_i - \boldsymbol{g}_i|| \leq \epsilon$ for all $i = 1, ..., m$ the intersection of $\{\boldsymbol{v}$ s.t. $||\boldsymbol{v}|| \leq R\}$ with $\{\boldsymbol{v} | \hat{\boldsymbol{g}}_i^T \boldsymbol{v} \leq -||\hat{\boldsymbol{g}}_i||$ for all i = 1, ...,m $\}$ is also empty.*
2. *If $\boldsymbol{v}_I$ is vector in $\mathbb{R}^n$ that satisfies $\boldsymbol{g}_i^T \boldsymbol{v}_I < -||\boldsymbol{g}_i||$ for all i = 1, ...,m , then, there exists an $\epsilon > 0$ such that for every collection of $\hat{\boldsymbol{g}}_i$'s in $\mathbb{R}^n$ such that $||\hat{\boldsymbol{g}}_i - \boldsymbol{g}_i|| \leq \epsilon$ for all $i = 1, ..., m$ the vector $\boldsymbol{v}_I$ also satisfies $\hat{\boldsymbol{g}}_i^T \boldsymbol{v}_I < -||\hat{\boldsymbol{g}}_i||$ for all i = 1, ...,m .*
3. *If $\boldsymbol{v}_E$ is a vector in $\mathbb{R}^n$ that satisfies $\boldsymbol{g}_i^T \boldsymbol{v}_E > -||\boldsymbol{g}_i||$ for some i = 1, ...,m , then, there exists an $\epsilon > 0$ such that for every collection of $\hat{\boldsymbol{g}}_i$'s in $\mathbb{R}^n$ such that $||\boldsymbol{g}_i - \hat{\boldsymbol{g}}_i|| \leq \epsilon$ for all $i = 1, ..., m$ the vector $\boldsymbol{v}_E$ also satisfies $\hat{\boldsymbol{g}}_i^T \boldsymbol{v}_E > -||\hat{\boldsymbol{g}}_i||$ for some i = 1, ...,m .*

Proof. Refer to the Appendix Section 5.A.2. □

Proof. Given that the gradients are L-Lipschitz continuous, we must have that $||\nabla f_i(\boldsymbol{x}_k) - \nabla f_i(\boldsymbol{x}_\infty)|| \to 0$ as $k \to \infty$. Let us assume that $\boldsymbol{x}_\infty$ is critical

with $\nabla f_i(\boldsymbol{x}_\infty) \neq 0$ for all i = 1, ..., m. Under these conditions the feasible set $\mathcal{S}_\infty \equiv \{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x}_\infty)^T \boldsymbol{v} \leq -||\nabla f_i(\boldsymbol{x}_\infty)|| \text{ for all i = 1, ..., m }\}$ is empty, and thus for any given $R > 0$ the intersection between $\{\boldsymbol{v} \text{ s.t. } ||\boldsymbol{v}|| \leq R\}$ and $\mathcal{S}_\infty$ is empty. Hence, as a consequence of Lemma 5 part 1 we have: for any $R > 0$ there will exist an $\epsilon > 0$ such that any collection of $\hat{\boldsymbol{g}}_i$'s in which $||\hat{\boldsymbol{g}}_i - \nabla f_i(\boldsymbol{x}_\infty)|| \leq \epsilon$ for all $i = 1, ..., m$ and any $\boldsymbol{v}$ which satisfies $||\boldsymbol{v}|| \leq R$ we have that the intersection between $\{\boldsymbol{v} \text{ s.t. } ||\boldsymbol{v}|| \leq R\}$ and $\{\boldsymbol{v} \text{ s.t. } \hat{\boldsymbol{g}}_i^T \boldsymbol{v} \leq -||\hat{\boldsymbol{g}}_i|| \text{ for all i = 1,...,m }\}$ is also empty. As a consequence of this fact, as $\nabla f_i(\boldsymbol{x}_k)$ approaches $\nabla f_i(\boldsymbol{x}_\infty)$, the value of $||\boldsymbol{V}_c(\boldsymbol{x}_k)||$ must arbitrarily increase.

Now, let us assume that $\boldsymbol{x}_\infty$ is non-critical, and hence, $\boldsymbol{V}_c(\boldsymbol{x}_\infty)$ is non empty. We recall the definition of the feasible set $\mathcal{S}_\infty$ as $\mathcal{S}_\infty \equiv \{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x}_\infty)^T \boldsymbol{v} \leq -||\nabla f_i(\boldsymbol{x}_\infty)|| \text{ for all i = 1, ..., m }\}$, and, we will refer to the *interior* of $\mathcal{S}_\infty$ as the set of vectors that satisfy all the inequalities strictly, i.e. the interior of $\mathcal{S}_\infty$ is the set $\{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x}_\infty)^T \boldsymbol{v} < -||\nabla f_i(\boldsymbol{x}_\infty)|| \text{ for all i = 1, ..., m }\}$. By considering the vector $\boldsymbol{v}_\delta \equiv (1 + \delta)\boldsymbol{V}_c(\boldsymbol{x}_\infty)$ for any $\delta > 0$ it is easy to see that $\boldsymbol{v}_\delta$ is in the interior of $\mathcal{S}_\infty$. As a consequence of Lemma 5 part 2, for every $\boldsymbol{v}_I$ in the interior of $\mathcal{S}_\infty$ there exists an $\epsilon > 0$ such that for every collection of $\hat{\boldsymbol{g}}_i$'s for $i = 1, ..., m$ that satisfy $||\hat{\boldsymbol{g}}_i - \nabla f_i(\boldsymbol{x}_\infty)|| \leq \epsilon$ we will have $\boldsymbol{v}_I$ also in the interior of $\{\boldsymbol{v} \text{ s.t. } \hat{\boldsymbol{g}}_i^T \boldsymbol{v} \leq -||\hat{\boldsymbol{g}}_i|| \text{ for all i = 1, ..., m }\}$. The same can be said about points in the exterior of $\mathcal{S}_\infty$ by applying Lemma 5 part 3, where $\boldsymbol{v}_E$ is said to be in the exterior of $\mathcal{S}_\infty$ if there exists an i between 1 and m such that $\nabla f_i(\boldsymbol{x}_\infty)^T \boldsymbol{v}_E > -||\nabla f_i(\boldsymbol{x}_\infty)||$. Hence, if any subsequence of $\boldsymbol{V}_k$'s converges, then, since it cannot converge to the interior of $\mathcal{S}_\infty$ nor the exterior, it must converge to the subset of $\mathcal{S}_\infty$ where at least one of the constraints $\nabla f_i(\boldsymbol{x}_\infty)^T \boldsymbol{v} \leq -||\nabla f_i(\boldsymbol{x}_\infty)||$ is satisfied with an equality.

Now, notice that every point in the interior of $\mathcal{S}_k \equiv \{\boldsymbol{v} \text{ s.t. } \nabla f_i(\boldsymbol{x}_k)^T \boldsymbol{v} < -||\nabla f_i(\boldsymbol{x}_k)|| \text{ for all i = 1, ..., m }\}$ provides an upper-bound on the value of $||\boldsymbol{V}_c(\boldsymbol{x}_k)||$ since $\boldsymbol{V}_c(\boldsymbol{x}_k)$ is the minimizer of $||\boldsymbol{v}||$ over $\mathcal{S}_k$. One such upper-bound is sufficient to recognize that $\boldsymbol{V}_c(\boldsymbol{x}_k)$ remains bounded for $k > \bar{k}$ for some $\bar{k} \in \mathbb{N}$. Thus, since bounded sequences must have at least one converging sub-sequence, all that is left is to show that every such sub-sequence converges to $\boldsymbol{V}_c(\boldsymbol{x}_\infty)$.

Since every point in the interior of $\mathcal{S}_k$ provides an upper-bound on the value of $||\boldsymbol{V}_c(\boldsymbol{x}_k)||$ we know that $\lim_{k\to\infty} ||\boldsymbol{V}_c(\boldsymbol{x}_k)|| \leq \lim_{\delta\to 0} ||\boldsymbol{v}_\delta|| = ||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$. Now, to see that $\lim_{k\to\infty} ||\boldsymbol{V}_c(\boldsymbol{x}_k)||$ also satisfies $\lim_{k\to\infty} ||\boldsymbol{V}_c(\boldsymbol{x}_k)|| \geq ||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$ notice that if $\{\boldsymbol{v} \text{ s.t. } ||\boldsymbol{v}|| \leq R\} \cap \mathcal{S}_\infty$ is empty for some $R > 0$, then, by Lemma 5 part 1 we have that for $\epsilon > 0$ sufficiently small, any collection of $\hat{\boldsymbol{g}}_i$'s that satisfy $||\hat{\boldsymbol{g}}_i - \nabla f_i(\boldsymbol{x}_\infty)|| \leq \epsilon$ will also have an empty intersection between $\{\boldsymbol{v} \text{ s.t. } ||\boldsymbol{v}|| \leq R\}$ and $\{\boldsymbol{v} \text{ s.t. } \hat{\boldsymbol{g}}_i^T \boldsymbol{v} \leq -||\hat{\boldsymbol{g}}_i \text{ for all i = 1, ..., m}||\}$. This is the case for any $R < ||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$, and thus for any $R < ||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$ and for any $k \geq \bar{k}$ for some sufficiently large $\bar{k} \in \mathbb{N}$ the value of $||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$ is greater than or equal to R. Taking the limit of R to $||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$ we obtain that $\lim_{k\to\infty} ||\boldsymbol{V}_c(\boldsymbol{x}_k)|| \geq ||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$.

Defining $\boldsymbol{v}_{\lim} \equiv \lim_{k\to\infty} \boldsymbol{V}_c(\boldsymbol{x}_k)$, since we have established that $||\boldsymbol{v}_{\lim}|| = ||\boldsymbol{V}_c(\boldsymbol{x}_\infty)||$ while satisfying $\nabla f_i(\boldsymbol{x}_\infty)^T \boldsymbol{v}_{\lim} \leq -||\nabla f_i(\boldsymbol{x}_\infty)||$ for all i = 1, ..., m (because it cannot be in the interior or the exterior of $\mathcal{S}_\infty$); and, since $\boldsymbol{V}_c(\boldsymbol{x}_\infty)$ is the unique minimizer of $||\boldsymbol{v}||$ over $\mathcal{S}_\infty$, we must conclude that $\lim_{k\to\infty} \boldsymbol{V}_c(\boldsymbol{x}_k) = \boldsymbol{V}_c(\boldsymbol{x}_\infty)$. □

5.A.2 Proof of Auxiliary Lemma 5

Proof. **Part 1.** Define $\epsilon(R)$ as any positive value strictly less than $\epsilon^*(R) \equiv \frac{z(R)+1}{R+1} \min_j\{||\boldsymbol{g}_j|| \text{ for j} = 1, ..., \text{m}\}$ where $z(R)$ is uniquely defined as:

$$z(R) \equiv \begin{cases} \min_{z\in\mathbb{R},\boldsymbol{v}\in\mathbb{R}^n} & z \\ \text{s.t.} & \boldsymbol{g}_i^T\boldsymbol{v} \leq z||\boldsymbol{g}_i|| \text{ for all i} = 1,...,\text{m}; \\ & ||\boldsymbol{v}|| \leq R. \end{cases} \tag{5.22}$$

Notice that since the intersection of $\{\boldsymbol{v} \text{ s.t. } ||\boldsymbol{v}|| \leq R\}$ with $\mathcal{S}$ is empty, it must be that $z(R)$ is strictly greater than -1 which implies that $\epsilon^*(R)$ is strictly greater than zero. Furthermore, notice that for any $\boldsymbol{v}$ which satisfies $||\boldsymbol{v}|| \leq R$ we will have that for some i between 1 and m the relation $\boldsymbol{g}_i^T\boldsymbol{v} \geq z(R)||\boldsymbol{g}_i||$ holds. This is because only for the minimizers $\boldsymbol{v}^*, z^*$ of (5.22) that $\boldsymbol{g}_i^T\boldsymbol{v}$ equates to $z(R)||\boldsymbol{g}_i||$ on (at least) one value of i between 1 and m. For non-optimal values of $\boldsymbol{v}$, there will exist an i where the condition $\boldsymbol{g}_i^T\boldsymbol{v} \leq z(R)||\boldsymbol{g}_i||$ must be broken. Thus, consider a collection of $\hat{\boldsymbol{g}}_i$'s in which $||\hat{\boldsymbol{g}}_i - \boldsymbol{g}_i|| \leq \epsilon(R)$ for all $i = 1, ..., m$ and any $\boldsymbol{v}$ which satisfies $||\boldsymbol{v}|| \leq R$; then, we have that for some i the following must hold:

$$\hat{\boldsymbol{g}}_i^T\boldsymbol{v} = \boldsymbol{g}_i^T\boldsymbol{v} + (\hat{\boldsymbol{g}}_i - \boldsymbol{g}_i)^T\boldsymbol{v} \geq z(R)||\boldsymbol{g}_i|| - \epsilon(R) \cdot R. \tag{5.23}$$

What we must show is that the right hands side of (5.23) is strictly greater than $-||\hat{\boldsymbol{g}}_i||$. This follows from

$$z(R)||\boldsymbol{g}_i|| - \epsilon(R) \cdot R > -||\hat{\boldsymbol{g}}_i|| \iff -\epsilon(R) \cdot R > -z(R)||\boldsymbol{g}_i|| - ||\hat{\boldsymbol{g}}_i||,$$

which holds if and only if

$$\tfrac{1}{R}\left[z(R)||\boldsymbol{g}_i|| + ||\hat{\boldsymbol{g}}_i||\right] > \epsilon(R). \tag{5.24}$$

Now observing that

$$\tfrac{1}{R}\left[z(R)||\boldsymbol{g}_i|| + ||\hat{\boldsymbol{g}}_i||\right] \geq \tfrac{1}{R}\left[z(R)||\boldsymbol{g}_i|| + ||\boldsymbol{g}_i|| - \epsilon(R)\right];$$

and, the right hand side is greater then $\epsilon(R)$ because

$$\begin{aligned} &\tfrac{1}{R}\left[z(R)||\boldsymbol{g}_i|| + ||\boldsymbol{g}_i|| - \epsilon(R)\right] > \epsilon(R) \\ \iff &\tfrac{1}{R}\left[z(R)||\boldsymbol{g}_i|| + ||\boldsymbol{g}_i||\right] > (1 + 1/R)\epsilon(R) \\ \iff &\tfrac{z(R)+1}{R+1}||\boldsymbol{g}_i|| > \epsilon(R); \end{aligned}$$

and, by the definition of $\epsilon(R)$, this inequality holds.

Part 2. If $\boldsymbol{v}_I$ satisfies $\boldsymbol{g}_i^T\boldsymbol{v}_I < -||\boldsymbol{g}_i||$ for all i = 1, ...,m , then clearly $s_i = \boldsymbol{g}_i^T\boldsymbol{v}_I + ||\boldsymbol{g}_i|| < 0$ for all $i = 1, ..., m$. Now, chose any $\epsilon > 0$ such that $\epsilon(1+||\boldsymbol{v}_I||) < -s_i$ for all $i = 1, ..., m$. Notice that

$$\hat{\boldsymbol{g}}_i^T\boldsymbol{v}_I = \boldsymbol{g}_i^T\boldsymbol{v}_I + (\hat{\boldsymbol{g}}_i - \boldsymbol{g}_i)^T\boldsymbol{v}_I = \boldsymbol{g}_i^T\boldsymbol{v}_I + ||\boldsymbol{g}_i|| - ||\boldsymbol{g}_i|| + (\hat{\boldsymbol{g}}_i - \boldsymbol{g}_i)^T\boldsymbol{v}_I;$$

and thus

$$\hat{\boldsymbol{g}}_i^T\boldsymbol{v}_I \leq s_i - ||\boldsymbol{g}_i|| + \epsilon||\boldsymbol{v}_I|| \leq s_i - ||\hat{\boldsymbol{g}}_i|| + \epsilon + \epsilon||\boldsymbol{v}_I|| = s_i - ||\hat{\boldsymbol{g}}_i|| + \epsilon(1 + ||\boldsymbol{v}_I||);$$

and, since ϵ was chosen to satisfy $\epsilon(1 + ||\boldsymbol{v}_I||) < -s_i$ for all $i = 1, ..., m$, we have that $\hat{\boldsymbol{g}}_i^T\boldsymbol{v}_I < -||\hat{\boldsymbol{g}}_i||$ for all $i = 1, ..., m$.

Part 3. If $\boldsymbol{v}_E$ satisfies $\boldsymbol{g}_i^T \boldsymbol{v}_E > -||\boldsymbol{g}_i||$ for some i = 1, ...,m , then clearly for one such $i = i^*$ that satisfies this inequality we have $s_{i^*} = \boldsymbol{g}_{i^*}^T \boldsymbol{v}_E + ||\boldsymbol{g}_{i^*}|| > 0$. Now, chose any $\epsilon > 0$ such that $\epsilon(1 + ||\boldsymbol{v}_E||) < s_{i^*}$. Notice that

$$\hat{\boldsymbol{g}}_{i^*}^T \boldsymbol{v}_E = \boldsymbol{g}_{i^*}^T \boldsymbol{v}_E + (\hat{\boldsymbol{g}}_{i^*} - \boldsymbol{g}_{i^*})^T \boldsymbol{v}_E = \boldsymbol{g}_{i^*}^T \boldsymbol{v}_E + ||\boldsymbol{g}_{i^*}|| - ||\boldsymbol{g}_{i^*}|| + (\hat{\boldsymbol{g}}_{i^*} - \boldsymbol{g}_{i^*})^T \boldsymbol{v}_E;$$

and thus

$$\hat{\boldsymbol{g}}_{i^*}^T \boldsymbol{v}_E \geq s_{i^*} - ||\boldsymbol{g}_{i^*}|| - \epsilon||\boldsymbol{v}_E|| \geq s_{i^*} - ||\hat{\boldsymbol{g}}_{i^*}|| - \epsilon - \epsilon||\boldsymbol{v}_E|| = s_{i^*} - ||\hat{\boldsymbol{g}}_{i^*}|| - \epsilon(1 + ||\boldsymbol{v}_E||);$$

and, since ϵ was chosen to satisfy $\epsilon(1 + ||\boldsymbol{v}_E||) < s_{i^*}$ we have that $\hat{\boldsymbol{g}}_{i^*}^T \boldsymbol{v}_E > -||\hat{\boldsymbol{g}}_{i^*}||$. □

References

Cartis, C., Gould, N. I. M., & Toint, P. L. (2010, September). On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, *20*(6), 2833-2852. doi: https://doi.org/10.1137/090774100

Cocchi, G., Liuzzi, G., Lucidi, S., & Sciandrone, M. (2020). On the convergence of steepest descent methods for multiobjective optimization. *Computational Optimization and Applications*, *77*, 1-27. doi: https://doi.org/10.1007/s10589-020-00192-0

Cortes, C., & Vapnik, V. (1995). Support-vector networks. In *Machine learning* (p. 273-297).

Cruz, J. Y. B., Pérez, L. R. L., & Melo, J. G. (2011). Convergence of the projected gradient method for quasiconvex multiobjective optimization. *Nonlinear Analysis: Theory, Methods and Applications*, *74*(16), 5268-5273. doi: https://doi.org/10.1016/j.na.2011.04.067

Drummond, L. M. G., & Iusem, A. N. (2004). A projected gradient method for vector optimization problems. *Computational Optimization and Applications*, *28*, 5-29. doi: https://doi.org/10.1023/B:COAP.0000018877.86161.8b

Fliege, J., Drummond, L. M. G., & Svaiter, B. (2009). Newton's method for multiobjective optimization. *Journal on Optimization*, *20*(2), 602-626. doi: https://doi.org/10.1137/08071692X

Fliege, J., & Svaiter, B. F. (2000). Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, *51*, 479-494. doi: https://doi.org/10.1007/s001860000043

Fliege, J., Vaz, A. I. F., & Vicente, L. N. (2019). Complexity of gradient descent for multiobjective optimization. *Optimization Methods and Software*, *34*(5), 949-959. doi: https://doi.org/10.1080/10556788.2018.1510928

Fukuda, E. H., & Drummond, L. M. G. (2014). A survey on multiobjective descent methods. *Pesquisa Operacional*, *34*(3), 585-620. doi: https://doi.org/10.1590/0101-7438.2014.034.03.0585

Gebken, B., & Peitz, S. (2021). An efficient descent method for locally lipschitz multiobjective optimization problems. *Journal of Optimization Theory and Applications*, *188*, 696-723. doi: https://doi.org/10.1007/s10957-020-01803-w

Kiefer, J. (1953). Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, *4*(3), 502-506.

Luenberger, D. G., & Ye, Y. (2018). Linear and nonlinear programming. In (4th ed.). Springer International Publishing.

Moudden, M. E., & Mouatasim, A. E. (2021). Accelerated diagonal steepest descent method for unconstrained multiobjective optimization. *Journal of Optimization Theory and Applications*, *188*, 220-242. doi: https://doi.org/10.1007/s10957-020-01785-9

Oliveira, I. F. D., & Takahashi, R. H. C. (2020). An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software*.

Oliveira, I. F. D., & Takahashi, R. H. C. (2021a). Efficient solvers for armijo's backtracking problem.
(Pre-print available at `https://arxiv.org/abs/2110.14072`)

Oliveira, I. F. D., & Takahashi, R. H. C. (2021b). An incremental descent method for multi-objective optimization. *Under consideration by the journal Optimization Methods and Software, Taylor & Francis*. (Pre-print available at `https://arxiv.org/abs/2105.11845`)

Pérez, L. R. L., & Prudente, L. F. (2019). A wolfe line search algorithm for vector optimization. *ACM Transactions on Mathematical Software*, *45*(4), 1-23. doi: https://doi.org/10.1145/3342104

Povalej, Z. (2014). Quasi-newton's method for multiobjective optimization. *Journal of Computational and Applied Mathematics*, *255*(1), 765-777. doi: https://doi.org/10.1016/j.cam.2013.06.045

Tanabe, H., Fukuda, E. H., & Yamashita, N. (2019). Proximal gradient methods for multiobjective optimization and their applications. *Computational Optimization and Applications*, *72*, 339–361. doi: https://doi.org/10.1007/s10589-018-0043-x

Vieira, D. A. G., Takahashi, R. H. C., & Saldanha, R. R. (2012). Multicriteria optimization with a multiobjective golden section line search. *Mathematical Programming*, *131*(1-2), 131–161. Retrieved from `https://doi.org/10.1007/s10107-010-0347-9` doi: 10.1007/s10107-010-0347-9

Zhou, X. (2018). On the fenchel duality between strong convexity and lipschitz continuous gradient. *arXiv Preprint arXiv:1803.06573*.

Concluding Remarks

Searching algorithms have served as building blocks for more complex computational systems, and, are often used as subroutines called multiple times by parent algorithms used in computer science, engineering and numerical analysis. This thesis revisits four such classical searching algorithms which are widely considered optimal with respect to the typical measures of performance. And, somewhat surprisingly, we find long overseen theoretical improvements to these procedures. The procedures and corresponding problems considered are: (i) the bisection method for continuous root-searching problems; (ii) the binary search algorithm for discrete list-searching; (iii) the back-tracking technique for inexact Armijo-type searching; and (iv) the n-dimensional steepest descent method for non-linear multi-objective optimization.

The four improvements proposed have, to a greater of lesser degree, one thing in common: we keep the worst case guarantees of our methods upper-bounded by the current state-of-the-art while ensuring improvements on alternative metrics such as average computational cost or asymptotic costs. Our theoretical improvements are well reflected in time savings and in reduction of computational costs as confirmed in many experiments and, at times, can span several orders of magnitude. These results are surprising precisely because the methods here considered have been both extensively studied in the literature as well as employed in multiple applications, and thus our advancements both push the theoretical boundaries and bring practical benefits to existing technology.

Much of the improvements come from a careful understanding of the optimality of binary searching strategies. More specifically, (i) the finding that binary searching is not uniquely minmax optimal motivated (ii) the search for other minmax optimal strategies with improved performance under different metrics; and also (iii) a better comprehension of the interdependence of the performance of binary searching with the underlying metrics pre-assumed on the domain of solutions clarified (iv) the improper use of binary searching in back-tracking procedures. The first two papers, found in Chapters 2 and 3, build on an alternative minmax optimal strategy which we name the ITP method, whereas, the two last papers, found in Chapters 4 and 5, build on this interplay between the metric used in the domain of solutions and the performance of the divide-and-conquer nature of searching strategies.

A few open problems

All four papers presented in this thesis point to different promising directions in which the techniques here identified may yet be improved upon. However, as a closing remark, we reiterate what we believe to be the most promising ideas that deserve further investigation. First, concerning one dimensional problems: The ITP method described and analysed in detail in this thesis still requires three user provided meta-parameters to be fully defined κ_1, κ_2 and n_0. Although we empirically adjust these parameters to the data available in our sample test-sets, it still remains an open question as to how to select these values in a theoretically sound and empirically satisfiable way. Second, concerning the multi-objective optimization scenario: Different classes of functions could have been considered in our derivation of the query complexity guarantees on objective functions with Lipschitz continuous gradients. Other natural classes of functions that deserve investigation include convex functions, strongly convex functions as well as classical (and widely used) quadratic forms. Third and lastly, concerning single-objective optimization: As reiterated multiple times in the second half of this thesis, multi-objective optimization can be seen as a relaxation to single-objective optimization, and, the addition of objectives to a single-objective optimization problem with the purpose of lowering computational cost is a much unexplored idea that deserves further investigation.

The following collected list of references contains the bibliography cited throughout the chapters of this thesis. It is presented in alphabetical order for ease of access.

References

Argyros, I. K., & Khattri, S. K. (2013). On the secant method. *Journal of Complexity*, *29*(6), 454-471.

Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, *16*(1), 1-3.

Bentley, J. L., & Sedgewick, R. (1997, January). Fast algorithms for sorting and searching strings. *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 360–369.

Bentley, J. L., & Yao, A. C. (1976). An almost optimal algorithm for unbounded searching. *Information Processing Letters*, *5*(3), 82-87. doi: https://doi.org/10.1016/0020-0190(76)90071-5

Boukis, C., Mandic, D. P., Constantinides, A. G., & Polymenakos, L. C. (2010, May). A modified armijo rule for the online selection of learning rate of the lms algorithm. *Digital Signal Processing*, *20*(3), 630–639. doi: https://doi.org/10.1016/j.dsp.2009.09.003

Boyd, S., & Vandenberghe, L. (2009). Convex optimization. In (7th ed.). Cambridge, UK: Cambridge University Press.

Brent, R. P. (1971). An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, *14*(4), 422–425. doi: https://doi.org/10.1093/comjnl/14.4.422

Burachik, R., Drummond, L. M. G., Iusem, A. N., & Svaiter, B. F. (1995, January). Full convergence of the steepest descent method with inexact line searches. *Optimization*, *32*, 137–146. doi: https://doi.org/10.1080/02331939508844042

Bus, J. C. P., & Dekker, T. J. (1975). Two efficient algorithms with guaranteed convergence for finding a zero of a function. *ACM Transactions on Mathematical Software*, *1*(4), 330–345. doi: https://doi.org/10.1145/355656.355659

Calatroni, L., & Chambolle, A. (2017, September). Backtracking strategies for accelerated descent methods with smooth composite objectives. *SIAM Journal on Optimization*, *29*, 1772–1798. doi: https://doi.org/10.1137/17M1149390

Cannizzo, F. (2018). Fast and vectorizable alternative to binary search in O(1) applicable to a wide domain of sorted arrays of floating point numbers. *Journal of Parallel and Distributed Computing*, *113*(5), 37. doi: https://doi.org/10.1016/j.jpdc.2017.10.007

Cartis, C., Gould, N. I. M., & Toint, P. L. (2010, September). On the complexity of steepest descent, newton's and regularized newton's methods for

nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, *20*(6), 2833-2852. doi: https://doi.org/10.1137/090774100

Chapra, S. C., & Canale, R. P. (2010). Numerical methods for engineers. In (6th ed. ed., p. 202-220). New York, NY: McGraw-Hill Higher Education.

Cocchi, G., Liuzzi, G., Lucidi, S., & Sciandrone, M. (2020). On the convergence of steepest descent methods for multiobjective optimization. *Computational Optimization and Applications*, *77*, 1-27. doi: https://doi.org/10.1007/s10589-020-00192-0

Conn, A. R., Gould, N. I. M., & Toint, P. L. (2000). *Trust region methods*. SIAM Society for Industrial and Applied Mathematics.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. In *Machine learning* (p. 273-297).

Cruz, J. Y. B., Pérez, L. R. L., & Melo, J. G. (2011). Convergence of the projected gradient method for quasiconvex multiobjective optimization. *Nonlinear Analysis: Theory, Methods and Applications*, *74*(16), 5268-5273. doi: https://doi.org/10.1016/j.na.2011.04.067

Dowell, M., & Jarratt, P. (1971, June). A modified regula falsi method for computing the root of an equation. *ACM Transactions on Mathematical Software*, *11*(2), 168-174. doi: https://doi.org/10.1007/BF01934364

Drummond, L. M. G., & Iusem, A. N. (2004). A projected gradient method for vector optimization problems. *Computational Optimization and Applications*, *28*, 5-29. doi: https://doi.org/10.1023/B:COAP.0000018877.86161.8b

Eiger, A., Sikorski, K., & Stenger, F. (1984). A bisection method for systems of nonlinear equations. *ACM Transactions on Mathematical Software*, *10*(4), 367-377. doi: https://doi.org/10.1145/2701.2705

Fletcher, R. (1987). *Practical methods of optimization* (Second ed.). New York, NY, USA: John Wiley & Sons.

Fliege, J., Drummond, L. M. G., & Svaiter, B. (2009). Newton's method for multiobjective optimization. *Journal on Optimization*, *20*(2), 602-626. doi: https://doi.org/10.1137/08071692X

Fliege, J., & Svaiter, B. F. (2000). Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, *51*, 479-494. doi: https://doi.org/10.1007/s001860000043

Fliege, J., Vaz, A. I. F., & Vicente, L. N. (2019). Complexity of gradient descent for multiobjective optimization. *Optimization Methods and Software*, *34*(5), 949-959. doi: https://doi.org/10.1080/10556788.2018.1510928

Ford, J. A. (1995). *Improved algorithms of illinois-type for the numerical solution of nonlinear equations* (Department of Computer Science Report). University of Essex.

Fukuda, E. H., & Drummond, L. M. G. (2014). A survey on multiobjective descent methods. *Pesquisa Operacional*, *34*(3), 585-620. doi: https://doi.org/10.1590/0101-7438.2014.034.03.0585

Gal, S., & Miranker, W. (1977). Optimal sequential and parallel search for finding a root. *Journal of Combinatorial Theory*, *23*(1), 1-14. doi: https://doi.org/10.1016/0097-3165(77)90074-7

Gebken, B., & Peitz, S. (2021). An efficient descent method for locally lipschitz multiobjective optimization problems. *Journal of Optimization Theory and Applications*, *188*, 696-723. doi: https://doi.org/10.1007/s10957-020-01803-w

Gill, P. E., Murray, W., & Wright, M. H. (1997). *Practical optimization* (11th ed.). Academic Press.

Graf, S., Novak, E., & Papageorgiou, A. (1989). Bisection is not optimal on the average. *Numerische Mathematik*, *55*, 481-491. doi: https://doi.org/10.1007/BF01396051

Kearfott, R. B. (1987). Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, *13*(3), 197-220. doi: https://doi.org/10.1145/29380.29862

Kiefer, J. (1953). Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, *4*(3), 502-506.

Knuth, D. E. (1998). The art of computer programming - sorting and searching. In (2nd ed., Vol. 3, chap. 6.2). Addison-Wesley.

Laber, E. S., Milidiú, R. L., & Pessoa, A. A. (2002). On binary searching with nonuniform costs. *SIAM Journal on Computing*, *31*(4), 1022-1047. doi: https://doi.org/10.1137/S0097539700381991

Le, D. (1985a). An efficient derivative-free method for solving nonlinear equations. *ACM Transactions on Mathematical Software*, *11*(3), 250-262. doi: https://doi.org/10.1145/214408.214416

Le, D. (1985b). Three new rapidly convergent algorithms for finding a zero of a function. *SIAM Journal on Scientific and Statistical Computing*, *6*(1), 193-208. doi: https://doi.org/10.1137/0906016

Luenberger, D. G., & Ye, Y. (2018). Linear and nonlinear programming. In (4th ed.). Springer International Publishing.

McNamee, J. M., & Pan, V. Y. (2012). Efficient polynomial root-refiners: A survey and new record efficiency estimates. *Computers & Mathematics with Applications*, *63*(1), 239-254. doi: https://doi.org/10.1016/j.camwa.2011.11.015

Moudden, M. E., & Mouatasim, A. E. (2021). Accelerated diagonal steepest descent method for unconstrained multiobjective optimization. *Journal of Optimization Theory and Applications*, *188*, 220-242. doi: https://doi.org/10.1007/s10957-020-01785-9

Muller, D. E. (1956). A method for solving algebraic equations using an automatic computer. *Mathematical Tables and Other Aids to Computation*, *10*(56), 208-215. doi: https://doi.org/10.2307/2001916

Nerinckx, D., & Haegemans, A. (1976). A comparison of non-linear equation solvers. *Journal of Computational and Applied Mathematics*, *2*(2), 145-148. doi: https://doi.org/10.1016/0771-050X(76)90017-6

Nesterov, Y. (2018). Lectures on convex optimization. In (2nd ed., Vol. 137). Springer International Publishing.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization* (second ed.). New York, NY, USA: Springer.

Norton, V. (1985). Algorithm 631 finding a bracketed zero by larkin's method of rational interpolation. *ACM Transactions on Mathematical Software*, *11*(2), 120-134. doi: https://doi.org/10.1145/214392.214396

Novak, E. (1989). Average-case results for zero finding. *Journal of Complexity*, *5*(4), 489-501.

Novak, E., & Ritter, K. (1993). Some complexity results for zero finding for univariate functions. *Journal of Complexity*, *9*(1), 15-40. doi: https://doi.org/10.1006/jcom.1993.1003

Novak, E., Ritter, K., & Woźniakowski, H. (1995). Average-case optimality of a hybrid secant-bisection method. *Mathematics of Computation*, *64*(212), 1517–1539. doi: https://doi.org/10.2307/2153369

Oliveira, I. F. D. (2013). Optimal black-box sequential searching. Masters Thesis of the Mathematics Department of the Federal University of Minas Gerais. Retrieved from `http://hdl.handle.net/1843/EABA-98VHPQ` (Advised by Professor R.H.C. Takahashi.)

Oliveira, I. F. D., & Takahashi, R. H. C. (2020a). An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software*.

Oliveira, I. F. D., & Takahashi, R. H. C. (2020b). Minmax-optimal list searching with $O(\log_2 \log_2 n)$ average cost. *Under consideration by the Journal of Computer and System Sciences, Elsevier*. (Pre-print available at `https://arxiv.org/abs/2105.11919`)

Oliveira, I. F. D., & Takahashi, R. H. C. (2021a). Efficient solvers for armijo's backtracking problem.
(Pre-print available at `https://arxiv.org/abs/2110.14072`)

Oliveira, I. F. D., & Takahashi, R. H. C. (2021b). An incremental descent method for multi-objective optimization. *Under consideration by the journal Optimization Methods and Software, Taylor & Francis*. (Pre-print available at `https://arxiv.org/abs/2105.11845`)

Pérez, L. R. L., & Prudente, L. F. (2019). A wolfe line search algorithm for vector optimization. *ACM Transactions on Mathematical Software*, *45*(4), 1-23. doi: https://doi.org/10.1145/3342104

Perl, Y., Itai, A., & Avni, H. (1978). Interpolation search—a log log n search. *Communications of the ACM*, *21*, 550-553. doi: https://doi.org/10.1145/359545.359557

Perl, Y., & Reingold, E. M. (1977). Understanding the complexity of interpolation search. *Information Processing Letters*, *6*(6), 219-222. doi: https://doi.org/10.1016/0020-0190(77)90072-2

Povalej, Z. (2014). Quasi-newton's method for multiobjective optimization. *Journal of Computational and Applied Mathematics*, *255*(1), 765-777. doi: https://doi.org/10.1016/j.cam.2013.06.045

Pownuk, A., & Kreinovich, V. (2017). Why linear interpolation? *Mathematical Structures and Modeling*, *3*(43), 43-49.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: the art of scientific computing. In (6th ed. ed., p. 442-486). Cambridge, UK: Cambridge University Press.

Ralha, R. (2012). The geometric mean algorithm. *Applied Mathematics and Computation*, *219*(4), 1607-1615. doi: https://doi.org/10.1016/j.amc.2012.08.002

Ralha, R. (2018). Mixed precision bisection. *Mathematics in Computer Science*, *12*, 173–181. doi: https://doi.org/10.1007/s11786-018-0336-6

Rice, J. R. (1969). *A set of 74 test functions for nonlinear equation solvers* (Department of Computer Science Report No. 64-034). Purdue University.

Ridders, C. (1979). A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, *26*(11), 979–980. doi: https://doi.org/10.1109/TCS.1979.1084580

Ritter, K. (1994). Average errors for zero finding: lower bounds for smooth or monotone functions. *Aequationes Mathematicae*, *48*(2), 194-219. doi: https://doi.org/10.1007/BF01832985

Schlegel, B., Gemulla, R., & Lehner, W. (2009). k-ary search on modern processors. *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, 52-60.

Segura, J. (2010). Reliable computation of the zeros of solutions of second order linear odes using a fourth order method. *SIAM Journal on Numerical Analysis*, *48*(2), 452-469. doi: https://doi.org/10.1137/090747762

Shi, Z. J., & Shen, J. (2005). New inexact line search method for unconstrained optimization. *Journal of Optimization Theory and Applications*, *127*, 425–446. doi: 10.1007/s10957-005-6553-6

Shrager, R. I. (1985). A rapid robust rootfinder. *Mathematics of Computation*, *44*(169), 151-165. doi: https://doi.org/10.2307/2007799

Sikorski, K. (1982). Bisection is optimal. *Numerische Mathematik*, *40*(1), 111-117. doi: https://doi.org/10.1007/BF01459080

Sikorski, K. (1985). Optimal solution of nonlinear equations. *Journal of Complexity*, *1*(2), 197-209. doi: https://doi.org/10.1016/0885-064X(85)90011-1

Stage, S. A. (2013). Comments on an improvement to the brent's method. *International Journal of Experimental Algorithms*, *4*(1), 1-16.

Tanabe, H., Fukuda, E. H., & Yamashita, N. (2019). Proximal gradient methods for multiobjective optimization and their applications. *Computational Optimization and Applications*, *72*, 339–361. doi: https://doi.org/10.1007/s10589-018-0043-x

Traub, J. F. (1963). Iterative methods for the solution of equations. *Bell Telephone Laboratories*, *8*(4), 550-551.

Truong, T. T., & Nguyen, H. T. (2021, September). Backtracking gradient descent method and some applications in large scale optimisation. part

2: Algorithms and experiments. *Applied Mathematics & Optimization*, *84*, 2557–2586. doi: https://doi.org/10.1007/s00245-020-09718-8

Vaswani, S., Mishkin, A., Laradji, I., Schmidt, M., Gidel, G., & Lacoste-Julieny, S. (2019). Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Proceedings of the 33rd conference on neural information processing systems - neurips).* Vancouver, Canada.

Vieira, D. A. G., Takahashi, R. H. C., & Saldanha, R. R. (2012). Multicriteria optimization with a multiobjective golden section line search. *Mathematical Programming*, *131*(1-2), 131–161. Retrieved from `https://doi.org/10.1007/s10107-010-0347-9` doi: 10.1007/s10107-010-0347-9

Vrahatis, M. N. (1988). Algorithm 666 chabis: A mathematical software package for locating and evaluating roots of systems of nonlinear equations. *ACM Transactions on Mathematical Software*, *14*(4), 330-336. doi: https://doi.org/10.1145/50063.51906

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM Review*, *11*(2), 226-235. doi: https://doi.org/10.1137/1011036

Wu, X. (2005). Improved muller method and bisection method with global and asymptotic superlinear convergence of both point and interval for solving nonlinear equations. *Applied Mathematics and Computation*, *166*(2), 299-311. doi: https://doi.org/10.1016/j.amc.2004.04.120

Yao, A. C., & Yao, F. F. (1976). The complexity of searching an ordered random table. *Proceedings of the Seventeenth Annual Symposium on Foundations of Computer Science*, 173-177. doi: https://doi.org/10.1109/SFCS.1976.32

Zhang, Z. (2011). An improvement to the brent's method. *International Journal of Experimental Algorithms*, *2*(1).

Zhou, X. (2018). On the fenchel duality between strong convexity and lipschitz continuous gradient. *arXiv Preprint arXiv:1803.06573*.