FRANCISCO NOGUEIRA CALMON SOBRAL

OTIMIZAÇÃO SEM DERIVADAS EM CONJUNTOS MAGROS

Este trabalho teve o apoio financeiro da FAPESP proc. nº 2008/00468-4

CAMPINAS
2012

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA

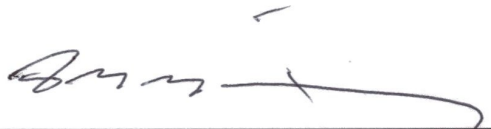
FRANCISCO NOGUEIRA CALMON SOBRAL

OTIMIZAÇÃO SEM DERIVADAS EM CONJUNTOS MAGROS

TESE DE DOUTORADO APRESENTADA AO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA DA UNICAMP PARA A OBTENÇÃO DO TÍTULO DE DOUTOR EM MATEMÁTICA APLICADA.

ORIENTADOR: PROF. DR. JOSÉ MARIO MARTÍNEZ PÉREZ

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE DEFENDIDA PELO ALUNO FRANCISCO NOGUEIRA CALMON SOBRAL E ORIENTADA PELO PROF. DR. JOSÉ MARIO MARTÍNEZ PÉREZ.



Prof. Dr. José Mario Martínez Pérez

CAMPINAS, 2012

FICHA CATALOGRÁFICA ELABORADA POR
ANA REGINA MACHADO - CRB8/5467
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

So12o Sobral, Francisco Nogueira Calmon, 1984-
Otimização sem derivadas em conjuntos magros / Francisco Nogueira Calmon Sobral. – Campinas, SP : [s.n.], 2012.

Orientador: José Mario Martínez Pérez.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Programação não-linear. 2. Otimização sem derivadas. 3. Otimização com restrições. 4. Métodos sem derivadas. I. Martínez Pérez, José Mario, 1948-. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

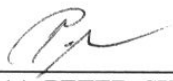
Informações para Biblioteca Digital

Título em inglês: Derivative-free optimization on thin domains
Palavras-chave em inglês:
Nonlinear programming
Derivative-free optimization
Constrained optimization
Derivative-free methods
Área de concentração: Matemática Aplicada
Titulação: Doutor em Matemática Aplicada
Banca examinadora:
José Mario Martínez Pérez [Orientador]
Peter Sussner
Ademir Alves Ribeiro
Elizabeth Wegner Karas
Ernesto Julián Goldberg Birgin
Data de defesa: 28-03-2012
Programa de Pós-Graduação: Matemática Aplicada

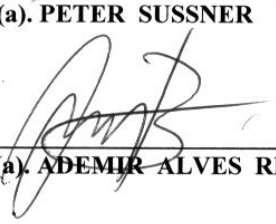
Tese de Doutorado defendida em 28 de março de 2012 e aprovada

Pela Banca Examinadora composta pelos Profs. Drs.

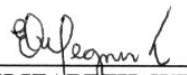
Prof(a). Dr(a). JOSÉ MARIO MARTÍNEZ PÉREZ



Prof(a). Dr(a). PETER SUSSNER



Prof(a). Dr(a). ADEMIR ALVES RIBEIRO



Prof(a). Dr(a). ELIZABETH WEGNER KARAS

Prof(a). Dr(a). ERNESTO JULIÁN GOLDBERG BIRGIN

Agradecimentos

Agradeço ao meu Pai, minha Mãe, meu pai e minha mãe por estarem permanentemente comigo, mesmo que muitas vezes eu não esteja com eles.

Agradeço ao Mario, por me mostrar que para todo problema difícil que eu era capaz de resolver, havia um problema fácil que eu não resolvia. Obrigado por me fazer sempre pensar no simples antes do complicado e por todos os valiosos ensinamentos.

Agradeço também ao Ernesto, por sempre ter me "obrigado" a ir para a carreira acadêmica e, embora não estivéssemos trabalhando diretamente desta vez, por continuar a me orientar sempre que precisei.

Agradeço especialmente à Meg, companheira das horas boas e, principalmente, das horas em que tudo dava errado. Embora ela ache que esse negócio de gordo e magro não seja coisa da matemática, sempre que precisei estava ali para me ouvir.

Agradeço aos meus irmãos, meus constantes exemplos, por mais que eu seja o mais velho.

Agradeço a todos os meus amigos de São Paulo, de Campinas e do sul. Embora muitos deles também achem que o negócio de engordar e emagrecer dê estrias, cada um deu sua contribuição a este trabalho de maneiras distintas. Todos foram pequenos mestres para a minha vida.

Por fim, mas não menos importante, agradeço à FAPESP.

Resumo

Os problemas de otimização sem derivadas surgem de modelos para os quais as derivadas das funções e das restrições envolvidas, por alguma razão, não estão disponíveis. Os motivos variam desde usuários que não querem programar as derivadas até funções excessivamente complexas e caixas-pretas, oriundas de simulações só possíveis graças ao crescimento na capacidade de processamento dos computadores. Acompanhando esse crescimento, o número de algoritmos para resolver problemas de otimização sem derivadas aumentou nos últimos anos. Porém, poucos são aqueles que conseguem lidar de forma eficiente com problemas cujos domínios são magros, como, por exemplo, quando há restrições de igualdade. Neste trabalho, apresentamos a teoria e implementação de dois algoritmos capazes de trabalhar com domínios magros em problemas de otimização sem derivadas. Ambos partem da premissa de que a parte mais custosa na resolução é a avaliação da função objetivo. Com isso em mente, o processo de resolução é dividido em duas fases. Na fase de restauração, buscamos por pontos menos inviáveis sem utilizar avaliações da função objetivo. Na fase de minimização, ou otimização, o objetivo é reduzir a função objetivo com o uso de algoritmos bem estabelecidos para problemas sem derivadas com restrições simples. O primeiro algoritmo utiliza ideias de Restauração Inexata associadas a uma tolerância decrescente à inviabilidade. Utilizando hipóteses simples e usuais dos métodos de busca direta direcional, mostramos propriedades de convergência a minimizadores globais. O segundo algoritmo recupera totalmente os resultados teóricos de um algoritmo recente de Restauração Inexata com busca linear e aplica-se a problemas nos quais apenas as derivadas da função objetivo não estão disponíveis. Testes numéricos mostram as boas propriedades dos dois algoritmos, em particular quando comparados com algoritmos baseados em penalidades.

Palavras-chave: conjuntos magros, convergência, convergência global, experimentos numéricos, implementação, otimização sem derivadas, Restauração Inexata, restrições de igualdade.

Abstract

Derivative-free optimization problems arise from models whose derivatives of some functions are not available. This information is unavailable due to extremely complex and black-box functions, originated from simulation procedures, or even to user inability. Following the growth in the number of applications, the number of derivative-free algorithms has increased in the last years. However, few algorithms are able to handle thin feasible domains efficiently, for example, in the presence of equality nonlinear constraints. In the present work, we describe the theory and implementation of two algorithms capable of dealing with thin-constrained derivative-free problems. Their definition considers that the objective function evaluation is the most expensive part of the problem. Based on this principle, the process of solving a problem is split into two phases. In the restoration phase, we try to improve the feasibility without evaluating the objective function. In the minimization phase, the aim is to decrease the objective function value by using well-established algorithms in order to solve derivative-free problems with simple constraints. The first algorithm uses Inexact Restoration ideas together with a decreasing infeasibility tolerance. Under the usual hypotheses of direct search methods, we show global minimization results. The second algorithm extends to the derivative-free case all the theoretical results obtained in a recent line-search Inexact Restoration algorithm. In this approach, only the derivatives of the objective function are not available. We perform numerical experiments to show the advantages of each algorithm, in particular when comparing with penalty-like algorithms.

Keywords: convergence, derivative-free optimization, equality constraints, global convergence, implementation, Inexact Restoration, numerical experiments, thin domains.

Sumário

Abreviações

Algencan (*Augmented Lagrangians using GENCAN for solving the subproblems*) algoritmo baseado em Lagrangianos Aumentados para problemas de programação não linear com restrições gerais. Para resolver os subproblemas de Lagrangiano Aumentado utiliza-se o algoritmo **GENCAN**. Veja [ABMS07].

BOBYQA (*Bound Optimization By Quadratic Approximation*) algoritmo baseado em regiões de confiança para problemas de otimização sem derivadas com restrições de caixa. Veja [Pow09], a Seção 1.6 e a Subseção 3.1.3.

DFO (*Derivative-free Optimization*) algoritmo baseado em regiões de confiança para problemas de otimização sem derivadas com restrições gerais. Veja [CST98] e a Seção 1.6.

GSS (*Generating Set Search*) algoritmo baseado em busca direta direcional para problemas sem derivadas com restrições lineares. Veja [KLT03] e a Seção 1.4.

HOPSPACK (*Hybrid Optimization Parallel Search Package*) algoritmo baseado em funções de penalidade para problemas de otimização sem derivadas com restrições gerais. Sua versão padrão utiliza Lagrangianos Aumentados. Veja [Pla09, GK10] e a Seção 1.5.

MADS (*Mesh Adaptive Direct Search*) classe de algoritmos para otimização sem derivadas com restrições gerais, baseada em busca direta direcional. Veja [ADJ06] e a Seção 1.2

NEWUOA (*New Unconstrained Optimization Algorithm*) algoritmo baseado em regiões de confiança para problemas irrestritos de otimização sem derivadas. Veja [Pow06, Pow11].

NOMAD (*Nonlinear Optimization with the MADS Algorithm*) algoritmo baseado em busca direta direcional para problemas de otimização sem derivadas com restrições gerais. O algoritmo é uma implementação eficiente do algoritmo MADS. Veja [LD11] e a Seção 1.2.

Lista de Algoritmos

Introdução

Existem problemas em otimização para os quais informações como gradiente e Hessiana da função objetivo e o Jacobiano das restrições não estão disponíveis ou o seu cálculo demanda operações muito complexas e demoradas. Os algoritmos para otimização sem derivadas são uma ferramenta bastante útil para tratar essas adversidades. Pelo fato de não utilizarem as derivadas, sua teoria de convergência pode ser estendida inclusive para problemas não suaves, nos quais os algoritmos clássicos de otimização não atingem resultados satisfatórios.

Ao falarmos de otimização sem derivadas, incluímos uma enorme diversidade de métodos como, por exemplo, o clássico método das diferenças finitas, no qual a derivada de uma função f no ponto x é aproximada por:

$$f'(x) = \frac{f(x+h) - f(x)}{h} \quad \text{ou} \quad f'(x) = \frac{f(x+h) - f(x-h)}{2h},$$

largamente conhecido e utilizado em análise numérica. Em otimização, podemos ainda citar o famoso método simplex de Nelder e Mead [NM65], métodos de aproximação por quadráticas como NEWUOA [Pow06] e a busca direta tradicionalmente associada ao trabalho de Hooke e Jeeves [HJ61].

O termo busca direta pode ter diversas interpretações, portanto, ao longo deste trabalho consideraremos a mesma interpretação utilizada na revisão bibliográfica [KLT03], que se baseou no método de Hooke e Jeeves. Torczon [Tor97] apresentou uma classe de métodos primeiramente denominada métodos de Busca Padrão Generalizada [ADJ03] (*generalized pattern search methods*), posteriormente denominada GSS [KLT03] (*Generating Set Search*), que utiliza apenas a avaliação da função objetivo em pontos escolhidos sobre uma malha que vai tornando-se cada vez mais fina. Supondo continuidade das derivadas da função, no caso irrestrito é possível provar que o método converge para um ponto estacionário do problema. Variações desse algoritmo tratam do problema quando há restrições de caixa e restrições lineares [LT99, LT00, KLT06b], com resultados de estacionariedade equivalentes.

Paralelamente aos trabalhos de Hooke e Jeeves, Powell [Pow70] apresentou os primeiros algoritmos que não utilizavam derivadas baseados em regiões de confiança. Winfield apresentou em [Win73] um dos primeiros estudos de minimização de funções aproximadas por interpolação, mas em [Pow94] Powell substituiu a aproximação de Taylor de segunda ordem, utilizada nos algoritmos clássicos de região de

confiança, pela aproximação polinomial. Uma grande quantidade de algoritmos se originou com base nessas ideias [CST97b, CST98, Pow02, Pow06, ST10]. Uma descrição geral dos métodos baseados em regiões de confiança sem derivadas para o caso irrestrito é encontrada em [CSV09a]. A inclusão de restrições, todavia, carece de maior aprofundamento teórico. Encontramos na literatura implementações para restrições de caixa [Pow09, BAEP11, GTT11] e restrições gerais [CST98], sem demonstração de convergência.

Inspirados em problemas com restrições lineares, cujas derivadas estão naturalmente disponíveis, em [LST02], Lucidi, Sciandrone e Tseng apresentaram um algoritmo para problemas nos quais apenas o gradiente da função objetivo não pode ser utilizado. As derivadas das restrições "quase ativas" são utilizadas para gerar um cone de direções de busca. O pontos candidatos são calculados através de uma estratégia que combina busca linear sem derivadas [LS02, DEMR08] e a projeção no conjunto viável. Os autores demonstraram que o algoritmo proposto converge a pontos estacionários.

Quando as restrições dos problemas tornam-se mais gerais, encontramos o trabalho de Audet e Dennis [ADJ06, ADJ09], que apresentaram um algoritmo baseado em busca direta direcional para problemas sem derivadas com restrições gerais, denominado Mads. Tanto a função objetivo quanto as restrições podem ser vistas como *caixas-pretas*, ou seja, dado um ponto, um valor é devolvido e não há o conhecimento dos cálculos que levaram a este valor. Da mesma forma, a caixa preta associada às restrições apenas indica se o ponto desejado é viável ou não. Para tratar as restrições foi aplicada uma técnica denominada **penalidade extrema**, que substitui a função objetivo original pela função:

$$f_X(x) = \begin{cases} f(x), & \text{se } x \in X \\ +\infty, & \text{caso contrário,} \end{cases}$$

onde X representa o conjunto viável. Essa abordagem permitiu que o problema fosse tratado como irrestrito. Uma outra vantagem da função penalidade extrema é que a verdadeira função objetivo não precisa ser avaliada quando o ponto não é viável, poupando recurso computacional. Utilizando uma estratégia de direções densas na hiperesfera unitária, os autores demonstraram convergência do algoritmo a pontos Clarke estacionários. Os resultados foram estendidos para problemas cujas funções apresentam descontinuidades [VC10].

Como apontado em [ADJ06], os algoritmos baseados em busca por direções viáveis têm dificuldades quando o conjunto viável é magro. Como magro, podemos entender que, para um número considerável de pontos no conjunto, digamos x, dada qualquer direção d, a probabilidade de $x+d$ também estar no conjunto é muito pequena ou zero. Um caso particular de conjuntos magros são aqueles formados por restrições de igualdade. A função penalidade extrema ressalta as dificuldades que tais algoritmos encontram ao resolver problemas com restrições de igualdade.

Uma outra abordagem para problemas sem derivadas com restrições consiste nos algoritmos que utilizam função de penalidade. A ideia é penalizar as restrições consideradas difíceis e resolver uma sequência de subproblemas sem derivadas, compostos

pela função penalidade sujeita às restrições fáceis. Nesse sentido, há algoritmos baseados em penalidade externa [LLS10] e Lagrangianos Aumentados [LT02, LT10, DEMP11]. A abordagem utilizada em [LT02] baseou-se no algoritmo LANCELOT [CGT91]. Os trabalhos [LT10, DEMP11] basearam-se na abordagem do algoritmo Algencan [ABMS07], onde [LT10] utilizou o algoritmo GSS para resolver os subproblemas com restrições lineares e [DEMP11] utilizou a busca coordenada para restrições de caixa, mas, de acordo com o algoritmo usado no subproblema, restrições arbitrárias também podem ser consideradas como fáceis.

As vantagens da abordagem por função penalidade são: (i) problemas com restrições gerais de igualdade e desigualdade podem ser considerados, (ii) as derivadas das restrições difíceis não são necessárias e (iii) todos os algoritmos possuem resultados de convergência a pontos estacionários. Por outro lado, algoritmos que penalizam as restrições reúnem o objetivo de minimização e de viabilidade em uma única função. Quando um dos objetivos é consideravelmente mais fácil do que o outro, não são capazes de aproveitar e incorporar essa nova informação. Além disso, sabe-se que algoritmos baseados em função de mérito (como Lagrangianos Aumentados, por exemplo) têm dificuldades com problemas que apresentam o fenômeno da voracidade [CMMS10], no qual os minimizadores irrestritos da função de mérito exercem uma forte atração nas iterações iniciais.

Um terceiro grupo de algoritmos também é capaz de resolver problemas sem derivadas com restrições gerais. Ele é composto pelos algoritmos de região de confiança para problemas sem derivadas que utilizam heurísticas para trabalhar com pontos viáveis [CST98, BB05]. Tais algoritmos nasceram da observação de que os subproblemas de região de confiança podem incorporar as restrições do problema original [CST98]. Porém, para resolvê-los, é necessário que haja algoritmos capazes de lidar com as restrições. No caso em que as derivadas das restrições estão disponíveis, os subproblemas tornam-se problemas de programação não linear com derivadas, que podem ser resolvidos com algoritmos clássicos e eficientes. Apesar de apresentarem bons resultados numéricos, tais algoritmos não possuem teoria de convergência.

Neste trabalho, propomos a inclusão de uma nova abordagem na resolução de problemas sem derivadas com restrições. Seu objetivo principal é contornar os inconvenientes apresentados pelas estratégias existentes até o momento. Isso significa que desejamos resolver problemas sem derivadas cujos conjuntos viáveis são magros. Supomos ainda que o custo computacional da avaliação da função objetivo é consideravelmente mais elevado do que o custo das restrições. Desta forma, se a tarefa de encontrar um ponto no conjunto viável exige procedimentos complexos (porém baratos, se comparados com o custo da função objetivo), então métodos de penalidade terão um trabalho árduo ao procurar pontos viáveis carregando junto o custo da função objetivo. Nesse caso, temos motivos suficientes para acreditar que o objetivo de viabilidade é mais fácil de ser atingido do que o objetivo de minimização, sugerindo uma separação dos processos.

Para aproveitar essa informação adicional do problema, nos algoritmos apresentados neste trabalho utilizamos uma estratégia baseada nos algoritmos de Restau-

ração Inexata [Mar98, MP00, FF10], que consiste em separar a resolução do problema em duas fases: a fase de restauração e a fase de otimização. Na fase de restauração, o objetivo é encontrar um ponto que aprimore a viabilidade sem utilizar avaliações da função objetivo. Qualquer esforço pode ser aplicado nessa fase, pois será considerado irrelevante quando comparado com o alto custo de uma avaliação funcional. Na fase de otimização, resolvemos subproblemas sem derivadas mais simples do que o problema original com o objetivo de reduzir o valor da função objetivo. Os subproblemas simples são construídos de forma que existam algoritmos eficientes e bem estabelecidos capazes de resolvê-los de forma aproximada.

O primeiro algoritmo que apresentaremos baseia-se na busca direta direcional [Tor97, ADJ06] e permite que problemas com restrições magras sejam resolvidos. Seu princípio de funcionamento reside no fato de que um conjunto magro deixa de sê-lo ao relaxarmos as restrições que o compõem, tolerando um determinado nível de inviabilidade. Assim, na fase de restauração, qualquer algoritmo é permitido na busca por um ponto no conjunto relaxado. Na fase de otimização (ou minimização), utilizamos algoritmos bem estabelecidos, capazes de lidar eficientemente com problemas sem derivadas em conjuntos gordos (não magros). O objetivo é resolver aproximadamente os subproblemas relaxados. Em cada iteração a tolerância torna-se menor até que, no limite, pontos viáveis são encontrados. Utilizando hipóteses de continuidade das funções envolvidas e a geração de um conjunto de direções de consulta denso em uma bola de raio Δ, resultados de convergência a minimizadores globais são obtidos.

A estratégia de utilizar uma tolerância decrescente da inviabilidade, restaurando pontos cuja inviabilidade está além de tal tolerância, não é nova. Em 1969, Paviani e Himmelblau [PH69] apresentaram um algoritmo, denominado método da tolerância flexível (*flexible tolerance method*), para resolver problemas de programação não linear com restrições. Na resolução, foi aplicado o método simplex de Nelder e Mead, desenvolvido originalmente para minimização irrestrita. No algoritmo, a tolerância à inviabilidade gera um relaxamento do conjunto viável original e é associada ao tamanho do simplex. Todo ponto candidato a constituir o simplex, e que se encontra fora do conjunto relaxado, é substituído por outro, cuja inviabilidade está dentro da tolerância. Claramente, este procedimento causa comportamentos anormais no método de Nelder e Mead, prejudicando sua convergência. Porém, a técnica de tolerâncias decrescentes é utilizada em diversos métodos recentes [BM05, BG08, GT10].

O segundo algoritmo que estudaremos neste trabalho estende, para o caso sem derivadas, o algoritmo de Restauração Inexata com busca linear de Fischer e Friedlander [FF10]. Com tal extensão, somos capazes de resolver problemas sem derivadas com restrições gerais suaves. Como já mencionamos, os algoritmos baseados em Restauração Inexata possuem duas fases: restauração e otimização. Há diversos algoritmos na literatura, que diferem entre si no critério utilizado para aceitar o ponto obtido na fase de otimização: regiões de confiança [MP00, Mar01], filtros [GKV03, KGR10] e busca linear [FF10]. Nenhum deles, todavia, considera o caso em que o gradiente da função objetivo não pode ser utilizado.

Para a correta e eficiente extensão do algoritmo de Fischer e Friedlander, no

caso dos problemas considerados neste trabalho devemos evitar ao máximo as avaliações da função objetivo. Supomos que as derivadas das restrições existem e podem ser utilizadas, pois serão necessárias para o novo algoritmo. Na fase de restauração, assim como no primeiro algoritmo, qualquer técnica pode ser utilizada para aprimorar a viabilidade encontrada até o momento. As avaliações da função objetivo devem ser evitadas nessa fase. Na fase de otimização, utilizando as derivadas das restrições, um subproblema dado pela função objetivo original sujeita à aproximação linear das restrições no ponto restaurado é resolvido. Para resolvê-lo, utilizamos o algoritmo GSS, desenvolvido especialmente para problemas sem derivadas com restrições lineares. Com tal descrição, sob hipóteses naturais para problemas sem derivadas e utilizando as boas propriedades do algoritmo GSS, demonstramos que o algoritmo proposto encontra pontos estacionários do problema original.

Os experimentos numéricos são uma parte fundamental no presente trabalho. Apresentamos sugestões adequadas para a implementação de cada passo dos algoritmos e sugerimos os parâmetros que se mostraram mais eficientes nos testes preliminares. Para demonstrar as vantagens dos dois algoritmos, realizamos comparações numéricas com outros algoritmos existentes na literatura, capazes de resolver problemas sem derivadas com restrições.

A estrutura do trabalho é descrita a seguir. No Capítulo 1, apresentamos um estudo detalhado dos princípios que regem os algoritmos de busca direta direcional. O objetivo é explicar o modo como operam, suas principais limitações e o seu principal representante: a classe de algoritmos Mads. Explicamos a complicada estratégia de direções densas na hiperesfera, utilizada por Mads [ADJ06], à luz de um algoritmo bastante simples, denominado Simplicíssimus. Esse algoritmo será a base para o algoritmo proposto no Capítulo 2. Aproveitamos o Capítulo 1 para apresentar detalhes de outros algoritmos para problemas sem derivadas com restrições que são referenciados, comparados e utilizados ao longo deste trabalho.

No Capítulo 2, descrevemos a teoria do primeiro algoritmo, baseado no relaxamento de Paviani e Himmelblau. Destacamos todas as hipóteses necessárias, culminando na demonstração de que o algoritmo proposto tem propriedades de convergência a minimizadores globais do problema. Basicamente, esses resultados são encontrados graças à utilização de direções de consulta densas em uma bola de raio $\Delta > 0$.

Com o algoritmo descrito no Capítulo 2, apresentamos uma implementação eficiente no Capítulo 3. Todos os procedimentos utilizados na fase de restauração e minimização são detalhadamente explicados. Grande parte dos parâmetros possui valores baseados na teoria sobre a qual o algoritmo foi construído, porém, para ajustar outros, realizamos experimentos numéricos preliminares. Com todos os parâmetros corretamente ajustados, comparamos o algoritmo resultante com outros algoritmos existentes que são capazes, inclusive heuristicamente, de trabalhar com restrições magras.

No Capítulo 4, apresentamos a teoria do algoritmo de Restauração Inexata sem derivadas. Discutimos as razões pelas quais tal extensão é recomendada para os problemas sem derivadas e concluímos com resultados de convergência a pontos estacionários. O Capítulo 5 destina-se à implementação adequada e cuidadosa dos passos

do algoritmo proposto no Capítulo 4. Nesse caso, sua própria teoria fornece indicação de implementações adequadas, confirmadas através de experimentos preliminares. A implementação final é comparada com outros algoritmos capazes de trabalhar com restrições e que se sustentam sobre alguma teoria de convergência.

Por fim, no Capítulo 6 apresentamos as conclusões do presente trabalho e uma série de possibilidades que surgiram no decorrer do estudo.

Notação

Acreditamos que todos os símbolos utilizados foram devidamente explicados antes de sua aplicação. Apresentamos aqui apenas alguns esclarecimentos.

- A norma de um vetor é representada por $\|\cdot\|$. Caso uma norma específica seja necessária para determinado procedimento, esta é claramente identificada: $\|\cdot\|_2$ representando a norma Euclidiana e $\|\cdot\|_\infty$ representando a norma infinito
- O número de variáveis é dado por n, o número de restrições de desigualdade é dado por m e o número de restrições de igualdade é dado por p
- $\mathbb{N}^*$ representa o conjunto $\{x \in \mathbb{N} \mid x \geq 1\}$
- $\mathbb{R}_+$ representa o conjunto $\{x \in \mathbb{R} \mid x \geq 0\}$
- X representa um conjunto viável genérico
- $\mathcal{X}$ representa o conjunto das restrições não-relaxáveis [CST98, ADJ06] (veja a Seção 2.4)
- $B(x, \delta) \doteq \{y \in \mathbb{R}^n \mid \|y - x\| < \delta\}$
- $\overline{B(x, \delta)} \doteq \{y \in \mathbb{R}^n \mid \|y - x\| \leq \delta\}$

Capítulo 1

Algoritmos para problemas sem derivadas

Neste capítulo discutimos e apresentamos diversos algoritmos para problemas de otimização sem derivadas encontrados na literatura. Não pretendemos, todavia, cobrir todos os algoritmos existentes, tarefa que seria impossível dada a grande quantidade e diversidade existentes.

Nosso objetivo é, primeiramente, apresentar algoritmos simples, baseados em busca direta. Discutimos as ideias essenciais e principais limitações, pois esses algoritmos foram a inspiração inicial para o desenvolvimento de um novo algoritmo, apresentado no Capítulo 2. Em segundo lugar, desejamos apresentar uma descrição sucinta dos algoritmos existentes na literatura de otimização sem derivadas capazes de lidar com restrições. Dentre os diversos existentes, escolhemos aqueles que foram utilizados nos testes comparativos e também aqueles que utilizamos para resolver os subproblemas gerados pelos algoritmos propostos neste trabalho.

O conteúdo inicial deste capítulo está fortemente baseado no livro de Conn, Scheinberg e Vicente [CSV09b] e na excelente revisão bibliográfica de Kolda, Lewis e Torczon [KLT03] sobre métodos de busca direta.

Na Seção 1.1 apresentamos um método simples de busca direta para problemas irrestritos. Tal método é conhecido atualmente como busca coordenada, mas uma de suas referências primordiais é atribuída ao trabalho de Hooke e Jeeves [HJ61]. Utilizando-o como exemplo, apresentamos teorias de convergência, limitações e extensões. Sua extensão mais geral, conhecida com MADS [ADJ06], é apresentada na Seção 1.2. Nessa seção discutimos a utilização de um conjunto denso de direções de busca na tentativa de trabalhar com restrições gerais. Baseando-nos em MADS, na Seção 1.3 estudamos um pouco mais profundamente os argumentos de densidade das direções de busca. Também apresentamos sua limitação em resolver problemas com restrições "magras".

Na Seção 1.4 discutimos os principais resultados da classe de algoritmos GSS [KLT03, KLT06b] para problemas sem derivadas com restrições lineares. Na Seção 1.5 apresentamos algoritmos baseados em Lagrangianos Aumentados, que são ca-

pazes de resolver problemas com restrições gerais, sem utilizar sequer as derivadas das mesmas [LT10, DEMP11]. Por fim, na Seção 1.6 apresentamos resultados de convergência associados com o método DFO [CST97b, CST98], baseado em regiões de confiança. Todos os algoritmos apresentados neste capítulo foram utilizados nos testes comparativos, embora DFO tenha também sido usado na resolução de subproblemas do algoritmo desenvolvido no Capítulo 2 (veja Capítulo 3).

1.1 Convergência de um algoritmo simples

Quando desejamos minimizar uma função para a qual não temos o gradiente disponível, um dos primeiros métodos que vem à cabeça é avaliar a função em certos lugares e deslocar-se para o ponto com menor valor funcional. Como a expressão "certos lugares" é muito vaga e inspirando-se na definição das derivadas parciais de uma função surge o método que estudaremos nesta seção: a **Busca Coordenada**. Esse método faz exatamente o que o nome diz: em cada passo a função objetivo é avaliada em $2n$ pontos, baseados nas n direções coordenadas (e_i) e seus respectivos opostos ($-e_i$). Ao longo de todo este trabalho utilizaremos que o vetor $e_i \in \mathbb{R}^n$ possui zeros em todas suas posições, exceto na i-ésima, que vale 1, ou seja,

$$[e_i]_j = \begin{cases} 0, & \text{se } j \neq i \\ 1, & \text{se } j = i. \end{cases}$$

No caso bidimensional os "certos lugares" que mencionamos acima são definidos pelas direções coordenadas: norte, sul, leste e oeste.

Começaremos com o estudo do comportamento da busca coordenada. Embora simples, esse algoritmo possui a mesma essência dos algoritmos de busca direta mais complexos, apresentados na literatura. De acordo com [CSV09b], nos métodos de busca direta os iterados são definidos apenas calculando-se a função objetivo em pontos pré-determinados. No caso da busca coordenada e similares, tais pontos são determinados por direções, motivo pelo qual são denominados métodos de busca direta *direcional*. Um outro tipo é o método de busca direta *baseado em simplex*, no qual os iterados são definidos por contrações, expansões e reflexões de um simplex. O algoritmo de Nelder e Mead [NM65] é o exemplo mais popular.

Nos algoritmos estudados, consideramos um problema básico de programação não linear:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & x \in X, \end{array} \tag{1.1}$$

onde $X \subseteq \mathbb{R}^n$ e $f : X \to \mathbb{R}$. O Algoritmo 1.1 descrito a seguir considera apenas o caso irrestrito $X = \mathbb{R}^n$. Um outro caso simples ocorre quando X é dado por uma caixa. Existem ainda bons algoritmos que tratam apenas com restrições lineares e algoritmos que tratam com restrições gerais, como veremos ao longo deste capítulo. O conjunto D utilizado no algoritmo representa as direções de busca. Como estamos considerando a

Algoritmo 1.1: Busca coordenada para problemas irrestritos

Dados: $\alpha_{\text{ini}} > 0$ e $D \doteq \{e_1, \ldots, e_n, -e_1, \ldots, -e_n\}$
Entrada: $x^0 \in \mathbb{R}^n$

1. $k \leftarrow 0$
   $\alpha_0 = \alpha_{\text{ini}}$
2. **se** existe $d^k \in D$ tal que $f(x^k + \alpha_k d^k) < f(x^k)$ **então**
   $x^{k+1} = x^k + \alpha_k d^k$
   $\alpha_{k+1} = \alpha_k$
   senão
   $x^{k+1} = x^k$
   $\alpha_{k+1} = \alpha_k/2$
3. $k \leftarrow k + 1$
   Volte para 2

busca coordenada, o conjunto é dado pelas n direções coordenadas em união com suas respectivas direções opostas.

Uma das maneiras mais intuitivas de compreendermos o funcionamento da busca coordenada é imaginar que estamos discretizando o conjunto viável com uma malha. O conjunto de direções D determina a estrutura e o termo α_k define o grau de refinamento da malha no passo k do algoritmo. Os pontos formados nas intersecções dessa malha são os candidatos a minimizadores para o respectivo grau de refinamento e é sobre alguns desses pontos que a função objetivo será avaliada em busca de menores valores funcionais, como mostra a Figura 1.1.

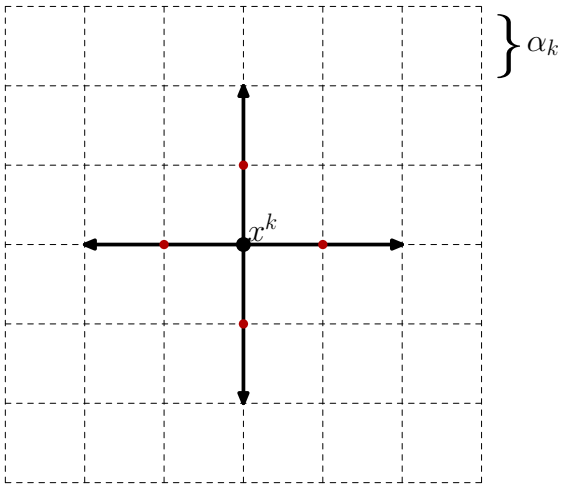


Figura 1.1: Dado o ponto x^k, o Algoritmo 1.1 constrói implicitamente uma malha usando as direções coordenadas e refinando-a de acordo com α_k. Nesta figura, $\alpha_k = 1/2$.

O conjunto D do Algoritmo 1.1 é um exemplo de um **conjunto gerador positivo** de $\mathbb{R}^n$. É fácil ver que todo elemento $x \in \mathbb{R}^n$ pode ser escrito como uma

combinação linear com coeficientes não negativos dos elementos de D. Essa propriedade é essencial para a prova de convergência do Algoritmo 1.1 a pontos estacionários de f. Na busca coordenada, os elementos de D são as direções coordenadas, mas qualquer conjunto de vetores que gere positivamente $\mathbb{R}^n$ pode ser usado. Os diferentes tipos de conjuntos definem malhas com formatos diferentes, como podemos ver na Figura 1.2(a) e, de fato, há algoritmos que se aproveitam dessa liberdade de escolha para trabalhar com mais de um conjunto de direções de busca. Podemos ver na Figura 1.2(b) uma malha construída com a união de dois conjuntos que geram positivamente $\mathbb{R}^n$.

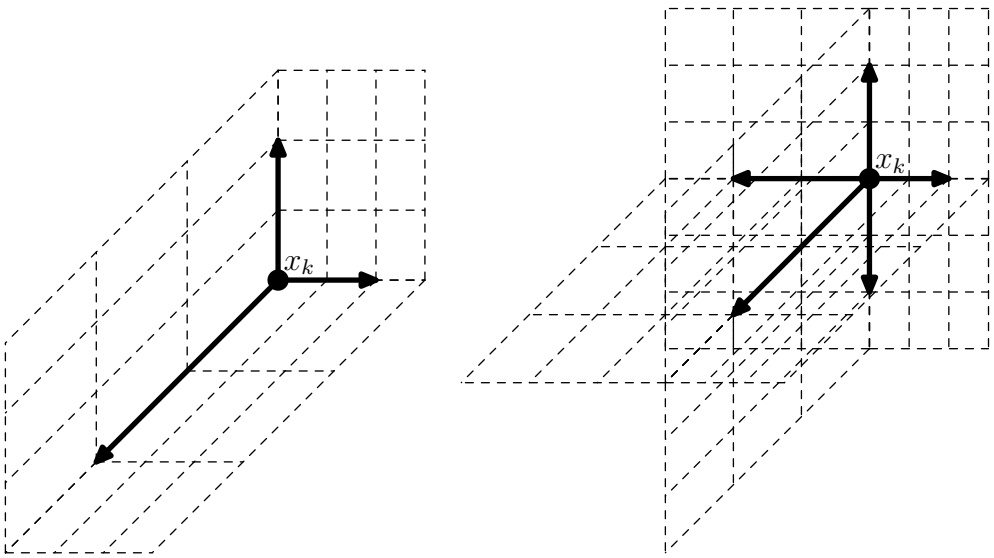


Figura 1.2: (a) Uma malha definida por um conjunto de direções diferente das direções coordenadas. (b) Uma malha definida por dois conjuntos que geram positivamente $\mathbb{R}^n$.

O Passo 2 do Algoritmo 1.1 procura por pontos na malha que diminuam o valor da função objetivo no ponto x^k. Quando um ponto é encontrado, então esse ponto torna-se o novo ponto central do algoritmo. A escolha da direção d^k adequada, caso exista mais de uma que decresça o valor de f, não influencia nas propriedades básicas de convergência do algoritmo. Quando nenhum dos pontos da forma $x^k + \alpha_k d$, $d \in D$, diminui o valor funcional de f então podemos dizer que o ponto corrente x^k é o minimizador para o respectivo grau de refinamento da malha, logo, concluímos que ela deve ser refinada ainda mais. Dessa forma, o refinamento só ocorre quando há um fracasso na busca pelo ponto na malha com menor valor funcional. Ao contrário de [ADJ06, Tor97], no Algoritmo 1.1 uma vez refinada, a malha não pode ser "engrossada", o que facilita a compreensão da importância da malha, utilizada para a prova do Lema 1.1 apresentado a seguir. Em seguida discutimos hipóteses adicionais que permitem o engrossamento da malha em caso de sucesso na busca.

1.1 Lema
Suponha que $X = \mathbb{R}^n$. *Se* $\{x^k\}_{k\in K}$, $K \subseteq \mathbb{N}$, *é uma subsequência convergente da*

sequência gerada pelo Algoritmo 1.1, então $\lim_{k\to\infty} \alpha_k = 0$.

Demonstração:

Suponha por contradição que α_k não converge a zero. Pela definição do Algoritmo 1.1, a redução de α_k é feita dividindo-o por 2, logo, existem $\bar{\alpha}$ e $L_1 > 0$ suficientemente grande, tais que $\alpha_k = \bar{\alpha}$ para todo $k > L_1$. Seja x^* tal que $\lim_{k\in K} x^k = x^*$. Dado $\varepsilon > 0$, existe $L_2 > L_1$ tal que $\|x^k - x^*\| \leq \varepsilon$ para todo $k \in K$, $k > L_2$, ou seja, a partir de um determinado momento todos os termos da subsequência estarão contidos em um compacto.

O valor $\bar{\alpha}$ define uma malha, cuja intersecção com o compacto acima definido resulta em um conjunto finito de pontos a serem explorados. Por outro lado, o fato de α_k não decrescer implica que sempre existe um ponto na malha para o qual o valor da função f decresce estritamente. Como esse conjunto de pontos é finito, temos uma contradição e concluímos que α_k tende para zero.

■

Os argumentos centrais do Lema 1.1 encontram-se no fato da malha possuir sempre a mesma estrutura, ou seja, ser formada por finitos conjuntos de direções geradoras, e o fato da sequência (ou subsequência) estar em um compacto. O fato de conseguirmos mostrar que a sequência toda converge para 0 advém fortemente do não engrossamento da malha [Tor97]. Com esse mesmo espírito, para algoritmos mais elaborados [ADJ06, Tor97, ADJ03, KLT03] também é possível demonstrar que existe ao menos uma subsequência $K \subset \mathbb{N}$ tal que $\{\alpha_k\}_{k\in K} \to 0$. Esse argumento não pode ser usado quando permitem-se infinitos conjuntos de direções geradoras positivas no algoritmo. Quando um conjunto infinito de direções geradoras positivas é utilizado, ainda assim temos a intuição de uma malha, mas apenas em cada iteração. A ideia de encontrar um minimizador para o dado grau de refinamento da malha, ou a ideia de estar sempre sobre algum ponto de uma malha fixa ao longo das iterações se perde, pois em cada iteração estamos trabalhando com malhas diferentes. Embora sob aspectos práticos seja mais interessante a possibilidade de mudar de conjunto em cada iteração, na teoria alguns argumentos não podem ser utilizados. Porém, com outras técnicas, ainda assim pode-se chegar a resultados semelhantes ao do Lema 1.1. Uma técnica necessária, por exemplo, é pedir decréscimo suficiente em cada iteração, no mesmo estilo da condição de Armijo para problemas com derivadas.

Uma comparação interessante pode ser feita entre os algoritmos baseados em busca direta direcional para problemas irrestritos e os métodos de descida para problemas de programação não linear com derivadas. Em [KLT03] essa relação, que está associada com as condições impostas para provar convergência global, é ilustrada para métodos gerais, mas mostraremos aqui através da busca coordenada.

A condição do ângulo

$$\nabla f(x^k)^T d \geq \beta \|\nabla f(x^k)\| \|d\|,$$

que impede que uma direção de descida quase ortogonal a $-\nabla f$ no ponto corrente seja escolhida, é automaticamente satisfeita. É fácil ver que sempre existirá uma direção em D cujo ângulo com $-\nabla f$ é menor ou igual a 45^o, independentemente da função considerada, bastando apenas escolher o parâmetro correto para satisfazer a condição. Em [Tor97] mostra-se que a condição é satisfeita com $\beta = 1/\sqrt{n}$. Esse argumento pode ser estendido para qualquer algoritmo de busca direta direcional, contanto que em cada iteração um conjunto gerador positivo seja considerado.

Os passos pequenos também são naturalmente evitados na busca coordenada. É fácil ver que

$$\|x^{k+1} - x^k\| = \|\alpha_k d^k\| = \alpha_k,$$

ou seja, os passos serão pequenos se a malha estiver bastante refinada. Como veremos em seguida, o refinamento da malha é o critério de parada natural dos algoritmos de busca direta direcional. No caso da busca coordenada a igualdade ocorre porque $\|d\| = 1$ para todo $d \in D$. Para os métodos que usam outros tipos de direção de busca, é necessário colocar um limitante inferior $\|d\| \geq \xi > 0$, para todo $d \in D_k$, onde D_k, $k \in \mathbb{N}$, é um conjunto de direções que gera positivamente $\mathbb{R}^n$ e pode, eventualmente, variar em cada iteração.

Nos métodos de descida usuais, a condição de Armijo impede que passos grandes sejam dados quando o decréscimo da função ocasionado por eles é pequeno. Na busca coordenada definida pelo Algoritmo 1.1, tal controle é feito pelas restrições implicitamente impostas pela escolha da malha e da forma como a refinamos. O fato dos pontos de busca estarem sobre a malha e a sequência estar em um compacto impede o decréscimo desproporcional da função com relação ao tamanho do passo, nesse caso dado por α_k. Se permitimos que a malha possa engrossar no caso de sucesso então não conseguimos mais provar a convergência de $\{\alpha_k\}_{k\in\mathbb{N}}$ para 0, e sim a existência de uma subsequência convergente para 0. Para a prova, é necessário que toda a sequência $\{x^k\}_{k\in\mathbb{N}}$ esteja em um compacto. A ideia principal da demonstração é que, em um compacto, se engrossarmos demais uma malha, não haverá mais pontos de consulta, e se mantivermos a malha sem refinar, teremos um número finito de pontos para escolher. Com uma pequena modificação no Algoritmo 1.1, no Lema 1.2 podemos mostrar quais os argumentos necessários. A generalização desse argumento será dada na Seção 1.2.

1.2 Lema
Suponha que o Passo 2 do Algoritmo 1.1 é modificado para que $\alpha_{k+1} \leftarrow 2\alpha_k$ em caso de sucesso, e o resto permaneça inalterado. Suponha ainda que $X = \mathbb{R}^n$. Se $\{x^k\}_{k\in\mathbb{N}}$ está contida em um compacto, então $\liminf_{k\to\infty} \alpha_k = 0$.

Demonstração:

Vamos supor por contradição que existe $\underline{\alpha} = 2^t$, para algum $t \in \mathbb{Z}$, tal que $\alpha_k \geq \underline{\alpha}$ para todo $k \in \mathbb{N}$. É fácil ver que para todo $k \in \mathbb{N}$

$$x^{k+1} = x^k + \alpha_k d^k = x^{k-1} + \alpha_{k-1} d^{k-1} + \alpha_k d^k,$$

onde $d^k \in D$ é a direção escolhida no passo k do Algoritmo 1.1. Note que, quando o passo é de insucesso, podemos considerar $d^k = 0$. Continuando a expressão até x^0 temos:

$$x^{k+1} = x^0 + \sum_{j=0}^{k} \alpha_j d^j = x^0 + \sum_{j=0}^{k} 2^{r_j} d^j = x^0 + \sum_{j=0}^{k} 2^{r_j - t}(2^t d^j), \tag{1.2}$$

onde r_k representa o refinamento da malha no passo k. Como $r_k - t \geq 0$, temos que $2^{r_k - t}$ é natural. Por conseguinte, a relação (1.2) mostra que todos o pontos gerados pela modificação do Algoritmo 1.1 encontram-se em uma malha definida pelos vetores coordenados e com um refinamento dado por 2^t.

Para que a malha não seja refinada é necessário o decréscimo simples em cada iteração, mas isso é impossível, pois, por hipótese a sequência toda está contida em um compacto e (1.2) implica que o número de pontos nos quais isso pode acontecer é finito. Assim, temos uma contradição com o fato de $\alpha_k \geq \underline{\alpha}$, o que implica que existe $K \subseteq \mathbb{N}$ tal que

$$\lim_{k \in K} \alpha_k = 0.$$

■

A hipótese da sequência toda gerada pelo algoritmo estar em um compacto é o grande diferencial do Lema 1.2 quando comparado com o Lema 1.1. A hipótese do Lema 1.1 é mais fraca, porém ficamos impedidos de permitir um engrossamento da malha. Uma maneira de garantir que a sequência toda esteja em um compacto é pedir que o conjunto de nível

$$\{x \in \mathbb{R}^n \mid f(x) \leq f(x^0)\}$$

seja um compacto.

Como mencionamos, o parâmetro de refinamento α_k geralmente é a escolha natural para indicar o sucesso na busca por um ponto com menor valor funcional, sendo, portanto, o critério de parada para os algoritmos de busca direta direcional. Por essa razão, é essencial em uma demonstração de convergência mostrar que o refinamento pode se tornar tão pequeno quanto necessário. Em [KLT03] os autores afirmam que existem na literatura três modos de provar que o termo α_k converge (ou possui alguma subsequência que converge) a zero. O primeiro modo é aquele sobre o qual o Algoritmo 1.1 foi construído, ou seja, baseado em decréscimo simples. Para esse caso, os ingredientes principais necessários são uma estrutura fixa da malha e refinamentos e engrossamentos que seguem uma estrutura restrita, da qual o Algoritmo 1.1 é apenas um caso simples e particular. Veremos o caso geral na Seção 1.2 com o método MADS [ADJ06]. Também se encaixam nesse caso os métodos de busca padrão [Tor97, LT99, LT00] e o método GPS [ADJ03].

O segundo modo é de simples compreensão por sua semelhança com os métodos de descida utilizados em otimização com derivadas. Ele consiste em exigir decréscimo suficiente em cada iteração [LS02]:

$$f(x^k + \alpha_k d) \leq f(x^k) - \rho(\alpha_k), \tag{1.3}$$

onde $\rho : \mathbb{R}_+ \to \mathbb{R}_+$ é uma função contínua que deve ter as seguintes propriedades:

- ρ não é identicamente nula
- ρ é uma função crescente
- $\lim_{\alpha \to 0} \frac{\rho(\alpha)}{\alpha} = 0.$

Uma prática usual é definir $\rho(\alpha) = \alpha^2$. Esse modo é uma versão sem derivadas do decréscimo suficiente pedido na condição de Armijo para otimização irrestrita com derivadas. O objetivo de ambos é impedir que passos muito grandes sejam dados. Utilizando decréscimo suficiente, não é preciso que a malha seja dada por uma estrutura fixa, nem que seu refinamento seja definido de um modo especial, como ocorre no decréscimo simples. A única hipótese necessária para garantir que exista uma subsequência para a qual o parâmetro de refinamento convirja para zero é que a função f seja limitada inferiormente.

O terceiro modo permite que vetores geradores da malha sejam alterados sempre que ocorre um fracasso na busca por um ponto com menor valor funcional na malha. A ideia é tentar aproximar localmente a curvatura da função através das informações já computadas anteriormente. Em [CP01] o método é descrito com mais detalhes.

Na Figura 1.3 temos três iterações do Algoritmo 1.1. Como é mostrado nos teoremas a seguir e encontrado em todos os algoritmos baseados em busca direta direcional, a análise de convergência não está baseada nos casos em que o algoritmo encontra um ponto com menor valor funcional e sim nos casos de fracasso nessa busca. Quando um fracasso ocorre, sabemos que o ponto de referência é um minimizador local com respeito aos pontos definidos pelo respectivo refinamento da malha. Logo, devemos refiná-la ainda mais. Levando tal processo ao limite, temos os resultados desejados:

$$\frac{f(x^k + \alpha_k d) - f(x^k)}{\alpha_k} \geq 0 \quad \underset{\alpha_k \to 0}{\Longrightarrow} \quad \nabla f(x^k)^T d \geq 0,$$

se a função for continuamente diferenciável, por exemplo. Essa ligação com o fracasso na busca fica evidente quando observamos os três modos mencionados nos parágrafos anteriores. Em todos os modos, a suposição de que o parâmetro permanece longe do zero em todas as iterações implica que infinitos sucessos na busca por um menor valor funcional estão acontecendo. Tal comportamento não é o esperado, dado que a função é limitada inferiormente ou então estamos trabalhando em um espaço com número finito de elementos.

No Teorema 1.3 que se segue, provamos a convergência do Algoritmo 1.1 a pontos estacionários no caso irrestrito utilizando a hipótese adicional de que as derivadas parciais da função objetivo são Lipschitz contínuas. De acordo com [CSV09b] essa hipótese pode ser enfraquecida, dadas as particularidades do algoritmo que estamos estudando. De fato, a Lipschitz continuidade é necessária quando consideramos um

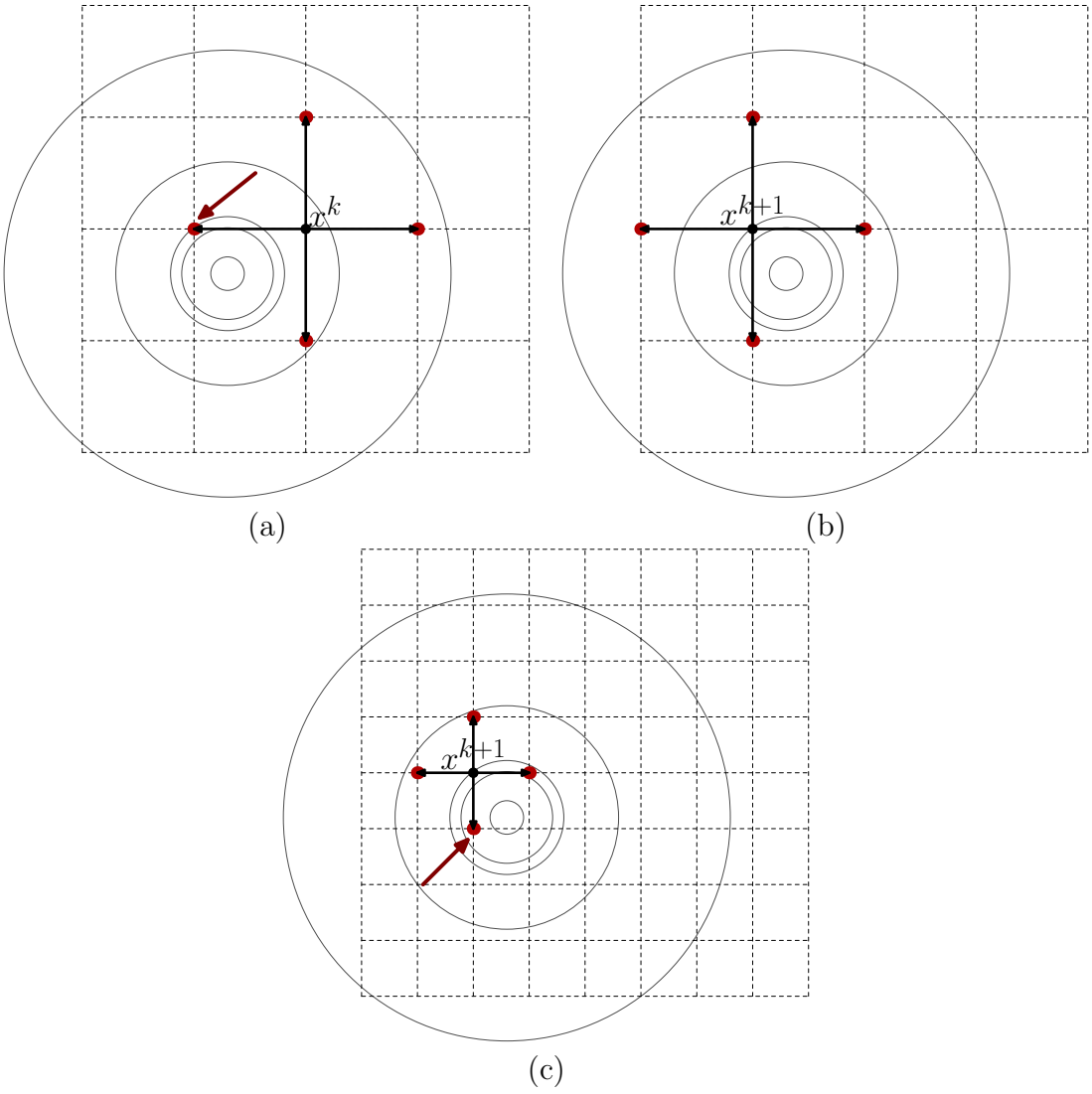


Figura 1.3: Exemplo do funcionamento do Algoritmo 1.1. Na figura (a) o valor da função no ponto x^k será comparado com os valores vizinhos na malha. As circunferências representam as curvas de nível de f. Em (b) temos que x^{k+1} é um minimizador local para os pontos consultados (fracasso) e a malha deve ser refinada, levando à iteração (c) na qual um ponto com menor valor funcional é encontrado e o algoritmo prossegue.

conjunto infinito de direções de busca. No caso finito, como mostra na sequência o Teorema 1.4, podemos exigir apenas continuidade das derivadas parciais de f.

1.3 Teorema
Suponha que as funções $\partial f(x)/\partial x_j$ são Lipschitz contínuas, com constante de Lipschitz L_j para $j \in \{1, \ldots, n\}$ e que $X = \mathbb{R}^n$. Se $\{x^k\}_{k \in K}$, $K \subseteq \mathbb{N}$, é uma subsequência convergente da sequência gerada pelo Algoritmo 1.1 e x^* é seu ponto de acumulação, então x^* é um ponto estacionário de (1.1).

DEMONSTRAÇÃO:

De acordo com o Lema 1.1, a subsequência $\{\alpha_k\}_{k\in K}$ converge para 0, o que implica que existe $K_1 \subseteq K$, tal que para todo $j \in \{1,\ldots,n\}$, $k \in K_1$,

$$f(x^k + \alpha_k e_j) \geq f(x^k) \quad \text{e} \quad f(x^k - \alpha_k e_j) \geq f(x^k),$$

ou seja, a função $\Phi_j(t) \doteq f(x^k + te_j)$ tem um minimizador $-\alpha_k < \bar{t}_j < \alpha_k$ para o qual

$$\frac{\partial f(x^k + \bar{t}_j e_j)}{\partial x_j} = \Phi_j'(\bar{t}_j) = 0.$$

Pela definição de $\Phi_j(t)$ e usando a hipótese de Lipschitz continuidade das derivadas, temos que

$$\left|\frac{\partial f(x^k)}{\partial x_j}\right| = \left|\frac{\partial f(x^k)}{\partial x_j} - \frac{\partial f(x^k + \bar{t}_j e_j)}{\partial x_j}\right| \leq L_j|\bar{t}_j| \leq L_j\alpha_k. \tag{1.4}$$

Como $\lim_{k\in K_1} \alpha_k = 0$, concluímos que x^* é um ponto estacionário de (1.1). ∎

Para garantir a existência de alguma subsequência convergente, podemos supor que os conjuntos de nível de f são compactos ou ainda pedir que o conjunto viável X seja um compacto. Uma maneira de fazer isso é colocar todo o problema em uma "caixa" n-dimensional. Tal procedimento é utilizado inclusive no caso irrestrito, no qual o problema original é colocado em uma caixa suficientemente grande de forma que os minimizadores não sejam perdidos.

No teorema a seguir, supomos que o conjunto X é formado apenas por restrições de caixa, ou seja, $l \leq x \leq u$, para $l, u \in \mathbb{R}^n$ e $l_i < u_i$ para todo $i = 1,\ldots,n$ e conseguimos mostrar que pontos de acumulação do Algoritmo 1.1 são pontos KKT. Os pontos KKT são aqueles que satisfazem as condições de Karush-Kuhn-Tucker conhecidas por, em associação com as condições de qualificação das restrições, serem condições necessárias de otimalidade em problemas com restrições [NW99]. Na demonstração, não utilizamos o fato das derivadas parciais serem Lipschitz contínuas, tornando o resultado mais forte, sendo que esse enfraquecimento das hipóteses é devido ao fato do conjunto de direções D, que gera positivamente $\mathbb{R}^n$, ser finito e não do fato de usarmos caixas no lugar de $\mathbb{R}^n$.

Para que a busca coordenada possa ser aplicada à otimização sem derivadas com restrições de caixa, há duas opções de modificação do Algoritmo 1.1. A mais simples e geral consiste em utilizar a **função penalidade extrema** no lugar da função objetivo f e aplicar o Algoritmo 1.1 normalmente. A função penalidade extrema, que denotaremos aqui por $f_X(x)$, é utilizada no trabalho de Audet e Dennis [ADJ06] para problemas com restrições gerais de desigualdade e é definida da seguinte forma:

$$f_X(x) = \begin{cases} f(x), & \text{se } x \in X \\ +\infty, & \text{caso contrário.} \end{cases} \tag{1.5}$$

De acordo com (1.5) podemos deduzir que, uma vez em posse de um ponto viável, o Algoritmo 1.1 nunca se interessará por pontos inviáveis, pois neles a função penalidade extrema possui maior valor funcional. Por esse motivo é de extrema importância que o ponto inicial do algoritmo seja viável. A outra modificação que podemos fazer no Algoritmo 1.1 consiste em tratar explicitamente as caixas no passo 2 do algoritmo. Esse processo pode ser visto como uma penalidade extrema apenas para caixas e é apresentado no Algoritmo 1.2. A vantagem desse processo é que independentemente do ponto inicial dado, sempre sabemos como projetar em uma caixa, além da compreensão do método ficar mais clara. No Teorema 1.4 que se segue, utilizamos a sequência gerada pelo Algoritmo 1.2 para mostrar convergência a pontos KKT de (1.1).

Algoritmo 1.2: Algoritmo de busca coordenada para restrições de caixa

Dados: $\alpha_{\text{ini}} > 0$ e $D \doteq \{e_1, \ldots, e_n, -e_1, \ldots, -e_n\}$
Entrada: $x^0 \in \mathbb{R}^n, \quad l \leq x^0 \leq u$

1. $k \leftarrow 0$
   $\alpha_0 = \alpha_{\text{ini}}$
2. **se** existe $d^k \in D$, $l \leq x^k + \alpha_k d^k \leq u$, tal que $f(x^k + \alpha_k d^k) < f(x^k)$ **então**
   $x^{k+1} = x^k + \alpha_k d^k$
   $\alpha_{k+1} = \alpha_k$
   senão
   $x^{k+1} = x^k$
   $\alpha_{k+1} = \alpha_k/2$
3. $k \leftarrow k + 1$
   Volte para 2

1.4 Teorema (Convergência em caixas)
Suponha que f é continuamente diferenciável em $X = \{x \in \mathbb{R}^n \mid l \leq x \leq u\}$. Se $\{x^k\}_{k \in K}$, $K \subseteq \mathbb{N}$, é uma subsequência convergente da sequência gerada pelo Algoritmo 1.2 e x^* é seu ponto de acumulação, então x^* é um ponto KKT de (1.1).

Demonstração:

Podemos verificar facilmente que todo ponto do conjunto X satisfaz a propriedade de que os gradientes das restrições ativas são linearmente independentes. As condições KKT para o problema (1.1) são dadas pela existência de vetores $\overline{\mu}$ e $\underline{\mu}$ tais que:

$$\begin{aligned} \nabla f(x) + \overline{\mu} - \underline{\mu} &= 0 \\ l \leq x &\leq u \\ [\overline{\mu}]_i (x_i - u_i) &= 0 \\ [\underline{\mu}]_i (l_i - x_i) &= 0 \\ \overline{\mu}, \underline{\mu} &\geq 0. \end{aligned}$$

Pelos mesmos argumentos utilizados no Lema 1.1 para o Algoritmo 1.1, podemos afirmar que $\{\alpha_k\}$, gerada pelo Algoritmo 1.2, converge para 0. Desta forma, podemos tomar $K_1 \subseteq K$ para a qual $\{\alpha_k\}_{k\in K_1}$ é estritamente decrescente.

Se $\dfrac{\partial f}{\partial x_j}(x^*) = 0$, podemos tomar $[\overline{\mu}]_j = [\underline{\mu}]_j = 0$. Caso $\dfrac{\partial f}{\partial x_j}(x^*) < 0$, vamos supor por contradição que

$$\frac{\partial f}{\partial x_j}(x^*) + [\overline{\mu}]_j - [\underline{\mu}]_j \neq 0. \tag{1.6}$$

Se $x_j^* = u_j$, tomando $[\overline{\mu}]_j = -\partial f(x^*)/\partial x_j$ e $[\underline{\mu}]_j = 0$ não teríamos (1.6). Portanto, sabemos que $x^* < u_j$ e podemos afirmar que $[x^* + te_j]_j \leq u_j$ para $0 \leq t \leq t'$, com t' suficientemente pequeno.

Pela continuidade de $\partial f(x^*)/\partial x_j$, existe $N > 0$ tal que, para $k \in K_1$, $k > N$

$$\frac{3}{2}\frac{\partial f(x^*)}{\partial x_j} < \frac{\partial f(x^k)}{\partial x_j} < \frac{1}{2}\frac{\partial f(x^*)}{\partial x_j} < 0.$$

Assim, aplicando a expansão de Taylor de primeira ordem,

$$\begin{aligned}\frac{f(x^k + te_j) - f(x^k)}{t} &= \frac{\partial f(x^k)}{\partial x_j} + \frac{o(t)}{t} \\ &< \frac{1}{2}\frac{\partial f(x^*)}{\partial x_j} + \frac{o(t)}{t} \\ &< 0,\end{aligned}$$

para todo $0 < t < t''$, onde t'' é suficientemente pequeno e não depende de j. Fazendo $\bar{t} = \min\{t', t''\}$ temos que em algum momento $\alpha_k < \bar{t}$, $k \in K_1$, e nesse caso α_k não poderá decrescer, pois

$$f(x^k + \alpha_k e_j) < f(x^k),$$

o que é uma contradição com o fato da subsequência K_1 ser apenas de decréscimos de α_k. Esse mesmo argumento pode ser feito no caso em que $\partial f(x^*)/\partial x_j > 0$, se considerarmos $-e_j$ e l_j no lugar de e_j e u_j. Logo, temos que x^* é um ponto KKT de (1.1). ■

O Algoritmo 1.1 e os teoremas apresentados anteriormente são um caso particular de uma classe de algoritmos chamados de **métodos de busca padrão** (do inglês *Pattern Search Methods*) apresentada por Torczon [Tor97] e generalizada por Audet e Dennis [ADJ03], que também a denominaram de **busca padrão generalizada** ou GPS (do inglês *Generalized Pattern Search*). O algoritmo denominado GSS [KLT03] (*Generating Set Search*) é uma suave generalização de GPS, na qual o decréscimo suficiente da função é pedido e, por consequência, um conjunto infinito de bases geradoras positivas de $\mathbb{R}^n$ pode ser utilizado.

Lewis e Torczon estenderam os resultados de convergência de [Tor97] para os casos em que o problema possui restrições de caixa [LT99] e restrições lineares [LT00].

Nesse último trabalho, utilizou-se técnicas de álgebra linear para gerar direções de busca que resultem em pontos de consulta sempre viáveis com respeito às restrições lineares. O aperfeiçoamento do Algoritmo 1.1 consiste em duas simples modificações. Em primeiro lugar, é permitido ao termo α_k tanto aumentar quanto ser reduzido, ou seja, a malha pode ser engrossada e refinada. O aumento de α_k ocorre quando um ponto com menor valor funcional é encontrado. Em segundo lugar, as direções do conjunto D, que formam a estrutura da malha, podem ser combinações inteiras de uma matriz $n \times n$ inversível qualquer, desde que qualquer elemento de $\mathbb{R}^n$ possa ser expresso por uma combinação linear positiva dos elementos de D. Porém, para poder considerar um conjunto viável qualquer, outras técnicas e hipóteses são utilizadas na literatura.

A extensão do Algoritmo 1.2 para outros tipos de restrições não é trivial. Em uma primeira tentativa, podemos utilizar a penalidade extrema (1.5) para evitar a saída do conjunto viável. Tal alternativa, porém, tem a desvantagem das direções de D não representarem suficientemente bem a geometria do conjunto viável ou das curvas de nível de f. De fato, as direções coordenadas usadas nos algoritmos 1.1 e 1.2 são boas representantes apenas das restrições de caixa. A Figura 1.4 ilustra a busca coordenada no caso em que há restrições lineares.

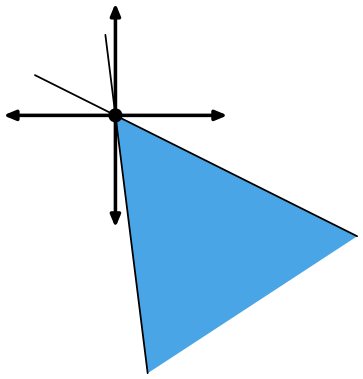


Figura 1.4: Dificuldade que o Algoritmo 1.1 encontra quando o conjunto de direções de busca não está de acordo com a geometria do conjunto viável (neste caso, dado pelo triângulo).

1.2 Mads

Nesta seção vamos descrever os algoritmos de busca direta direcional baseados em decréscimo simples da função objetivo, capazes de lidar com restrições gerais. Nesse sentido, o método mais geral encontrado na literatura é Mads. O problema de otimização sem derivadas com restrições gerais é considerado em [ADJ06]. O algoritmo apresentado, denominado Mads (*Mesh Adaptive Direct Search*) é uma generalização do algoritmo GPS de Lewis e Torczon. Antes relatar as diferenças de Mads para seus

antecessores, iremos descrever os passos comuns desse algoritmo e relacioná-los com a busca coordenada, um caso particular de MADS com a qual estamos familiarizados.

Como já mencionamos anteriormente, duas generalizações podem facilmente ser feitas no Algoritmo 1.1: considerar diferentes conjuntos D e considerar diferentes maneiras de atualizar o parâmetro de refinamento da malha.

Com respeito ao conjunto D, podemos considerá-lo como um conjunto de bases geradoras positivas de $\mathbb{R}^n$ e não mais uma única base geradora. Nesse sentido, em cada passo do algoritmo escolhemos $D_k \subseteq D$, de forma que D_k seja uma base geradora positiva de $\mathbb{R}^n$. No algoritmo desta seção, sempre consideramos que D é *finito*, e, por consequência, existe um número finito de subconjuntos distintos de D, em particular aqueles que formam uma base geradora positiva. A ideia dessa generalização é escapar de inconvenientes como o ilustrado na Figura 1.4, ou de funções que possuem curvas de nível mais complexas, como mostrado em [CSV09b]. No algoritmo MADS, D é formado por n_D vetores n dimensionais tais que

$$d_j = Gz_j, \text{ com } z_j \in \mathbb{Z}^n \quad j = 1, \ldots, n_D, \tag{1.7}$$

onde $G \in \mathbb{R}^{n\times n}$ é uma matriz inversível, chamada **matriz geradora**. Usando notação matricial, podemos escrever $D = GZ$, com $Z \in \mathbb{Z}^{n\times n_D}$. Definida a estrutura, a malha na qual o algoritmo irá procurar por pontos com menor valor funcional é definida por

$$M_k \doteq \bigcup_{x\in S_k} \{x + \alpha_k Dz \mid z \in \mathbb{N}^{n_D}\}, \tag{1.8}$$

onde S_k é o conjunto de todos os pontos viáveis nos quais a função objetivo foi avaliada até o passo $k-1$. É importante frisar que a malha nunca é construída por completo. Utilizando (1.8) é possível mantermos a noção de que o algoritmo caminha sobre uma malha, embora distintas (porém finitas) bases geradoras sejam consultadas.

A generalização que pode ser feita no parâmetro de refinamento da malha é mais simples: refiná-la em caso de fracasso e engrossá-la (ou mantê-la igual) em caso de sucesso na busca por um menor valor funcional. Para isso, definimos o parâmetro $\tau \in \mathbb{Q}$ e dois inteiros: $w^+ \geq 0$ e $w^- \leq -1$. Assim, dado um valor inicial $\alpha_0 > 0$ a atualização do parâmetro é descrita por:

$$\alpha_{k+1} = \alpha_k \tau^{w_k}, \text{ com } \begin{cases} w_k \in \{w^-, \ldots, -1\}, & \text{em caso de fracasso} \\ w_k \in \{0, \ldots, w^+\}, & \text{em caso de sucesso.} \end{cases} \tag{1.9}$$

Tanto a definição das direções do conjunto D, quanto a racionalidade de τ e a integralidade de w^+ e w^- são requisitos necessários para mostrarmos a existência de uma subsequência dos parâmetros de refinamento que converge para 0, como já discutimos na seção anterior.

Podemos ainda utilizar informações *a priori* do problema a ser resolvido para acelerar a convergência do Algoritmo 1.1. Isso de fato é feito nos algoritmos de busca direta direcional e consiste em realizar um passo de busca em um conjunto finito

de pontos especialmente definido para o problema. O passo é denominado **passo de busca** e ocorre antes do chamado **passo de consulta** (passo 2 do Algoritmo 1.1), devendo utilizar um número finito de pontos para ser considerado irrelevante no processo de demonstração de convergência a pontos estacionários. Embora teoricamente irrelevante, observa-se na prática [ADJ03, KLT03] que o passo de busca melhora muito o desempenho dos algoritmos de busca direta direcional. No caso de Mads e outros similares que utilizam decréscimo simples, é necessário também que os pontos de busca estejam contidos na malha M_k.

Toda a descrição de Mads que foi feita até o momento aplica-se para todo algoritmo de busca direta direcional que utiliza decréscimo simples da função objetivo. Mads, por sua vez, possui peculiaridades na escolha das direções do conjunto das direções de consulta D_k que não são encontradas em nenhum outro algoritmo. O grande diferencial de Mads, quando comparado com seus antecessores, é a introdução do parâmetro Δ_k^p, que determina o tamanho das direções de consulta. Além disso, Δ_k^p está ligado ao parâmetro α_k, que determina o refinamento da malha, devendo satisfazer:

$$\alpha_k \leq \Delta_k^p \quad \text{e} \quad \alpha_k \to 0 \iff \Delta_k^p \to 0. \tag{1.10}$$

Dessa forma, se permitimos que $\{\alpha_k\}$ convirja para zero mais rapidamente que $\{\Delta_k^p\}$, possibilitamos uma liberdade de escolha cada vez maior para as direções de consulta. A ideia é que as direções quase se libertem da malha, e possam ser basicamente qualquer vetor do $\mathbb{R}^n$ com norma menor ou igual a Δ_k^p. Podemos ter uma ideia do que acontece com Mads através da Figura 1.5.

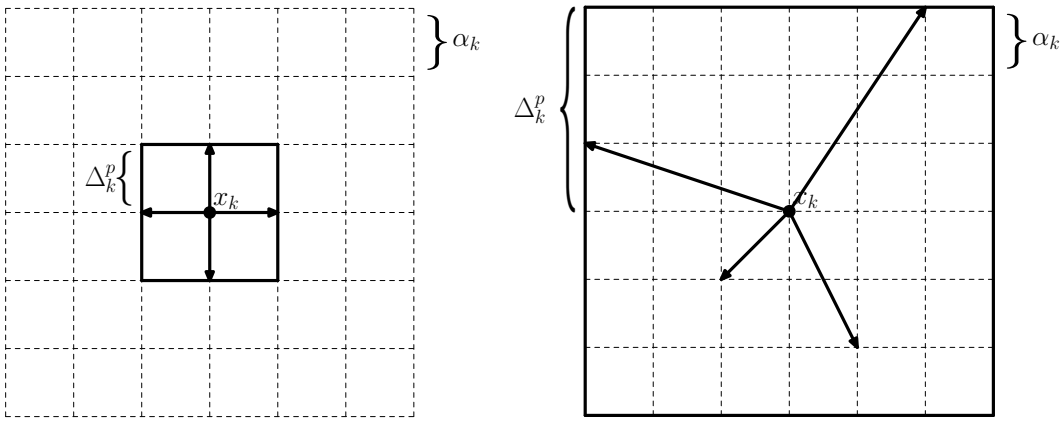


Figura 1.5: A principal diferença entre os algoritmos GPS (esquerda) e Mads (direita). Nas figuras, $D = \{-e_1, e_1, -e_2, e_2\}$ e $\alpha = 1/2$. O termo Δ_k^p delimita o tamanho que as direções de busca podem ter: enquanto que no GPS $\Delta_k^p = \alpha_k$, Mads permite que $\Delta_k^p \geq \alpha_k$. Se α_k decrescer mais rapidamente que Δ_k^p, existe a possibilidade de gerar uma variedade cada vez maior de direções de busca, como vemos à direita.

Para que haja mais liberdade na escolha das direções de consulta, as direções $d \in \mathbb{R}^n$ que fazem parte do conjunto D_k devem ser construídas de modo a satisfazer os três requisitos a seguir:

1. $d = Du$, $u \in \mathbb{N}^{n_D}$, com $d \neq 0$
2. A distância do ponto x^k para o ponto da forma $x^k + \alpha_k d$ é limitada superiormente, ou seja,
$$\alpha_k \|d\| \leq \Delta_k^p \max\{\|d'\| \mid d' \in D\}$$
3. Os limites dos conjuntos normalizados $\left\{ \frac{d}{\|d\|} \mid d \in D_k \right\}$ são conjuntos geradores positivos.

O requisito 1 garante que todas as direções de consulta estejam na malha M_k. O requisito 2 permite que um número maior de direções possa ser escolhido, desde que atualizemos corretamente o parâmetro Δ_k^p. Ele também garante que, se α_k está próximo de zero, então os pontos consultados estão cada vez mais próximos de x^k, pela condição (1.10). Na Figura 1.5 as possíveis direções de consulta são as intersecções da malha que estão dentro do quadrado de lados $2\Delta_k^p$. Notemos que a possibilidade de diferentes direções existe porque o valor de α_k é menor do que Δ_k^p. O algoritmo MADS é descrito no Algoritmo 1.3.

A sequência de resultados a seguir tem por objetivo mostrar a existência de uma subsequência da sequência gerada pelo Algoritmo 1.3, para a qual o parâmetro de refinamento converge a zero. Todos os resultados podem ser encontrados em [ADJ03, ADJ06]. Os argumentos se assemelham aos utilizados nos resultados de convergência dos algoritmos 1.1 e 1.2. Uma diferença está na substituição do fator de refinamento/engrossamento da malha, que era 2, pelo parâmetro racional τ.

1.5 LEMA (DISTÂNCIA ENTRE PONTOS DA MALHA)
Para todo $k \geq 0$ e qualquer norma, na qual um vetor inteiro não nulo possui norma maior ou igual a 1, vale que

$$u, v \in M_k,\ u \neq v \quad \Longrightarrow \quad \|u - v\| \geq \frac{\alpha_k}{\|G^{-1}\|},$$

onde M_k é a malha dada por (1.8).

DEMONSTRAÇÃO:

Como $u, v \in M_k$, temos que $u = x^k + \alpha_k D z_u$ e $v = x^k + \alpha_k D z_v$. Então:

$$\begin{aligned} \|u - v\| &= \alpha_k \|D(z_u - z_v)\| = \alpha_k \|GZ(z_u - z_v)\| && \text{por (1.7)} \\ &= \alpha_k \frac{\|G^{-1}\| \|GZ(z_u - z_v)\|}{\|G^{-1}\|} \geq \alpha_k \frac{\|Z(z_u - z_v)\|}{\|G^{-1}\|} \\ &\geq \frac{\alpha_k}{\|G^{-1}\|}. \end{aligned}$$

A última desigualdade ocorre pela hipótese sobre a norma e por que o vetor $Z(z_u - z_v)$ é inteiro e não nulo, pois $u \neq v$. ∎

Algoritmo 1.3: MADS

Dados:
$G \in \mathbb{R}^{n\times n}$ uma matriz inversível
D conjunto finito de geradores positivos de $\mathbb{R}^n$ de acordo com (1.7)
$\tau \in \mathbb{Q}$, $w^+ \geq 0$ e $w^- \leq -1$ inteiros.
Entrada: $x^0 \in X$

1. $k \leftarrow 0$
   $0 < \alpha_0 \leq \Delta_0^p$
2. Construa D_k de modo que satisfaça os requisitos 1, 2 e 3
3. **Passo de busca**
   Verifique em um número finito de pontos da malha M_k se existe $\bar{x}$ tal que
   $$f_X(\bar{x}) < f_X(x^k).$$
   Se encontrou tal ponto, faça $x^{k+1} = \bar{x}$ e vá para o passo 5.
4. **Passo de consulta**
   Verifique se existe $d^k \in D_k$ tal que
   $$f_X(x^k + \alpha_k d^k) < f_X(x^k).$$
   Se tal ponto existe, faça $x^{k+1} = x^k + \alpha_k d^k$ (sucesso), caso contrário tome $x^{k+1} = x^k$ (fracasso). Vá para o passo 5.
5. Escolha α_{k+1} de acordo com (1.9).
6. Escolha Δ_{k+1}^p para que valha (1.10).
7. $k \leftarrow k + 1$
   Volte para o passo 2.

1.6 LEMA
Suponha que a sequência $\{x^k\}$ gerada pelo Algoritmo 1.3 está contida em um compacto. Então existe r^+ tal que $\alpha_k \leq \alpha_0 \tau^{r^+}$ para todo $k \geq 0$.

DEMONSTRAÇÃO:

Seja γ a maior distância entre dois pontos do compacto no qual a sequência $\{x^k\}$ está contida. Se $\alpha_k > \gamma\|G^{-1}\|$ então pelo Lema 1.5, fazendo $v = x^k$, todos os pontos de M_k estarão fora do compacto, pois

$$\|x^k - u\| \geq \frac{\alpha_k}{\|G^{-1}\|} > \gamma,$$

obrigando o algoritmo a reduzir o valor de α_k. É fácil ver, pela descrição do Algoritmo 1.3, que para todo $k \geq 0$ existe um inteiro r_k para o qual

$$\alpha_k = \alpha_0 \tau^{r_k}.$$

Logo, podemos escolher r^+ suficientemente grande tal que $\alpha_0\tau^{r^+} \geq \gamma\|G^{-1}\|$ e conseguimos o resultado. ∎

Os lemas 1.5 e 1.6 não foram necessários para os resultados provados para o Algoritmo 1.2, pela escolha específica do conjunto D e do parâmetro de refinamento da malha. Como vemos no Lema 1.7 a seguir, que utiliza os mesmos argumentos do Lema 1.2, os lemas 1.5 e 1.6 são necessários para garantirmos que todos os pontos gerados pelo Algoritmo 1.3 se encontram em uma mesma malha.

1.7 Lema
Se a sequência $\{x^k\}$ gerada pelo Algoritmo 1.3 está contida em um compacto, então existe $K \subseteq \mathbb{N}$ tal que $\lim_{k\in K} \alpha_k = 0$. Por consequência, $\lim_{k\in K} \Delta_k^p = 0$.

Demonstração:

Sabemos que para todo $k \geq 0$ existe um inteiro $r_k \leq r^+$ (pelo Lema 1.6) tal que $\alpha_k = \alpha_0\tau^{r_k}$. Suponha por contradição que existe r^- tal que $\alpha_k \geq \alpha_0\tau^{r^-}$. Isso é equivalente a supor que não existe subsequência convergente de $\{\alpha_k\}$ para zero. Portanto, temos que

$$r_k \in \{r^-, r^- + 1, \ldots, r^+\} \tag{1.11}$$

para todo $k \geq 0$. De forma similar ao Lema 1.2, podemos verificar que para todo $k \geq 0$ temos que:

$$x^{k+1} = x^0 + \sum_{j=0}^{k} \alpha_j d^j = x^0 + \sum_{j=0}^{k} \alpha_0\tau^{r_j} Du^j,$$

onde $d^j = Du^j$, $u^j \in \mathbb{N}^{n_D}$, pertence ao conjunto D_j, que satisfaz os requisitos 1, 2 e 3, de acordo com o Algoritmo 1.3. Note que, em caso de fracasso na iteração k, consideramos $d^k = 0$. Como o parâmetro τ é um número racional podemos supor que é dado pela razão de dois números inteiros p e q primos entre si. Assim, por (1.11),

$$x^{k+1} = x^0 + \alpha_0 \sum_{j=0}^{k} \left(\frac{p}{q}\right)^{r_j} Du^j = x^0 + \sum_{j=0}^{k} \left(\frac{p^{r^-}}{q^{r^+}} D\right) \left(p^{r_j - r^-} q^{r^+ - r_j} u^j\right). \tag{1.12}$$

Por (1.11), sabemos que $p^{r_j - r^-}$ e $q^{r^+ - r_j}$ são inteiros. Logo, a equação (1.12) nos diz que todos os pontos da sequência $\{x^k\}$ estão em uma malha definida pelos vetores do conjunto D e com um refinamento dado por p^{r^-}/q^{r^+}. Como a sequência toda está em um compacto, utilizamos os mesmos argumentos do Lema 1.2 para chegar a uma contradição. Assim, existe uma subsequência $K \subseteq \mathbb{N}$ para a qual $\lim_{k\in K} \alpha_k = 0$. O fato de $\lim_{k\in K} \Delta_k^p = 0$ resulta do requisito (1.10). ∎

Em [ADJ06], uma subsequência $K \subseteq \mathbb{N}$ tal que $\{\alpha_k\}_{k\in K} \to 0$ é denominada de **subsequência refinada** (*refining subsequence*), pois está associada com uma

sequência de refinamentos da malha. Se $\{x^k\}_{k\in K} \to x^*$, a subsequência denomina-se **subsequência refinada convergente**. Uma direção $v \in \mathbb{R}^n$ é denominada **direção refinada** (*refining direction*) para x^* se existe $K_1 \subseteq K$ tal que $\lim_{k\in K_1} \frac{d^k}{\|d^k\|} = v$, onde d^k é uma direção viável de busca na iteração k, ou seja, $x^k + \alpha_k d^k \in X$.

A teoria de convergência de Mads baseia-se na hipótese de que o conjunto de direções refinadas para x^* construído pelo algoritmo é um bom representante da geometria do conjunto X em uma vizinhança de x^*. Porém, ainda é necessário explicar quando existem e como gerar as direções refinadas. Isso significa garantir que, no processo de convergência ao ponto x^*, um número suficientemente grande de pontos viáveis da forma $x^k + \alpha_k d^k$ foi consultado e, por consequência, seus valores funcionais $f(x^k + \alpha_k d^k)$ foram realmente calculados. Lembremos que, devido à função penalidade extrema, a verdadeira função objetivo não é calculada em pontos inviáveis.

As direções e os cones hipertangentes, definidos a seguir, indicam quais os requisitos necessários para que o algoritmo seja capaz de gerar direções refinadas.

1.8 Definição (Direção e cone hipertangente [ADJ06])
Um vetor $v \in \mathbb{R}^n$ é chamado ***vetor hipertangente*** *ao conjunto $X \subseteq \mathbb{R}^n$ no ponto $x \in X$ se existe $\varepsilon > 0$ tal que*

$$y + tw \in X \quad \text{para todo } y \in X \cap B(x,\varepsilon),\ w \in B(v,\varepsilon) \text{ e } t \in (0,\varepsilon). \tag{1.13}$$

O conjunto dos vetores hipertangentes a X em x é chamado ***cone hipertangente a X em x*** *e é denotado por $T_X^H(x)$.*

A Definição 1.8 afirma que, em uma vizinhança do ponto x somos capazes de encontrar pontos viáveis utilizando uma estratégia de busca direcional. Na Figura 1.6, apresentamos três exemplos de cones hipertangentes, representados pelas regiões cheias da figura. No primeiro, à esquerda, o conjunto X é dado pelo semiespaço à direita do hiperplano, incluindo o próprio hiperplano. O cone hipertangente a x é dado pelo semiespaço, excluindo o hiperplano. No segundo exemplo, ao centro, o conjunto X é dado pela união do primeiro quadrante de $\mathbb{R}^2$ com a reta $x - y = 0$. Se x é dado pela origem, o cone hipertangente é apenas o primeiro quadrante, excluindo os eixos. No terceiro exemplo, o conjunto X é dado pela curva. Nesse caso, o cone hipertangente a X em x é vazio.

A definição de uma direção hipertangente a X em x^* está intimamente ligada às direções refinadas do algoritmo Mads. Podemos ver que, se d é uma direção refinada, então existe uma subsequência K_1 tal que

(i) $\left\{\frac{d^k}{\|d^k\|}\right\}_{k\in K_1} \to d$;

(ii) $\{x^k\}_{k\in K_1} \to x^*$;

(iii) $\{\alpha_k \|d^k\|\}_{k\in K_1} \to 0$ (por (1.10) e pelo requisito 2) e

(iv) $x^k + \alpha_k d^k \in X$ para $k \in K_1$ suficientemente grande.

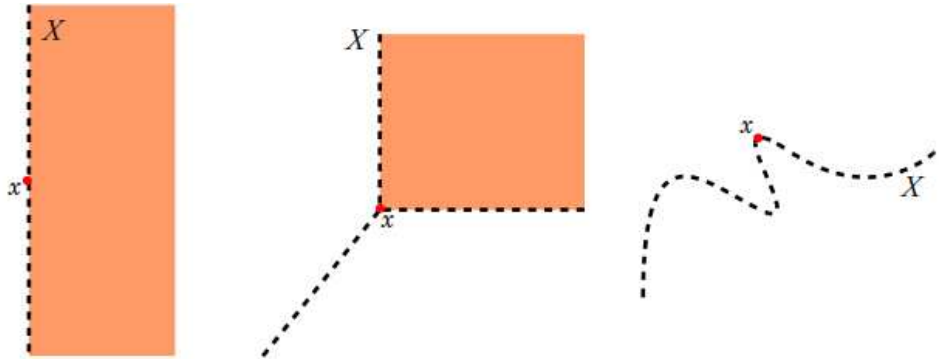


Figura 1.6: Exemplos de cones hipertangentes.

Por (i)–(iv), verificamos que, em (1.13), x^k pode fazer o papel de y, $\frac{d^k}{\|d^k\|}$ o papel de w e $\alpha_k\|d^k\|$ o papel de t. Além do mais, se $x^k + \alpha_k d^k \in X$ então

$$x^k + \alpha_k\|d^k\|\frac{d^k}{\|d^k\|} \in X \iff y + tw \in X.$$

Desta forma, se d é uma direção refinada, então existe a possibilidade de que d seja um vetor hipertangente a X em x.

Nosso interesse, todavia, é na recíproca. Suponha que existe uma direção $d \in T_X^H(x^*)$, onde x^* é o ponto limite de uma subsequência convergente $K \subseteq \mathbb{N}$ gerada pelo algoritmo. Então podemos tentar construir $\{d^k\}$ tal que $\lim_{k\in K_1} \frac{d^k}{\|d^k\|} = d$ para alguma subsequência $K_1 \subseteq K$. Logo, pela Definição 1.8 e pela descrição de MADS temos que, para $k \in K_1$ suficientemente grande, $x^k + \alpha_k d^k \in X$. Em outras palavras, d é uma direção refinada. Concluímos que, se $T_X^H(x^*)$ é não vazio e há um modo eficiente de gerar direções no algoritmo MADS que convirjam para direções em $T_X^H(x^*)$, as direções refinadas estão bem definidas.

Por esse motivo, a teoria de convergência de MADS está sustentada sobre a hipótese de que o conjunto das direções refinadas é denso no cone hipertangente ao conjunto X em x^*. O resultado principal de convergência de MADS é apresentado a seguir. Omitimos a demonstração porque vários conceitos de otimização não-suave são utilizados, tais como derivadas generalizadas de Clarke [Cla83] e sua generalização para restrições [Jah96]. A demonstração encontra-se em [ADJ06].

1.9 TEOREMA (CONVERGÊNCIA DE MADS)
Seja x^* o ponto limite de uma subsequência refinada, gerada por MADS *(Algoritmo 1.3). Suponha que f é Lipschitz contínua em uma vizinhança de x^* e que $T_X^H(x^*) \neq \emptyset$. Se o conjunto das direções refinadas para x^* é denso em $T_X^H(x^*)$, então x^* é um ponto Clarke estacionário de f em X.*

Esse teorema resume o que explicamos sobre MADS. Supondo que o cone hipertangente a X no ponto limite x^* é não vazio, garantimos que é possível construir

direções refinadas. Se as direções refinadas forem boas representantes da geometria do conjunto X (hipótese da densidade), então o ponto x^* possui algum tipo de estacionariedade. A estacionariedade de Clarke é a maneira não-suave de dizer que o ponto é KKT. De fato, se uma condição mais forte for imposta sobre f é possível provar que $-\nabla f$ pertence ao polar do cone tangente, conhecido como cone normal. Tal condição é uma condição necessária de otimalidade para problemas restritos com com derivadas [Ber99, Capítulo 3].

Na forma como está definido, não há garantias de que o conjunto das direções refinadas geradas por MADS satisfaça a hipótese de densidade no cone hipertangente, necessária para o Teorema 1.9. Por isso, dizemos que MADS define uma classe de algoritmos. A maneira de gerar as direções de busca usando simultaneamente os parâmetros Δ_k^p e α_k, mostrados na Figura 1.5, permite uma diversidade de aplicações práticas. Nesse sentido, duas implementações foram feitas com o objetivo de gerar direções refinadas densas: LTMADS [ADJ06] e ORTHOMADS [AADJLD09].

Recentemente, em [VC10], foi apresentada uma classe de algoritmos baseados em busca direta que também utiliza direções densas. Seu diferencial é que o ponto que diminui a função objetivo é escolhido através do critério de decréscimo suficiente (1.3). Essa pequena alteração permite que todas as condições impostas para definir MADS, tais como direções inteiras, reduções/expansões racionais, etc, possam ser descartadas. O resultado é um algoritmo bastante simples, que herda praticamente todas as propriedades de convergência de MADS.

1.3 Conjunto denso de direções

Para compreendermos o funcionamento de MADS e a aplicação da hipótese associada às direções refinadas sem precisarmos usar toda sua complicada nomenclatura, definimos um novo algoritmo, Algoritmo 1.4. O algoritmo é denominado Simplicíssimus, em alusão ao algoritmo de *Anonymous* [Ano72], do qual carrega forte semelhança. Dois teoremas de convergência são demonstrados para ele: no caso irrestrito e no caso similar ao Teorema 1.9. Supomos que as direções de consulta, no lugar de serem escolhidas do conjunto fixo D, são construídas de forma que a sua união seja densa na bola de raio Δ. Apesar de parecer uma hipótese forte, é semelhante à utilizada para as implementações LTMADS e ORTHOMADS.

Observe que o uso da função penalidade extrema f_X é apenas uma maneira de abreviarmos as seguintes condições

$$x^k + d^j \in X \quad \text{e} \quad f(x^k + d^j) < f(x^k)$$

na descrição do algoritmo. O teorema a seguir considera primeiramente o caso irrestrito. Para demonstrá-lo, precisamos de uma hipótese referente às direções efetivamente consultadas por Simplicíssimus. Note que, pela descrição de Simplicíssimus, é possível que existam direções do conjunto D_k que não foram consultadas.

Algoritmo 1.4: Simplicíssimus.

Dados: $\{\eta_k\}$, com $\eta_k \in \mathbb{N}^*$.
Entrada: $x^0 \in X$

1. $k \leftarrow 0$
2. Seja um conjunto de direções D_k, com η_k elementos.
3. **se** existe $j \in \{1, \ldots, \eta_k\}$ tal que $f_X(x^k + d^j) < f_X(x^k)$ **então**
   $x^{k+1} = x^k + d^j$
   senão
   $x^{k+1} = x^k$
4. $k \leftarrow k + 1$
   Volte para 2

Hipótese D – A sequência $\{\eta_k\}$ é limitada e o conjunto formado pela união das direções consultadas pelo Algoritmo Simplicíssimus é denso na bola de raio $\Delta > 0$.

1.10 Teorema
Considere o problema (1.1)*, com* $X = \mathbb{R}^n$ *e a sequência* $\{x^k\}$ *gerada por Simplicíssimus. Suponha que a Hipótese D é válida e que* f *é contínua. Se* $\lim_{k\to\infty} x^k = x^*$*, então* x^* *é um minimizador local de* (1.1)*. Mais ainda,* x^* *é um minimizador global do problema:*

$$\begin{array}{ll} \min & f(x) \\ s.\ a & \|x - x^*\| \leq \Delta. \end{array} \tag{1.14}$$

Demonstração:

Suponha por contradição que existe $z^* \in \mathbb{R}^n$, $\|z^* - x^*\| \leq \Delta$, tal que $f(z^*) < f(x^*)$. Definindo $d = z^* - x^*$, temos que $\|d\| \leq \Delta$ e sabemos que existe $K \subseteq \mathbb{N}$ tal que $\lim_{k\in K} d^k = d$, onde $d^k \in D_k$ pertence ao conjunto das direções consultadas por Simplicíssimus.

Pela continuidade de f temos que existe $\delta > 0$ tal que $f(z) < f(x^*)$ para todo $z \in B(z^*, \delta)$. Observe que

$$\|z^* - (x^k + d^k)\| = \|x^* + d - x^k - d^k\| \leq \|x^* - x^k\| + \|d - d^k\| < \delta,$$

para $k \in K$ suficientemente grande, o que é uma contradição com a hipótese da sequência ser convergente a x^*. Portanto, vemos que x^* é um minimizador de (1.14). ∎

No teorema anterior, vemos que a existência da subsequência K, na qual há direções de busca d^k tão próximas de d quanto quisermos, só está garantida devido à hipótese de convergência da sequência inteira $\{x^k\}$ ao ponto x^*. Uma maneira de contornar tal hipótese é supor que, para toda subsequência convergente S, $D = \cup_{k\in S} D_k$

seja denso na bola de raio Δ. Nesse caso, o Teorema 1.10 é apenas um caso particular, embora a prova seja essencialmente a mesma.

Não podemos nos deixar enganar pela excelente propriedade de convergência de Simplicíssimus, apresentada no teorema anterior. Pedindo apenas a continuidade da função f, demonstramos convergência a minimizadores locais do caso irrestrito de (1.1). Para Δ suficientemente grande, de forma a abranger todos os minimizadores, temos minimizadores globais. De fato, supor que o conjunto das direções utilizadas pelo algoritmo é denso na bola unitária é uma hipótese extremamente forte e, em aspectos práticos, uma tarefa infinitamente dispendiosa. Além disso, na teoria, gerar um conjunto denso na bola de raio Δ é uma tarefa tão difícil quanto gerar um conjunto denso no espaço todo. Sob esse aspecto, Simplicíssimus nada mais é do que uma versão local de um algoritmo que avalia a função objetivo em um conjunto denso e enumerável de pontos em $\mathbb{R}^n$. Usando essa estratégia, sabemos ser possível provar convergência a minimizadores globais, como é o caso do algoritmo de *Anonymous*, apresentado na primeira edição de *Mathematical Programming* (veja [Ano72]).

Por outro lado, o estudo de algoritmos como o Simplicíssimus lança um lampejo de luz sobre o que esperar de métodos que utilizam ideias semelhantes (entre eles MADS), além de estimular alternativas implementáveis de tais ideias. O Teorema 1.11 mostra uma das limitações do Algoritmo Simplicíssimus, quando são adicionadas restrições ao problema (1.1), ou seja, $X \subset \mathbb{R}^n$. Também compreenderemos por que na teoria de MADS existe a hipótese de que o cone hipertangente a X no ponto limite é não vazio.

1.11 TEOREMA

Suponha que a Hipótese D é válida e que f é contínua. Suponha ainda que o conjunto X é tal que a intersecção $B(x,\varepsilon) \cap X$ contém uma bola aberta, para todo $x \in X$ e para todo $\varepsilon > 0$. Se $\lim_{k\to\infty} x^k = x^*$, com $\{x^k\}$ gerada por Simplicíssimus, então x^* é um minimizador local do problema (1.1)*. Mais ainda, para todo $\delta \in [0, \Delta)$, x^* é um minimizador global do problema:*

$$\begin{array}{ll} \min\limits_{x\in X} & f(x) \\ s.\ a & \|x - x^*\| \leq \delta. \end{array} \tag{1.15}$$

DEMONSTRAÇÃO:

Suponha por contradição que existe $z^* \in \overline{B(x^*,\delta)} \cap X$ tal que $f(z^*) < f(x^*)$. Pela continuidade de f, $f(x) < f(x^*)\ \forall\, x \in B(z^*, \gamma_1)$, para algum $\gamma_1 > 0$. Pelo fato de $\{x^k\} \to x^*$, temos que, para k suficientemente grande e γ_1 suficientemente pequeno, $B(z^*, \gamma_1) \subset B(x^k, \Delta)$, pois $\delta < \Delta$. Sejam $\bar{z} \in B(z^*, \gamma_1) \cap X$ e γ_2 para os quais

$$B(\bar{z}, \gamma_2) \subset B(z^*, \gamma_1) \cap X \quad \big(\subset B(x^k, \Delta) \big).$$

Tanto $\bar{z}$ quanto $\gamma_2 > 0$ existem, pois pelas hipóteses do teorema, a intersecção $B(z^*, \gamma_1) \cap X$ contém uma bola aberta.

Como $\bar{z} \in B(x^k, \Delta)$, para k suficientemente grande, e sabendo que o conjunto das direções consultadas é denso na bola de raio Δ, podemos tomar uma subsequência $K \subset \mathbb{N}$ tal que $\lim_{k \in K} d^k = (\bar{z} - x^*)$. Vemos que

$$\|x^k + d^k - \bar{z}\| = \|(x^k - x^*) + (d^k - (\bar{z} - x^*))\| \leq \|x^k - x^*\| + \|d^k - (\bar{z} - x^*)\|$$

e em algum momento haverá um ponto consultado $x^k + d^k \in B(\bar{z}, \gamma_2)$, o que implica que $f(x^k + d^k) < f(x^k)$, pela continuidade de f. Isso é uma contradição com a definição do algoritmo, pois, nesse caso, x^k não pertenceria à sequência. ■

O Teorema 1.11 não pode ser estendido para considerar o caso $\delta = \Delta$. Na Figura 1.7, o ponto $z^* \in \overline{B(x^*, \Delta)}$, mas não é possível encontrar uma bola aberta arbitrariamente próxima de z^* que esteja contida tanto em X quanto em $B(x^*, \Delta)$.

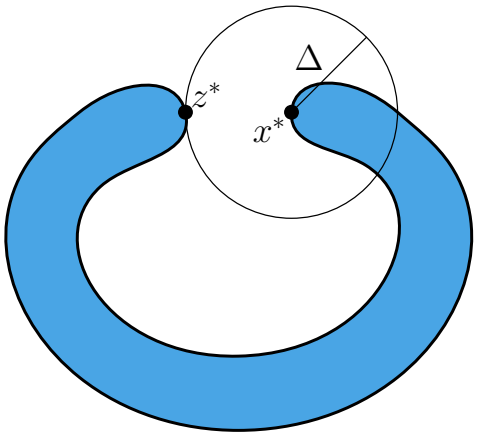


Figura 1.7: O algoritmo não consegue consultar pontos viáveis arbitrariamente próximos de z^*.

A hipótese de que, para todo $x \in X$ e para todo $\varepsilon > 0$, a intersecção $B(x, \varepsilon) \cap X$ contém uma bola aberta, é uma maneira explícita de dizermos que Simplicíssimus precisa que o conjunto X seja "gordo", em um sentido que será formalizado na Definição 1.12 a seguir. Hipóteses semelhantes em [ADJ06] também indicam a dependência do algoritmo MADS a esse tipo de conjunto, como a hipótese de que o cone hipertangente no ponto limite x^* não pode ser vazio. No capítulo que se segue, iniciamos um estudo de algoritmos que tentam contornar esse inconveniente e resolver uma classe maior de problemas utilizando os métodos que foram feitos para conjuntos gordos. No restante desta seção, explicamos os cones hipertangentes na luz dos conjuntos que chamaremos *conjuntos gordos*.

1.12 DEFINIÇÃO (CONJUNTO GORDO)
Um conjunto $X \subseteq \mathbb{R}^n$ é um ***conjunto localmente gordo em*** $\hat{x} \in X$ *se, para todo $\varepsilon > 0$, existem y e $\delta > 0$ tais que*

$$B(y, \delta) \subset B(\hat{x}, \varepsilon) \cap X.$$

De maneira análoga, X é um ***conjunto gordo*** *se é localmente gordo para todo $x \in X$.*

Uma pergunta natural é se existe relação entre a Definição 1.12, o fato do cone hipertangente a X em um ponto $\hat{x}$ ser não vazio e um conjunto possuir interior não vazio.

Realizamos, primeiramente, a relação entre a definição de conjunto localmente gordo em $\hat{x} \in X$ e o fato do cone hipertangente a X em $\hat{x}$, $T_X^H(\hat{x})$, ser não vazio. Sejam $\hat{x} \in X$ e $v \in T_X^H(\hat{x})$. Pela Definição 1.8 de cone hipertangente a X em $\hat{x}$, temos que existe $\hat{\varepsilon} > 0$ tal que

$$x' + tv' \in X,$$

para todo $x' \in B(\hat{x}, \hat{\varepsilon})$, para todo $v' \in B(v, \hat{\varepsilon})$ e $t \in (0, \hat{\varepsilon})$. Fixamos $\hat{t} \in (0, \hat{\varepsilon})$ e consideramos o ponto $\hat{x} + \hat{t}v$ e todos os pontos da forma

$$(\hat{x} + \hat{t}v) + w = \hat{x} + \hat{t}\left(v + \frac{w}{\hat{t}}\right),$$

com $\|w\| < \rho$, $w \in \mathbb{R}^n$. Então,

$$\left\| v - \left(v + \frac{w}{\hat{t}}\right)\right\| = \left\|\frac{w}{\hat{t}}\right\| < \frac{\rho}{\hat{t}}.$$

Portanto, para $\rho < \hat{t}\hat{\varepsilon}$, temos que $v + \dfrac{w}{\hat{t}} \in B(v, \hat{\varepsilon})$, e podemos concluir que a bola de centro em $\hat{x} + \hat{t}v$ e raio ρ está contida em X, pois $\hat{t} \in (0, \hat{\varepsilon})$ e $\hat{x} \in B(\hat{x}, \hat{\varepsilon})$. Dessa forma, para todo $\varepsilon > 0$, conseguimos encontrar $y \in X$ e $\delta < \min\{\varepsilon, \hat{\varepsilon}\}$, dados por valores adequados de ρ, tais que

$$B(y, \delta) = B(\hat{x} + \hat{t}v, \rho) \subset B(\hat{x}, \hat{\varepsilon}) \cap X,$$

ou seja, X é localmente gordo em $\hat{x}$.

A recíproca não é verdadeira, como mostra o seguinte exemplo. Considere o conjunto

$$X = \{(x, y) \in \mathbb{R}^2 \mid y - x^2 \leq 0 \text{ e } y + x^2 \leq 0\},$$

como mostra a Figura 1.8(a). Considerando, por exemplo, a norma $\|\cdot\|_\infty$, X é localmente gordo no ponto $(0, 0)$, pois para todo $\varepsilon > 0$ podemos tomar a bola (quadrada) $B((\varepsilon/2, 0), \min\{(\varepsilon/4)^2, \varepsilon/2\})$ que garantidamente estará contida na intersecção

$$B((0, 0), \varepsilon) \cap X.$$

O cone hipertangente $T_X^H(0, 0)$, por sua vez, é vazio.

Concluímos que a Definição 1.8 implica na Definição 1.12, mas a recíproca não é verdadeira. Logo, a hipótese de um conjunto ser localmente gordo em x é mais fraca do que a hipótese $T_X^H(x) \neq \emptyset$.

A comparação entre conjuntos gordos e conjuntos com interior não vazio é mais simples. Lembremos que o interior de um conjunto Ω é formado pelos pontos

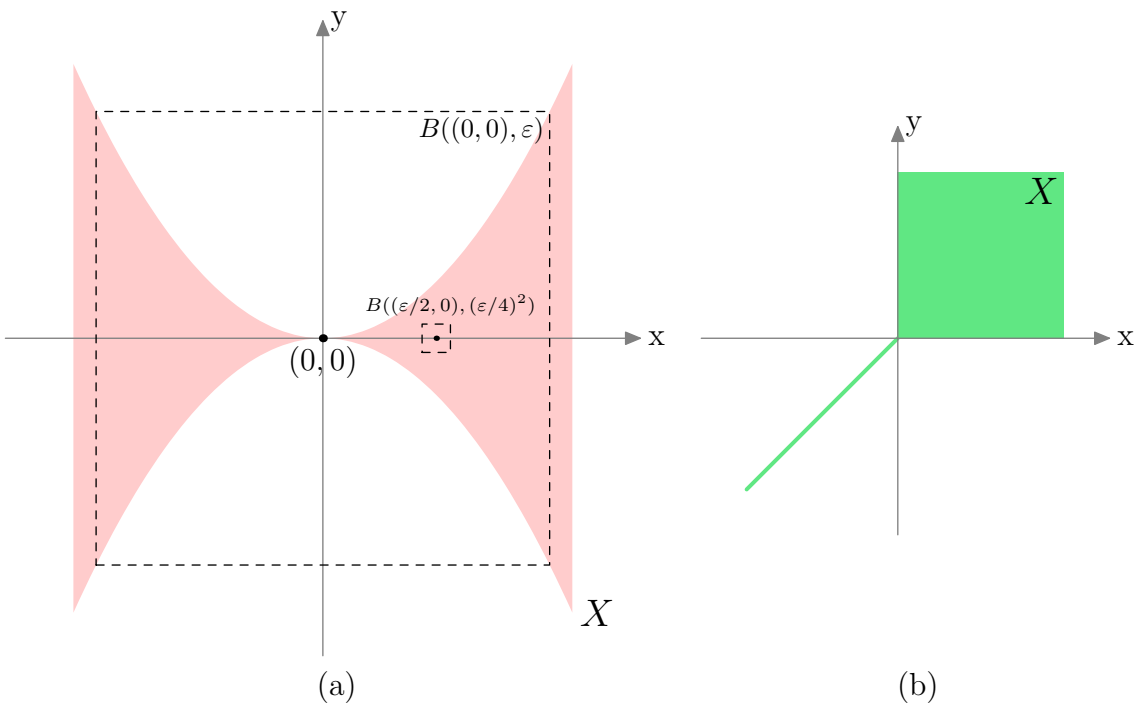


Figura 1.8: Exemplos para os casos em que (a) conjunto ser localmente gordo em um ponto não implica no cone hipertangente no mesmo ponto ser não vazio e (b) o conjunto possuir interior não vazio não implica em ser gordo.

$x \in \Omega$ tais que $B(x,\delta) \subset \Omega$, para algum $\delta > 0$. Com isso, é fácil verificar que se o conjunto X é localmente gordo em algum ponto $\hat{x} \in X$, então $\text{int}(X) \neq \emptyset$. Porém, se $\text{int}(X) \neq \emptyset$ não necessariamente X é um conjunto gordo, como mostra a Figura 1.8(b).

Muito mais frutífera é a comparação dos conjuntos gordos com os conjuntos X tais que $\overline{\text{int}(X)} = \overline{X}$, onde $\overline{X}$ denota o *fecho* de X. Mostremos que as duas definições são equivalentes.

Se $\overline{\text{int}(X)} = \overline{X}$, então para todo $\hat{x} \in X$ podemos tomar a sequência $\{y^k\}$, com $y^k \in \text{int}(X)$, tal que $\{y^k\} \to \hat{x}$. Pela definição de ponto interior, existe $\delta_k > 0$ tal que $B(y^k, \delta_k) \subset X$. Assim, dado $\varepsilon > 0$, só precisamos escolher k suficientemente grande de forma que y^k e δ_k sejam tais que $B(y^k, \delta_k) \subset B(\hat{x}, \varepsilon) \cap X$. Como $\hat{x}$ foi arbitrário, concluímos que X é gordo.

Reciprocamente, suponha que X é gordo. A inclusão $\overline{\text{int}(X)} \subset \overline{X}$ é imediata, pela Definição 1.12. Tomemos $\hat{x} \in \overline{X}$. Pela definição do fecho, existe $\{x^k\}$, $x^k \in X$, tal que $\{x^k\} \to \hat{x}$. Pela Definição 1.12, para todo x^k existe $y^k \in \text{int}(X)$, $y^k \in B(x^k, 1/k)$ tal que

$$\|y^k - \hat{x}\| \leq \|y^k - x^k\| + \|x^k - \hat{x}\|.$$

Portanto, $\{y^k\} \to \hat{x}$ e temos que $\hat{x} \in \overline{\text{int}(X)}$. Concluímos que $\overline{X} = \overline{\text{int}(X)}$.

Em análise funcional e topologia [Lim83], um conjunto X é dito *magro* se puder ser escrito como a união enumerável de conjuntos nunca densos, ou seja, conjuntos

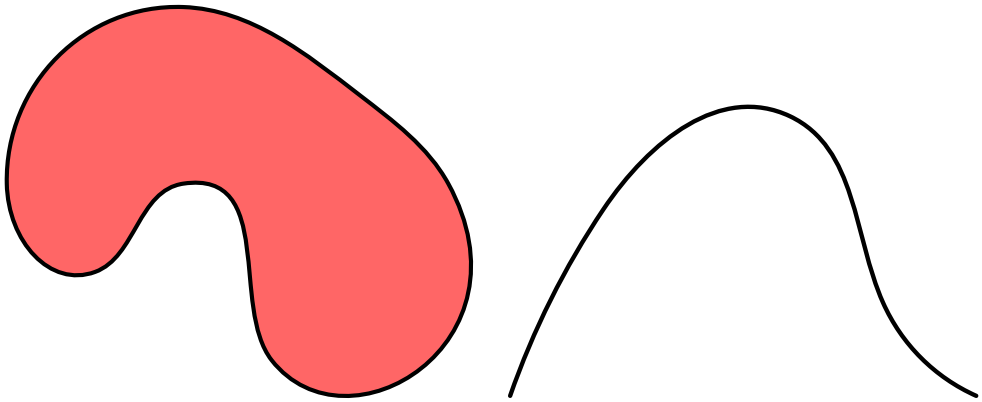


Figura 1.9: Conjunto gordo (à esquerda) e magro (à direita).

cujo interior do fecho é vazio:

$$X = \bigcup_{i \in \mathbb{N}} X_i, \quad \text{int}\left\{\overline{X_i}\right\} = \emptyset.$$

Embora o conceito se assemelhe com os tipos de conjunto com os quais estamos interessados neste trabalho, não usaremos essa definição formal de conjunto magro. Consideramos que um conjunto X é **magro** se ele não é gordo, ou se não é localmente gordo em uma determinada quantidade de pontos de X, no sentido da Definição 1.12. Com essa definição informal, temos a liberdade para afirmar que tanto o conjunto da direita na Figura 1.9 quanto o conjunto da Figura 1.8(b) podem ser considerados conjuntos magros.

1.4 Busca direta direcional com restrições lineares

Como discutimos na Seção 1.2, a classe de algoritmos denominada MADS engloba todos os algoritmos do tipo busca padrão [Tor97, ADJ03] para problemas sem derivadas. Entre eles, o algoritmo que trata especificamente de restrições lineares apresentado em [LT00].

Alguns anos antes da definição de MADS, em [KLT03] uma nova classe de algoritmos já havia sido proposta, denominada GSS (*Generating Set Search*). Tal classe também possui os algoritmos de busca padrão como um caso particular, porém permite a utilização do decréscimo suficiente (1.3). Além da busca padrão, engloba vários outros algoritmos para os casos irrestritos, com caixas e com restrições lineares. No caso particular de restrições lineares, GSS está fortemente relacionado com os algoritmos apresentados em [LST02], no qual os autores analisaram o caso em que as restrições do problema são gerais e possuem derivadas.

Suponha que temos o seguinte problema de otimização com restrições line-

ares:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & a_i^T x \leq b_i, \quad i = 1, \ldots, m, \end{array} \tag{1.16}$$

onde $a_i \in \mathbb{R}^n$, $i = 1, \ldots, m$, e $b \in \mathbb{R}^m$. Mantendo a notação deste capítulo, denotaremos por X o conjunto $\{x \in \mathbb{R}^n \mid a_i^T x \leq b_i, i = 1, \ldots, m\}$.

Todos os algoritmos baseados em direções de busca seguem, iterativamente, o princípio de que é necessário consultar um conjunto de direções que esteja de acordo com o conjunto viável. No caso irrestrito ou com restrições de caixa, isso é conseguido, por exemplo, com as direções consideradas na busca coordenada (veja o Algoritmo 1.1). Basta relembrarmos os resultados da Seção 1.1. A classe de algoritmos representada por MADS (Algoritmo 1.3) consegue lidar com conjuntos viáveis gerais gordos usando um conjunto denso de direções de busca.

De acordo com [KLT03], o nome GSS está associado à característica principal dessa nova classe: as direções de busca devem gerar o cone associado com os vetores normais às restrições "quase"-ativas no ponto corrente. Esse cone é denotado por

$$T(\hat{x}, \varepsilon) = \{d \in \mathbb{R}^n \mid a_i^T d \leq 0 \quad \forall i \in I(\hat{x}, \varepsilon)\},$$

onde $\hat{x}$ é o ponto corrente, ε representa a tolerância sobre as restrições e $I(\hat{x}, \varepsilon)$ é composto pelos índices $i \in \{1, \ldots, m\}$ para os quais a distância de $\hat{x}$ ao hiperplano $\{x \mid a_i^T x = b_i\}$ é menor ou igual a ε. A Figura 1.10, encontrada em [KLT06b], ilustra esse tipo de cone. $T(\hat{x}, \varepsilon)$ nada mais é do que o cone tangente [NW99] associado às restrições ε-ativas (ou quase-ativas).

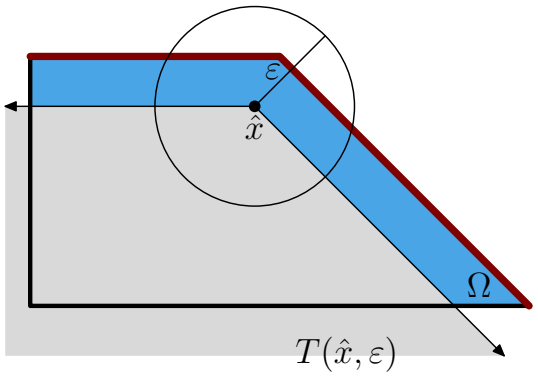


Figura 1.10: Restrições ε-ativas em $\hat{x}$ (linhas grossas) e o cone $T(\hat{x}, \varepsilon)$ associado.

Assim como certas condições eram necessárias nas direções de busca de MADS, na classe de algoritmos GSS desejamos que o conjunto de direções $D_k = G_k \cup H_k$ (definido para toda iteração k) satisfaça as seguintes condições:

1. Para todo k no qual $T(x^k, \varepsilon_k) \neq \{0\}$, o conjunto G_k gera "suficientemente bem" o cone $T(x^k, \varepsilon_k)$.

Algoritmo 1.5: GSS com decréscimo suficiente.

Entrada: $x^0 \in X$

Dados: $\alpha_{\min} > 0$, $\beta_{\max} \geq \beta_{\min} > 0$, $\alpha_0 > 0$ e $\varepsilon_{\max} > \beta_{\max}\alpha_{\min}$.

1. $k \leftarrow 0$
2. $\varepsilon_k = \min\{\varepsilon_{\max}, \beta_{\max}\alpha_k\}$

   Escolha um conjunto de direções de busca $D_k = G_k \cup H_k$ tal que as condições 1 e 2 sejam satisfeitas.
3. **se** existe $d^k \in D_k$ e correspondente $\tilde{\alpha}_k \in [0, \alpha_k]$ tais que $x^k + \tilde{\alpha}_k d^k \in X$ e

$$f(x^k + \tilde{\alpha}_k d^k) < f(x^k) - \rho(\alpha_k)$$

   então
   - $x^{k+1} = x^k + \tilde{\alpha}_k d^k$
   - $\alpha_{k+1} = \gamma_k \alpha_k$, para $\gamma_k \geq 1$
4. **senão**
   - $x^{k+1} = x^k$
   - $\alpha_{k+1} = \theta_k \alpha_k$, para $\theta_k \in (0, \theta_{\max}]$, com $\theta_{\max} < 1$
5. $k \leftarrow k + 1$ e volte para 2.

2. Existem $\beta_{\max} \geq \beta_{\min} > 0$, independentes de k, tais que, para todo k no qual $T(x^k, \varepsilon_k) \neq \{0\}$ temos

$$\beta_{\min} \leq \|d\| \leq \beta_{\max} \quad \forall d \in G_k.$$

O subconjunto de direções G_k é o mais importante para a teoria de convergência e H_k é formado por direções heurísticas, podendo ser vazio. Os conjuntos H_k e G_k têm papel equivalente aos passos de busca e consulta de MADS, respectivamente. Embora H_k seja irrelevante na teoria, observou-se na prática que direções obtidas por intuição dos usuários e/ou por heurísticas aumentam muito a eficiência dos algoritmos [ADJ03]. Esse é um dos pontos em que GSS se diferencia do algoritmo apresentado em [LST02].

Optamos por escrever a expressão "suficientemente bem" na condição 1 para evitar definições desnecessárias. Esse termo significa que existe uma constante, independente de k, que impede a degeneração das direções de G_k ao longo das iterações. Explicações mais detalhadas das condições 1 e 2 podem ser encontradas em [KLT03, KLT06b, CSV09b].

O Algoritmo 1.5 que apresentamos nesta seção é baseado no decréscimo suficiente da função objetivo e foi descrito em [KLT06b]. A deste momento, quando nos referimos ao algoritmo GSS, ou ao Algoritmo 1.5, fica implícita a referência à classe de algoritmos GSS. Outro algoritmo, que utiliza direções inteiras e decréscimo simples ao estilo de MADS, também foi apresentado em [KLT06b], com resultados de convergência análogos aos obtidos para o Algoritmo 1.5.

A simplicidade de descrição e a necessidade de um ponto inicial viável são características comuns de muitos algoritmos de direções de busca. Com um ponto viável em mãos, cada iteração do Algoritmo 1.5 visa garantir a viabilidade do novo ponto escolhido. Na implementação de GSS presente no algoritmo HOPSPACK [Pla09], por exemplo, caso o ponto inicial seja inviável, o passo primordial é a sua projeção no conjunto viável do problema.

A função $\rho(\cdot)$ deve possuir as mesmas características da função usada para definir o decréscimo suficiente (1.3) na Seção 1.1 deste capítulo, satisfazendo as mesmas condições.

A escolha de $\tilde{\alpha}_k$ merece um comentário. Para cada direção $d^i \in D_k$ associamos um tamanho de passo $\tilde{\alpha}_k^i$ tal que $x^k + \tilde{\alpha}_k^i d^i \in X$. O termo $\tilde{\alpha}_k$ é escolhido dentre $\{\tilde{\alpha}_k^i, i = 1, \ldots, |D_k|\}$ tal que $f(x^k + \tilde{\alpha}_k^i d^i)$ resulta em um decréscimo suficiente com respeito a $f(x^k)$. O Algoritmo 1.5 permite que $\tilde{\alpha}_k \in [0, \alpha_k]$, onde α_k é o limitante superior para o tamanho de passo. Porém, sempre que possível, o passo máximo deve ser dado, ou seja, $\tilde{\alpha}_k = \alpha_k$. Uma maneira simples de satisfazer essa condição é apresentada em [KLT06b].

Para analisar a convergência do Algoritmo 1.5 foi utilizada a seguinte medida de estacionariedade [KLT03, KLT06b]:

$$\chi(x) \doteq \max_{\substack{x+w \in X \\ \|w\| \leq 1}} \; -\nabla f(x)^T w,$$

onde X representa o conjunto viável de (1.16). Os autores afirmaram que, com essa medida, foram encontrados melhores resultados do que os obtidos com o gradiente contínuo projetado em trabalhos anteriores [LT00]. Além disso, é um resultado conhecido que $\chi(x) = 0$, para $x \in X$, se e somente se x é um ponto KKT do problema (1.16) [CGST96].

Apresentamos a seguir os dois resultados mais importantes de convergência de GSS. O primeiro teorema (encontrado em [KLT06b, Teorema 6.3]) estabelece uma relação entre a projeção de $\nabla f(x)$ em $T(x, \varepsilon)$ e o limitante do tamanho do passo α_k.

1.13 Teorema ([KLT06b, Teorema 6.3])
Suponha que ∇f é Lipschitz contínuo em X, com constante de Lipschitz M. Se k é uma iteração mal sucedida do Algoritmo 1.5 (o passo 4 é executado) e $\varepsilon_k = \beta_{\max}\alpha_k$, então

$$\left\| P_{T(x^k, \varepsilon_k)}(-\nabla f(x^k)) \right\| \leq \frac{1}{\kappa} \left(M \beta_{\max} \alpha_k + \frac{\rho(\alpha_k)}{\beta_{\min} \alpha_k} \right),$$

onde $P_{T(x^k, \varepsilon_k)}$ representa a projeção ortogonal no cone $T(x^k, \varepsilon_k)$ e as demais constantes estão associadas à hipótese e às condições 1 e 2.

Esse resultado é importante no presente trabalho, porque será utilizado na convergência do algoritmo proposto no Capítulo 4. Como pretendemos usar GSS como um algoritmo interno (para resolver subproblemas) desejamos boas propriedades após um número finito de iterações. Isso significa terminar GSS caso $\alpha_k \leq \alpha_{\min}$, com $\alpha_{\min} >$

0. Em [Bue11, Capítulo 6], como consequência direta do Teorema 1.13, temos que se k é uma iteração mal sucedida e $\bar{\varepsilon}_k$ é definido por

$$\bar{\varepsilon}_k = \max\left\{\frac{1}{\kappa}\left(M\beta_{\max}\alpha_k + \frac{\rho(\alpha_k)}{\beta_{\min}\alpha_k}\right), \varepsilon_k\|A\|_F\right\},$$

com todos os termos dados no teorema anterior e $\|A\|_F$ representando a norma de Frobenius da matriz de restrições lineares, então x^k é um ponto $\bar{\varepsilon}_k$-KKT do problema (1.16). De acordo com [AHM11], um ponto $x \in \mathbb{R}^n$ é ε-KKT do problema (1.16) se existe $\mu \in \mathbb{R}^m_+$ tal que

$$\begin{aligned}\left\|\nabla f(x) + \sum_{i=1}^{m}\mu_i a_i\right\| &\leq \varepsilon \\ a_i^T x - b_i &\leq \varepsilon \quad i = 1, \ldots, m \\ a_i^T x - b_i < -\varepsilon \;\Rightarrow\; \mu_i &= 0 \quad i = 1, \ldots, m,\end{aligned}$$

para qualquer norma. Informalmente falando, x é ε-KKT se estiver a caminho de ser um ponto KKT, caso $\varepsilon \to 0$.

O segundo teorema ([KLT06b, Teorema 6.4]) garante que, se existir uma subsequência K tal que $\lim_{k\in K}\alpha_k = 0$, então o ponto limite desta subsequência é um ponto KKT de (1.16). Esse resultado está de acordo com as conclusões do parágrafo anterior e com [AHM11], pois o ponto limite x^* seria o limite de pontos $\bar{\varepsilon}_k$-KKT, com $\bar{\varepsilon}_k \to 0$. Estratégias que garantem a existência da subsequência K são chamadas de estratégias de globalização e foram discutidas na Seção 1.1. As mais comuns são: decréscimo suficiente e decréscimo simples em uma malha.

1.14 Teorema ([KLT06b, Teorema 6.4])
Suponha que f é limitada no conjunto $\{x \in X |\ f(x) \leq f(x^0)\}$ e que ∇f é Lipschitz contínuo em X, com constante de Lipschitz M. Se k é uma iteração mal sucedida do Algoritmo 1.5 (o passo 4 é executado) e $\varepsilon_k = \beta_{\max}\alpha_k$, então

$$\chi(x^k) \leq \kappa_1\beta_{\max}\alpha_k + \kappa_2\frac{\rho(\alpha_k)}{\alpha_k},$$

onde κ_1 e κ_2 são constantes associadas com as condições impostas sobre o algoritmo e com limitantes de f.

Podemos perceber que os resultados dos dois teoremas estão relacionados com o tamanho de α_k. Em cada iteração mal sucedida (passo 4) de GSS, o valor de α_k é reduzido. Como mencionamos no estudo do Algoritmo 1.1, os passos mal sucedidos indicam que estamos nos aproximando de um minimizador (ou ponto estacionário) do problema. Dessa forma, é natural que o critério de parada de GSS esteja relacionado com o tamanho de α_k, digamos

$$\alpha_k \leq \alpha_{\min}.$$

É importante ressaltarmos isso, pois, nos métodos que aplicam GSS como um algoritmo interno, é necessário que $\alpha_{\min} \to 0$ gradativamente. Neste trabalho descrevemos 2 algoritmos que utilizam GSS como algoritmo interno: Lagrangianos Aumentados na Seção 1.5 e Restauração Inexata no Capítulo 4.

1.5 Lagrangianos Aumentados

Métodos baseados em Lagrangianos Aumentados são métodos de penalidade muito usados na área de programação não linear. Uma explicação detalhada e intuitiva pode ser encontrada em [Mar06] e sobre métodos de penalidade em geral pode-se consultar [Ber99, NW99].

O princípio básico de um método de penalidade é que existem algoritmos comprovadamente eficientes para resolver o problema de minimizar uma função sem restrições ou com restrições simples. Dessa forma, para resolver um problema com restrições complexas, basta colocá-las junto com a função objetivo e penalizá-las de modo que os pontos inviáveis sejam indesejáveis. O novo problema, agora com uma função objetivo mais complicada e com restrições simples, será resolvido com algoritmos eficientes. O modo como a nova função objetivo é feita define o tipo de método de penalidade.

Suponha que o seguinte problema deve ser resolvido:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & g(x) \leq 0 \text{ e } h(x) = 0 \\ & \underline{g}(x) \leq 0 \text{ e } \underline{h}(x) = 0, \end{array} \tag{1.17}$$

com $f : \mathbb{R}^n \to \mathbb{R}$. As restrições $g : \mathbb{R}^n \to \mathbb{R}^m$ e $h : \mathbb{R}^n \to \mathbb{R}^p$ são consideradas difíceis e $\underline{g} : \mathbb{R}^n \to \mathbb{R}^{\underline{m}}$ e $\underline{h} : \mathbb{R}^n \to \mathbb{R}^{\underline{p}}$ são consideradas fáceis, ou simples. Se houvessem restrições de caixa ($l \leq x \leq u$, com $l, u \in \mathbb{R}^n$), estas estariam inclusas nas restrições fáceis.

Quando consideramos Lagrangianos Aumentados, podemos tomar a função de Powell–Hestenes–Rockafellar como função de penalidade [Mar06, ABMS07]:

$$L_\rho(x, \lambda, \mu) = f(x) + \frac{\rho}{2} \sum_{i=1}^{m} \max\left\{0, g(x) + \frac{\mu_i}{\rho}\right\}^2 + \frac{\rho}{2} \sum_{i=1}^{p} \left(h(x) + \frac{\lambda_i}{\rho}\right)^2, \tag{1.18}$$

onde ρ é o parâmetro de penalização e λ_i, $i = 1, \ldots, p$, e μ_i, $i = 1, \ldots, m$, são estimativas para os multiplicadores de Lagrange das restrições difíceis de igualdade e desigualdade, respectivamente. Usando a função (1.18), no método de Lagrangianos Aumentados resolve-se iterativamente o problema:

$$\begin{array}{ll} \min\limits_{x} & L_\rho(x, \lambda, \mu) \\ \text{s. a} & \underline{g}(x) \leq 0 \\ & \underline{h}(x) = 0. \end{array} \tag{1.19}$$

Em cada iteração, os multiplicadores de Lagrange são atualizados e, de acordo com a qualidade da solução obtida em (1.19), o parâmetro de penalidade é mantido ou aumentado.

Diversos são os motivos pelos quais a abordagem por Lagrangianos Aumentados é recomendada, mas a maior vantagem é a possibilidade de trabalhar com restrições gerais utilizando métodos conhecidos. Uma lista de outras razões foi apresentada em [ABMS07]. Em vista disso, tais métodos mantém-se altamente eficientes e competitivos, disputando de igual para igual com os métodos que estão mais em evidência, como aqueles baseados em pontos interiores [WB06], por exemplo.

Na otimização sem derivadas o cenário não foi diferente. Diante da dificuldade em lidar com restrições gerais nos problemas sem derivadas, diversos trabalhos apresentaram abordagens baseadas em métodos de penalidade, como a penalidade externa [LLS10] e, em particular, Lagrangianos Aumentados [LT02, KLT06a, GK10, LT10, DEMP11]. Em [LT02] os autores estenderam os resultados de convergência do algoritmo LANCELOT [CGT91] e em [LT10, DEMP11] os resultados de convergência do algoritmo Algencan [ABMS07] foram recuperados. Nesta seção discutimos os resultados obtidos nas extensões para o algoritmo Algencan, com particular ênfase em [LT10], por utilizar estratégias semelhantes às utilizadas nos algoritmos deste trabalho.

Os resultados apresentados em [DEMP11] consideraram quaisquer tipo de restrições $\underline{g}(x)$ e $\underline{h}(x)$ no problema (1.19). Em virtude disso, sua aplicação é mais ampla, desde que hajam algoritmos sem derivadas que consigam lidar eficientemente com as restrições "fáceis". Por outro lado, os resultados apresentados em [LT10] contemplam apenas restrições lineares nos subproblemas (1.19), mas estão associados com o algoritmo GSS [KLT03, KLT06b] discutido na Seção 1.4, para o qual já existem implementações eficientes.

O Algoritmo 1.6 é uma descrição simplificada do algoritmo de Lagrangianos Aumentados Algencan para problemas com derivadas, que foi estendido em [LT10] e [DEMP11]. Optamos por apresentar primeiramente os resultados mais simples para o caso em que as restrições de (1.19) são apenas caixas, seguindo o mesmo raciocínio de [DEMP11]. Com isso, conseguimos expressar qual o espírito por trás das adaptações sem derivadas para Lagrangianos Aumentados. Em seguida, mostramos os resultados para restrições lineares, encontrados em [LT10].

A teoria desenvolvida em [ABMS07] para o Algoritmo 1.6 utiliza hipóteses mais fracas que as normalmente usadas para Lagrangianos Aumentados. Em vista disso, consegue-se englobar um maior conjunto de problemas. Grande parte do enfraquecimento das hipóteses está relacionado com as condições de qualificação das restrições. Enquanto LANCELOT supõe que os pontos limites gerados pelo algoritmo satisfazem a Condição de Qualificação de Independência Linear (LICQ – *Linear Independence Constraint Qualification*), o Algoritmo 1.6 utiliza a Condição da Dependência Linear Positiva Constante (CPLD – *Constant Positive Linear Dependence*) [AMS05]. Recentemente, utilizou-se condições de qualificação ainda mais fracas do que a CPLD para demonstrar convergência do Algoritmo 1.6 a pontos KKT. Tais condições são RCPLD (*Relaxed Constant Positive Linear Dependence*) [AHSS11a] e CPG (*Constant Positive*

Algoritmo 1.6: Algoritmo baseado em Lagrangianos Aumentados [ABMS07].

Entrada: Seja $x^0 \in \mathbb{R}^n$. Sejam $\tau \in [0,1)$, $\gamma > 1$, $\rho_1 > 0$, $-\infty < \overline{\lambda}_{\min} < \overline{\lambda}^1 < \overline{\lambda}_{\max} < \infty$, $0 \leq \overline{\mu}^1 < \overline{\mu}_{\max} < \infty$ e $\{\varepsilon_k\}$, $\varepsilon_k \geq 0$, uma sequência de parâmetros de tolerância tais que $\lim_{k\to\infty} \varepsilon_k = 0$.

1. $k \leftarrow 1$

2. **Resolução aproximada do subproblema**
Encontre $x^k \in \mathbb{R}^n$ para o problema (1.19) tais que existem $v^k \in \mathbb{R}^{\underline{p}}$ e $u^k \in \mathbb{R}^{\underline{m}}_+$ que satisfaçam

$$\left\| \nabla L_{\rho_k}(x^k, \overline{\lambda}^k, \overline{\mu}^k) + \sum_{i=1}^{\underline{p}} v_i^k \nabla \underline{h}(x^k) + \sum_{i=1}^{\underline{m}} u_i^k \nabla \underline{g}(x^k) \right\| \leq \varepsilon_k \tag{1.20}$$

$$u_i^k \geq 0 \quad \text{e} \quad \underline{g}_i(x^k) \leq \varepsilon_k \quad i = 1, \ldots, \underline{m} \tag{1.21}$$

$$\underline{g}_i(x^k) < -\varepsilon_k \;\Rightarrow\; u_i^k = 0 \quad i = 1, \ldots, \underline{m} \tag{1.22}$$

$$\|\underline{h}(x^k)\| \leq \varepsilon_k \tag{1.23}$$

(x^k também é chamado de ponto ε_k-KKT [AHM11].)

3. **Atualização dos multiplicadores de Lagrange**
$\lambda^{k+1} = \overline{\lambda}^k + \rho_k h(x^k)$
Calcule $\overline{\lambda}^{k+1}$, projetando λ^{k+1} em $[\overline{\lambda}_{\min}, \overline{\lambda}_{\max}]$.
$\sigma^k = \max\{g(x^k), -\overline{\mu}^k/\rho_k\}$
$\mu^{k+1} = \max\{0, \overline{\mu}^k + \rho_k g(x^k)\}$
Calcule $\overline{\mu}^{k+1}$ projetando μ^{k+1} em $[0, \overline{\mu}_{\max}]$.

4. **se** $k > 1$ *e* $\max\{\|h(x^k)\|_\infty, \|\sigma^k\|_\infty\} > \tau \max\{\|h(x^{k-1})\|_\infty, \|\sigma^{k-1}\|_\infty\}$ **então**
   $\rho_{k+1} = \gamma \rho_k$
   senão
   $\rho_{k+1} = \rho_k$

5. $k \leftarrow k + 1$ e volte para 2.

Generator) [AHSS11b].

Supondo que todas as funções envolvidas são continuamente diferenciáveis, a seguir apresentamos os dois teoremas principais de convergência do Algoritmo 1.6. Todas as extensões do algoritmo para o caso sem derivadas têm por objetivo principal recuperar esses dois importantes resultados.

1.15 Teorema ([ABMS07, Teorema 4.1])
Seja x^* o ponto de acumulação da sequência $\{x^k\}_{k\in\mathbb{N}}$ gerada pelo Algoritmo 1.6. Se a sequência dos parâmetros de penalidade $\{\rho_k\}$ é limitada, então x^* é viável de (1.17).

Caso contrário, ao menos uma das seguintes possibilidades ocorre:

- x^* *é um ponto KKT do problema*

$$\begin{array}{ll} \min & \dfrac{1}{2}\left[\displaystyle\sum_{i=1}^{p} h_i(x)^2 + \sum_{i=1}^{m} \max\{0, g_i(x)\}^2\right] \\ s.\ a & \underline{g}(x) \leq 0 \\ & \underline{h}(x) = 0. \end{array}$$

- x^* *não satisfaz a condição de qualificação CPLD associada às restrições (fáceis)* $\underline{g}(x)$ *e* $\underline{h}(x)$.

1.16 Teorema (Convergência a pontos KKT)
Seja $\{x^k\}$ *uma sequência de pontos gerada pelo Algoritmo 1.6. Suponha que* x^* *é um ponto de acumulação viável do problema* (1.17) *e satisfaz a condição de qualificação CPLD. Então,* x^* *é um ponto KKT do problema* (1.17).

Demonstração:

Veja [ABMS07, Teorema 4.2]. ∎

A primeira extensão que discutimos a seguir foi considerada em [Ped09, DEMP11] para o caso em que o conjunto $\{x \in \mathbb{R}^n \mid \underline{g}(x) \leq 0 \text{ e } \underline{h}(x) = 0\}$ é definido apenas pelas restrições de caixa ($l \leq x \leq u$). Em outras palavras, todas as outras restrições do problema são consideradas "difíceis", acopladas à função (1.18) e penalizadas.

O Algoritmo 1.7 se diferencia do Algoritmo 1.6 apenas pelo passo 2. Isso ocorre devido às condições (1.20)–(1.23), que devem ser substituídas por condições que sejam satisfeitas por algoritmos que não utilizam derivadas. Pontos que satisfazem estas condições também são chamados de pontos ε-KKT [AHM11]. Qualquer semelhança entre a condição (1.24) e o Algoritmo 1.1, baseado em busca coordenada, não é mera coincidência. De fato, se a busca coordenada for utilizada para resolver o subproblema (1.19), então temos que a condição (1.24) será satisfeita. Essa observação demonstra a boa definição do algoritmo.

Supondo que ∇f, $\nabla g \in \mathbb{R}^{n\times m}$ e $\nabla h \in \mathbb{R}^{n\times p}$ são Lipschitz contínuas na caixa, em [DEMP11] demonstrou-se que o Algoritmo 1.7 é um caso particular do Algoritmo 1.6. Isso significa que, se a condição (1.24) é satisfeita, então (1.20)–(1.23) também o são. Consequentemente, os resultados conseguidos nos teoremas 1.15 e 1.16 continuam válidos para o Algoritmo 1.7.

Porém, um inconveniente conhecido dos métodos do tipo penalidade é que, por acoplar todas as restrições em uma única função, além da função objetivo tornar-se mais difícil, perde-se a oportunidade de usar características específicas que as restrições podem ter, como a linearidade, por exemplo. Por essa razão, embora o Algoritmo 1.7 se destaque pela simplicidade e abrangência, perde em eficiência. O objetivo é que tenhamos o maior número possível de restrições em (1.19) e deixemos na função $L_\rho(x, \lambda, \mu)$

Algoritmo 1.7: Lagrangianos Aumentados sem derivadas para caixas.

Entrada: Seja $x^0 \in \mathbb{R}^n$. Sejam $\tau \in [0,1)$, $\gamma > 1$, $\rho_1 > 0$, $-\infty < \overline{\lambda}_{\min} < \overline{\lambda}^1 < \overline{\lambda}_{\max} < \infty$, $0 \leq \overline{\mu}^1 < \overline{\mu}_{\max} < \infty$ e $\{\varepsilon_k\}$, $\varepsilon_k \geq 0$, uma sequência de parâmetros de tolerância tais que $\lim_{k\to\infty} \varepsilon_k = 0$.

Considere todos os passos do Algoritmo 1.6, com exceção do passo 2, que é substituído pelo seguinte passo.

2. Resolução aproximada do subproblema
Escolha uma tolerância $\alpha_k \leq (u_i - l_i)/2$, $i = 1, \ldots, n$, tal que

$$\alpha_k \leq \min\left\{\frac{\varepsilon_k}{\rho_k}, \varepsilon_k\right\}.$$

Encontre $l \leq x^k \leq u$ tal que, para $i = 1, \ldots, n$

$$l \leq x^k \pm \alpha_k e_i \leq u \;\Rightarrow\; L_{\rho_k}(x^k \pm \alpha_k e_i, \overline{\lambda}^k, \overline{\mu}^k) \geq L_{\rho_k}(x^k, \overline{\lambda}^k, \overline{\mu}^k), \tag{1.24}$$

onde e_i são as direções coordenadas.

somente as restrições realmente difíceis. Um algoritmo que considera restrições gerais nos subproblemas (1.19), também foi apresentado em [DEMP11], mas vamos discutir um outro algoritmo, apresentado em [LT10], que considera o caso em que as restrições $\underline{g}(x)$ e $\underline{h}(x)$ são lineares.

A ideia é muito semelhante à que utilizamos no algoritmo de Lagrangianos Aumentados sem derivadas com caixas. A diferença é que as condições (1.20)–(1.23) são simplesmente reescritas considerando-se apenas restrições lineares, ou seja, tomando $\underline{h}_i(x) = a_i^T x - b_i$ e $\underline{g}_i(x) = c_i^T x - d_i$:

$$\left\| \nabla L_{\rho_k}(x^k, \overline{\lambda}^k, \overline{\mu}^k) + \sum_{i=1}^{\underline{p}} v_i^k a_i + \sum_{i=1}^{\underline{m}} u_i^k c_i \right\|_2 \leq \varepsilon_k \tag{1.25}$$

$$u_i^k \geq 0 \quad \text{e} \quad c_i^T x^k - d_i \leq \varepsilon_k \quad i = 1, \ldots, \underline{m} \tag{1.26}$$

$$c_i^T x^k - d_i < -\varepsilon_k \;\Rightarrow\; u_i^k = 0 \quad i = 1, \ldots, \underline{m} \tag{1.27}$$

$$\left(\sum_{i=1}^{\underline{p}} (a_i^T x^k - b_i)^2 \right)^{1/2} \leq \varepsilon_k, \tag{1.28}$$

onde a norma Euclidiana foi usada para acompanhar a descrição de [LT10], u^k e v^k são tais como em (1.20)–(1.23), $a_i \in \mathbb{R}^n$, $i = 1, \ldots, \underline{p}$, $b \in \mathbb{R}^{\underline{p}}$, $c_i \in \mathbb{R}^n$, $i = 1, \ldots, \underline{m}$, e $d \in \mathbb{R}^{\underline{m}}$. Em vista disso, o algoritmo para o caso linear é idêntico ao Algoritmo 1.6.

Da mesma forma que a condição (1.24) estava associada à busca coordenada (Algoritmo 1.1, por exemplo), as condições (1.25)–(1.28) estão associadas à busca

direta direcional com restrições lineares apresentada no Algoritmo 1.5 (GSS) da Seção 1.4. Essa relação é feita através da proposição a seguir, cuja demonstração pode ser encontrada em [LT10, Proposição 5.3].

1.17 Proposição
Seja $\{x^j\}$ a sequência gerada pelo Algoritmo 1.5 (GSS) aplicado ao subproblema (1.19)*, cujas restrições "fáceis" são apenas lineares. Suponha que $L_\rho(x, \overline{\lambda}, \overline{\mu})$ é Lipschitz contínua em x, com constante $M(\overline{\lambda}, \overline{\mu}, \rho)$, no conjunto viável de* (1.19)*, onde $\overline{\lambda}$ e $\overline{\mu}$ são estimativas limitadas para os multiplicadores de Lagrange. Se j é uma iteração malsucedida (redução de α_j), então existem $u \in \mathbb{R}^{\underline{m}}_+$ e $v \in \mathbb{R}^{\underline{p}}$ tais que*

$$\begin{aligned} \left\| \nabla L_\rho(x^j, \overline{\lambda}, \overline{\mu}) + \sum_{i=1}^{\underline{p}} v_i a_i + \sum_{i=1}^{\underline{m}} u_i c_i \right\|_2 &\leq C\alpha_j \\ u_i \geq 0 \quad e \quad c_i^T x^j - d_i \leq 0 \quad & i = 1, \ldots, \underline{m} \\ c_i^T x^j - d_i < -\alpha_j \|c_i\|_2 \;\Rightarrow\; u_i = 0 \quad & i = 1, \ldots, \underline{m} \\ a_i^T x^j - b_i = 0 \quad & i = 1, \ldots, \underline{p}, \end{aligned}$$

onde α_j é o tamanho do passo e C é uma constante que depende da constante de $M(\overline{\lambda}, \overline{\mu}, \rho)$, de limitantes para a função f e da forma como as direções de busca são feitas.

O resultado principal do Teorema 1.17 é que, se GSS é aplicado na resolução de um subproblema de Lagrangianos Aumentados, e é interrompido após uma iteração j malsucedida, devolve um ponto $\bar{\varepsilon}$-KKT (veja a Seção 1.4), onde $\bar{\varepsilon}$ é dado por

$$\bar{\varepsilon} = \max\left\{\alpha_j C, \alpha_j \min\{\|a_i\|_2\}\right\}.$$

Logo, as condições (1.25)–(1.28) serão satisfeitas para $\bar{\varepsilon}$.

Como explicado na Seção 1.4, o principal critério de parada do algoritmo GSS (Algoritmo 1.5) é o tamanho do passo:

$$\alpha_j \leq \alpha_{\min}.$$

Desta forma, se em cada iteração k do algoritmo de Lagrangianos Aumentados utilizamos um critério de parada $\alpha_{\min}^{(k)}$ para GSS, tal que $\alpha_{\min}^{(k)} \to 0$ quando $k \to \infty$, então também conseguimos gerar uma sequência $\{\varepsilon_k\} \to 0$, utilizando hipóteses razoáveis para f, g e h. Com essa observação, demonstra-se que os resultados de convergência dos teoremas 1.15 e 1.16 também são válidos para o caso sem derivadas com restrições lineares, utilizando GSS na resolução dos subproblemas de Lagrangianos Aumentados.

1.6 Região de confiança

Em otimização clássica sem restrições, um algoritmo de região de confiança é caracterizado pela sua maneira diferenciada de escolher as direções de decréscimo da

função objetivo [MS95]. A maior parte dos algoritmos conhecidos utiliza algum tipo de modelo da função objetivo. Porém, com região de confiança a direção é aquela que minimiza o modelo sujeito à uma região na qual acreditamos que este é um bom representante da função objetivo original. Algoritmos baseados em regiões de confiança são bastante conhecidos no âmbito de otimização irrestrita [NW99] e sua extensão para o caso restrito está representada, por exemplo, por algumas versões dos conhecidos métodos de Programação Quadrática Sequencial (PQS) [NW99, MS95].

Suponha que desejamos resolver o problema irrestrito definido por

$$\min \ f(x) \quad \text{s. a} \quad x \in \mathbb{R}^n.$$

Estudaremos tal problema, pois os resultados e algoritmos mais importantes estão relacionados a ele. A teoria de convergência dos problemas com restrições ainda não foi desenvolvida para o caso sem derivadas.

Quando trabalhamos com funções gerais, a utilização de modelos mais simples de tais funções é uma técnica bastante comum. Como um exemplo, temos a busca linear clássica, que modela a função f no ponto $\hat{x}$ por sua aproximação de Taylor de primeira ordem

$$f(x) \approx f(\hat{x}) + \nabla f(\hat{x})^T (x - \hat{x})$$

para encontrar uma direção de decréscimo [NW99]. A esses modelos aplicamos as técnicas de minimização que já encontram-se bem estabelecidas, como a minimização unidimensional, no caso da busca linear. Infelizmente, além das funções lineares, raros são os casos em que o minimizador de um modelo e o da própria função coincidem. Para conseguirmos encontrar os minimizadores verdadeiros, precisamos recalcular os modelos iterativamente e definir quão bons representantes eles são da função original.

Nos métodos de região de confiança para o caso irrestrito, geralmente modelamos a função f por sua aproximação de Taylor de segunda ordem no ponto corrente $\hat{x}$:

$$f(x) \approx \tilde{f}(x) = f(\hat{x}) + \nabla f(\hat{x})^T (x - \hat{x}) + \frac{1}{2}(x - \hat{x})^T \nabla^2 f(\hat{x})(x - \hat{x}).$$

Dessa forma, resolve-se iterativamente o seguinte problema:

$$\min \ \tilde{f}(x) \quad \text{s. a} \quad \|x - \hat{x}\| \leq \Delta, \tag{1.29}$$

onde Δ é o raio da região de confiança, cujo centro encontra-se no ponto corrente $\hat{x}$, para o qual acreditamos que $\tilde{f}$ é uma boa representante de f.

Com a impossibilidade de utilizarmos derivadas em alguns problemas, algoritmos que se baseavam em modelos desse tipo tornaram-se impraticáveis. Poderia-se dizer que o gradiente e a Hessiana de f podem ser calculados por técnicas de diferenças finitas ou de diferenciação automática. A primeira é ineficiente se o custo computacional para avaliar f é muito alto, o que é uma situação frequente em problemas sem derivadas. A segunda não pode ser feita para casos em que o cálculo de f é dado por uma caixa preta, ou seja, o código computacional que descreve f não está disponível. Essas justificativas foram discutidas em [CST97a, CST97b].

Ao tratarmos de funções cujo custo de avaliação computacional é elevado, tentamos aproveitar ao máximo as informações verdadeiras conseguidas até o momento. Essa é a maior vantagem dos modelos: utilizamos toda informação disponível no momento para construí-los e, durante sua minimização, não são necessárias avaliações da função original. Consequentemente, o custo de resolver tal subproblema de minimização é irrelevante quando comparado ao custo de uma avaliação da função. Desde a década de 60, M. J. D. Powell já desenvolvia métodos de otimização que não utilizavam derivadas [CST97a]. Porém, em [Pow94] começou-se a estudar as propriedades da interpolação polinomial na geração de modelos lineares e quadráticos. Atualmente existem diversos algoritmos eficientes: DFO [CST97b, CSV09a], UOBYQA [Pow02], NEWUOA [Pow06] e BOBYQA [Pow09]. Apesar dos trabalhos e algoritmos de Powell serem uma referência para essa técnica, um dos primeiros estudos sobre tais tipos de modelos são devidos a Winfield [Win73].

Em [CST97a] e [CST97b] os autores apresentaram um algoritmo, cuja implementação foi chamada DFO, e sua respectiva teoria de convergência. O algoritmo teórico englobava diversos métodos de região de confiança para minimização irrestrita sem derivadas desenvolvidos até então. Todos utilizavam interpolação para gerar os modelos (lineares e/ou quadráticos) de f. Posteriormente, em [CSV09b] o método tornou-se mais geral, permitindo a utilização de modelos quadráticos através de regressão. Atualmente, a melhor fonte de referência dos algoritmos de região de confiança para problemas sem derivadas encontra-se em [CSV09a], mas nosso estudo nesta seção segue quase totalmente o trabalho [CST97b], que apresenta uma simples descrição de tais algoritmos e as principais ideias da teoria de convergência, além de ser a base do algoritmo DFO.

O caso mais simples de interpolação que podemos considerar é com uma função de uma variável $f : \mathbb{R} \to \mathbb{R}$. É fácil ver que por dois pontos distintos passa uma única reta e por três, também distintos, uma única parábola. Se dois desses três pontos fossem iguais, então haveria infinitas parábolas que interpolariam os três pontos. Para entender esse "requisito" sobre os pontos, suponha que temos três pontos $y^1, y^2, y^3 \in \mathbb{R}$ e desejamos encontrar uma parábola $m(x) : \mathbb{R} \to \mathbb{R}$ tal que

$$m(y^i) = f(y^i) \quad i = 1, 2, 3. \tag{1.30}$$

Usando a base dos monômios $\phi = \{1, x, x^2\}$ para o espaço dos polinômios de grau até 2 com uma variável, temos que

$$m(x) = a_1 + a_2 x + a_3 x^2, \tag{1.31}$$

onde a_1, a_2 e a_3 são os coeficientes que desejamos determinar. Assim, para encontrar o modelo $m(x)$ desejado, usando (1.30) e (1.31) resolvemos o seguinte sistema linear:

$$\begin{bmatrix} 1 & y^1 & (y^1)^2 \\ 1 & y^2 & (y^2)^2 \\ 1 & y^3 & (y^3)^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} f(y^1) \\ f(y^2) \\ f(y^3) \end{bmatrix}. \tag{1.32}$$

Denotaremos por $M(\phi, \{y^1, y^2, y^3\})$ a matriz do sistema (1.32), para lembrar que ela depende da base ϕ para os polinômios e também do conjunto de pontos $\{y^1, y^2, y^3\}$ que usamos para a interpolação. Claramente verificamos que, se há dois pontos iguais, então $M(\phi, \{y^1, y^2, y^3\})$ é singular e (1.32) admite infinitas soluções. Este "requisito" sobre os pontos a serem interpolados pode ser estendido para $\mathbb{R}^n$ sob o conceito de **conjunto equilibrado** (tradução livre de *poised* [CST97a]).

Generalizando, seja $Y = \{y^i\}_{i=1}^p$ o conjunto de pontos a serem interpolados, para os quais conhecemos $f(y^i)$. O modelo de f é dado pela seguinte função:

$$m(x) = f(\hat{x}) + g^T(x - \hat{x}) + \frac{1}{2}(x - \hat{x})^T H(x - \hat{x})$$

ou, definindo $s = x - \hat{x}$,

$$m(\hat{x} + s) = f(\hat{x}) + g^T s + \frac{1}{2} s^T H s, \tag{1.33}$$

onde $\hat{x} \in Y$ foi escolhido adequadamente, $g \in \mathbb{R}^n$ e $H \in \mathbb{R}^{n \times n}$. Se as derivadas estivessem disponíveis, fazendo $g = \nabla f(\hat{x})$ e $H = \nabla^2 f(\hat{x})$ teríamos a aproximação de Taylor de segunda ordem em $\hat{x}$. Assim como fizemos em uma dimensão, o modelo definido em (1.33) deve satisfazer

$$m(y) = f(y) \quad \forall y \in Y \tag{1.34}$$

e, para que o modelo $m(x)$ seja completamente definido pelo sistema (1.34), devemos também ter que a cardinalidade de Y, $|Y|$, seja igual a

$$p = \frac{(n+1)n}{2} + n + 1 = \frac{1}{2}(n+1)(n+2), \tag{1.35}$$

no caso quadrático. A primeira parte da soma vem de H, que é simétrica, a segunda vem de g e a terceira parte vem do termo constante. Embora o termo constante não apareça em (1.33), ele está implícito no termo $f(\hat{x})$, dado que $\hat{x} \in Y$. Para encontrar o modelo usual, onde cada monômio está multiplicado por uma constante (como em (1.31)), basta expandir o modelo (1.33), substituir s por $x - \hat{x}$ e juntar os termos com mesmos monômios.

De acordo com [CST97b], o fato do requisito (1.35) ser satisfeito não implica que o modelo $m(x)$ existirá e nem que estará unicamente definido. Tão pouco garantir que os pontos de Y são distintos. Se $n = 2$, por exemplo, e os pontos de Y estiverem todos sobre uma reta ou sobre uma circunferência, poderão haver inconvenientes. O Exemplo 1.18 discute esse caso, mencionado em [CST97a] e [CST97b].

1.18 Exemplo

Suponha que a função $f : \mathbb{R}^2 \to \mathbb{R}$ será interpolada utilizando-se o conjunto

$$Y = \{(x_i, y_i)\}_{i=1}^6.$$

Suponha ainda que os pontos de Y pertencem à reta $r : ax - by - c = 0$, com $a \neq 0$, ou seja, $x_i = \frac{c}{a} + \frac{b}{a}y_i$, para $i = 1, \ldots, 6$. Em duas dimensões ($n = 2$), podemos considerar $\phi = \{1, x, y, xy, x^2, y^2\}$ como base para os polinômios de grau menor ou igual a 2, obtendo, dessa forma,

$$M(\phi, Y) = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 & (x_1)^2 & (y_1)^2 \\ & & & \vdots & & \\ 1 & x_6 & y_6 & x_6y_6 & (x_6)^2 & (y_6)^2 \end{bmatrix}.$$

Logo, usando a equação da reta,

$$M(\phi, Y) = \begin{bmatrix} 1 & \frac{c}{a} + \frac{b}{a}y_1 & y_1 & \frac{c}{a}y_1 + \frac{b}{a}(y_1)^2 & \left(\frac{c}{a}\right)^2 + 2\frac{bc}{a^2}y_1 + \left(\frac{b}{a}\right)^2(y_1)^2 & (y_1)^2 \\ & & & \vdots & & \\ 1 & \frac{c}{a} + \frac{b}{a}y_6 & y_6 & \frac{c}{a}y_6 + \frac{b}{a}(y_6)^2 & \left(\frac{c}{a}\right)^2 + 2\frac{bc}{a^2}y_6 + \left(\frac{b}{a}\right)^2(y_6)^2 & (y_6)^2 \end{bmatrix}.$$

Podemos ver que a segunda coluna é uma combinação linear da primeira e da terceira colunas, a quarta coluna, combinação da terceira e da sexta colunas e a quinta coluna, uma combinação das colunas 1, 3 e 6. Consequentemente, o sistema dado por (1.34) *não admite soluções ou admite infinitas soluções.*

Uma outra forma de verificar tal indeterminação é ver que, se $m(x)$ fosse uma função que interpola f em Y definidos acima, então $m(x) + \xi(ax - by - c)$ também interpola f em Y, para qualquer $\xi \in \mathbb{R}$. Argumento idêntico pode ser feito no caso em que os pontos de Y encontram-se sobre uma circunferência e pode também ser generalizado para $n > 2$.

O Exemplo 1.18 indica que algo mais deve ser pedido ao conjunto Y de forma que se garanta existência e unicidade da função de interpolação. Esse requisito é o **equilíbrio** (tradução livre de *poisedness*), o que basicamente pede que a matriz $M(\phi, Y)$ seja não singular. Porém, para obtermos uma medida de quão equilibrado o conjunto Y é, e consequentemente saber quão bom é o modelo dado pela função de interpolação, a prática mais comum é utilizarmos a base $\phi = \{\ell_i(\cdot)\}_{i=1}^p$ dos polinômios de Lagrange de grau até 2 [CSV09b] e calcularmos:

$$\Lambda \geq \max_{i=1,\ldots,p} \max_{B} |\ell_i(x + x)|, \tag{1.36}$$

onde $B \subset \mathbb{R}^n$. Se existir $\Lambda > 0$ tal que (1.36) é verdadeira, então o conjunto é bem equilibrado, ou **Λ-equilibrado**, em B. Outras definições de conjunto Λ-equilibrado podem ser consultadas em [CSV09b], incluindo a relação com polinômios de Newton (usados em [CST97a, CST97b]) e com o número de condição de uma matriz relacionada com $M(\phi, Y)$ (usada em [Pow11]).

O Algoritmo 1.8 mostrado a seguir é conhecido como DFO e foi apresentado em [CST97a]. A grande maioria dos algoritmos de regiões de confiança tenta manter boas propriedades do conjunto interpolante Y ao longo das iterações. Em [CST97a], os autores tentam manter, na iteração k, o conjunto Y **adequado** na região $\overline{B(x^k, \Delta_k)}$, onde $x^k \in Y$ é o centro e Δ_k é o raio da região de confiança. Isso significa garantir que $Y \subset \overline{B(x^k, \Delta_k)}$, que o modelo existe e é ao menos linear (Y é Λ-equilibrado e $p \geq n+1$) e que a base de polinômios de Newton é limitada em $\overline{B(x^k, \Delta_k)}$.

Algoritmo 1.8: Algoritmo DFO

Dados: $0 < \eta_0 \leq \eta_1 < 1$, $0 < \gamma_0 \leq \gamma_1 < 1 < \gamma_2$, $\varepsilon > 0$ e $\kappa \geq 1$.
Entrada: $x^s \in \mathbb{R}^n$ e $f(x^s)$

1. **Inicialização**
   Escolha um conjunto inicial de interpolação Y que contenha x^s e ao menos um outro ponto. Tome x^0 tal que $f(x^0) = \min_{y^i \in Y} f(y^i)$. Escolha o raio inicial da região de confiança $\Delta_0 > 0$ e faça $k \leftarrow 0$.

2. **Construção do modelo**
   Usando Y, construa $m_k(x^k + s)$, como em (1.33), que satisfaça (1.34). Se $\|g^k\|_2 \leq \varepsilon_g$ e Y não é adequado em $\overline{B(x^k, \kappa\|g^k\|_2)}$, então melhore a geometria de Y em $\overline{B(x^k, \delta)}$, para $\delta \in (0, \kappa\|g^k\|_2]$.

3. **Minimização do modelo**
   Encontre $x^k + s^k = \arg\min_{x \in \overline{B(x^k, \Delta_k)}} m_k(x)$.
   Calcule $f(x^k + s^k)$ e
   $$\rho_k \doteq \frac{f(x^k) - f(x^k + s^k)}{m_k(x^k) - m_k(x^k + s^k)}.$$

4. **Atualização de Y**
   se $\rho_k \geq \eta_1$ **então**
   | Insira $x^k + s^k$ em Y, removendo um outro ponto de Y caso $|Y| = p$.
   senão
   | Se Y não é adequado em $\overline{B(x^k, \Delta_k)}$, melhore a geometria de Y nesse conjunto.

5. **Atualização do raio da região de confiança**
   se $\rho_k \geq \eta_1$ **então**
   | $\Delta_{k+1} \in [\Delta_k, \gamma_2 \Delta_k]$
   senão se $\rho_k < \eta_1$ **e** Y *não é adequado em* $\overline{B(x^k, \Delta_k)}$ **então**
   | $\Delta_{k+1} \in [\gamma_0 \Delta_k, \gamma_1 \Delta_k]$
   senão
   | $\Delta_{k+1} = \Delta_k$

6. **Atualização do centro da região de confiança**
   Escolha $\hat{x}^k$ tal que $f(\hat{x}^k) = \min_{y^i \in Y,\ y^i \neq x^k} f(y^i)$.
   Calcule $\hat{\rho}_k = \dfrac{f(x^k) - f(\hat{x}^k)}{m_k(x^k) - m_k(\hat{x}^k)}$
   se $\hat{\rho}_k \geq \eta_0$ **então**
   | $x^{k+1} \leftarrow \hat{x}^k$
   senão
   | $x^{k+1} = x^k$
   $k = k + 1$ e volte para 2.

A ideia central é que com um conjunto adequado em $\overline{B(x^k, \Delta_k)}$ consegue-se limitar superiormente o valor funcional da base de polinômios interpoladores. Com isso, supondo que f é continuamente diferenciável, que $\|\nabla f\|_2$ e $\|\nabla^2 f\|_2$ são limitadas e que as Hessianas de todos os modelos são uniformemente limitadas, os autores demonstraram que

$$|f(x) - m_k(x)| \leq \kappa_1 \max\{\delta^2, \delta^3\} \tag{1.37}$$

$$\|\nabla f(x) - g^k\|_2 \leq \kappa_2 \max\{\delta, \delta^2\}, \tag{1.38}$$

para todo $x \in \overline{B(x^k, \delta)}$, onde $\delta > 0$, g^k é o gradiente do modelo $m_k(x)$ e $\kappa_1, \kappa_2 > 0$ são constantes independentes de δ. Quanto menor δ, melhor é a aproximação definida pelo modelo, por isso tentamos fazer com que $\Delta_k \to 0$.

Dois processos usados no Algoritmo 1.8 não foram explicados ainda: como inserir um novo ponto em Y e como melhorar a geometria de Y em uma bola $\overline{B(x^k, \delta)}$, para algum valor δ.

Quanto à inserção de um novo ponto, se a cardinalidade de Y não é a máxima, ou seja, $|Y| < p$, então podemos simplesmente adicioná-lo em Y. Caso $|Y| = p$, devemos escolher um ponto de Y para ser removido. Os autores em [CST97a] sugerem que o ponto seja aquele associado ao elemento ϕ_i da base de polinômios de Newton tal que $|\phi_i(x)|$ seja máxima, porém outras técnicas podem ser usadas. É importante ressaltar que a inserção ou troca de pontos de Y não garante que ele continue adequado. Por esse motivo, é necessário o processo de melhoria da geometria de Y.

No processo de melhoria da geometria, o objetivo é fazer o conjunto Y voltar a ser adequado em $\overline{B(x^k, \delta)}$, onde geralmente $\delta \in (0, \Delta_k]$. Primeiramente, tentamos remover $y^i \in Y$ tal que $y^i \notin \overline{B(x^k, \delta)}$, caso exista algum. Outra alternativa é remover alguns elementos de Y, usando a mesma técnica do parágrafo anterior, e calcular novos. Em [CST97a] há referências para outros métodos, embora os autores afirmem que não importa o que seja feito, desde que o processo encontre um conjunto adequado em tempo finito.

Ao contrário dos algoritmos de região de confiança usuais, podemos observar no Algoritmo 1.8 que, no caso em que a melhoria do valor de f não é suficientemente boa, não necessariamente reduzimos o raio da região de confiança. Isso se deve ao fato de que o modelo pode ser ruim pela não adequação do conjunto Y, motivo pelo qual o processo de melhoria deve ser feito. Uma outra característica interessante de DFO é que ele permite que, no começo, o modelo seja incompleto, ou seja, que Y tenha menos que $n + 1$ pontos necessários para a construção da interpolação linear.

Todos os resultados conhecidos para regiões de confiança foram recuperados para o caso sem derivadas. A seguir apresentamos o teorema principal de convergência do Algoritmo 1.8, cuja demonstração encontra-se em [CST97a]. Esse resultado se baseia principalmente nos limitantes (1.37) e (1.38).

1.19 Teorema

Suponha que f é continuamente diferenciável e limitada inferiormente, $\|\nabla f\|_2$ e $\|\nabla^2 f\|_2$

são limitadas e que as Hessianas de todos os modelos são uniformemente limitadas. Se x^* *é um ponto de acumulação da sequência gerada pelo Algoritmo 1.8, então* $\nabla f(x^*) = 0$.

Posteriormente, Conn, Scheinberg e Vicente em [CSV09a, CSV09b] usaram os limitantes (1.37) e (1.38) para definir uma classe mais geral de algoritmos, da qual DFO é um caso particular.

Para o caso restrito, podemos mencionar o algoritmo BOBYQA [Pow09] que considera problemas com restrições de caixa e o próprio DFO que em [CST98] é apresentado como um algoritmo que consegue lidar com restrições gerais. Como mostraremos no Capítulo 3, DFO apresenta resultados surpreendentes com problemas restritos. Infelizmente, para nenhum algoritmo desse tipo há teoria de convergência.

Capítulo 2

Busca direta em conjuntos magros

Os algoritmos de busca direta direcional descritos no capítulo anterior possuem boas qualidades de convergência. Infelizmente, seu comportamento piora quanto mais apertado é o conjunto de restrições. No limite, quando o conjunto contém restrições de igualdade ou é desconexo, por exemplo, toda a teoria de convergência não pode ser aplicada diretamente ao problema.

Neste capítulo, discutimos um método cuja ideia principal é a resolução de uma sequência de subproblemas nos quais o conjunto viável é gordo, porém vai emagrecendo a cada passo até, no limite, se tornar o conjunto viável do problema original. Como sabemos ser possível aplicar algoritmos de otimização sem derivadas aos subproblemas, a esperança é que a sequência de soluções dos subproblemas convirja a minimizadores locais ou pontos estacionários.

As condições essenciais que sustentam o algoritmo proposto são que a função objetivo f não possui derivadas disponíveis (embora estas possam existir) e é computacionalmente cara de ser avaliada, e que a restrição g é computacionalmente simples e pode possuir ou não as derivadas disponíveis. Uma hipótese adicional sobre o conjunto viável é que, embora as restrições sejam fáceis, o conjunto é magro (não gordo, pela Definição 1.12) e topologicamente complicado[1].

Funções objetivo que demandam grande esforço computacional são comuns em problemas de otimização sem derivadas. Se adicionamos restrições simples e diferenciáveis, mas altamente não lineares, a fase de viabilidade pode ser complexa, embora o seu custo computacional seja irrelevante. Quando as restrições formam um conjunto magro, este é o cenário ideal para aplicarmos o algoritmo proposto.

O algoritmo que apresentamos neste capítulo se aproveita dessas características para separar o processo de resolução em dois: a fase de restauração, na qual um ponto viável é encontrado, e a fase de minimização, na qual a solução de um subproblema engordado é encontrada. Esse processo de separação em duas fases e de minimização em um conjunto viável um pouco mais relaxado do que o original pode ser denominado de restauração inexata, em analogia aos métodos de **Restauração Inexata** apresen-

[1]No nosso sentido, totalmente informal, em um conjunto topologicamente complicado, a tarefa de encontrar um ponto que o pertença pode ser uma tarefa computacionalmente complexa.

tados em [Mar98, MP00]. Porém, outros autores utilizaram conceitos semelhantes para desenvolver algoritmos, como o **método da tolerância flexível** (*flexible tolerance method*) de Paviani e Himmelblau [PH69, Him72] e algoritmos que seguem o espírito da programação quadrática sequencial e separam a busca por uma direção de descida em duas: a direção normal e a tangente, tais como [MP00, BG08, GT10, GT12]. Esses métodos consideram cilindros de confiança [BG08] e funis de confiança [GT10, GT12] na tentativa de resolver um problema um pouco mais "gordo" a cada passo.

Na Seção 2.1, recapitulamos a ideia do Algoritmo Simplicíssimus e fornecemos a motivação para o desenvolvimento de um algoritmo para tratar conjuntos magros. O algoritmo proposto é descrito na Seção 2.2, na qual também apresentamos sua teoria de convergência a minimizadores globais do problema. Na Seção 2.3, apresentamos uma sugestão para o algoritmo a ser usado na fase de minimização e, na Seção 2.4, são feitas algumas considerações adicionais.

2.1 Motivação

Vamos recordar o problema de programação não linear considerado até o momento:

$$\begin{aligned} \min \quad & f(x) \\ \text{s. a} \quad & x \in X, \end{aligned} \tag{2.1}$$

onde $X \subset \mathbb{R}^n$ é um conjunto magro (ou seja, que não é gordo, segundo a Definição 1.12) e $f : X \to \mathbb{R}^n$ é contínua. Com relação ao conjunto viável X, é provável que a noção mais intuitiva de transformá-lo em um conjunto gordo seja adicionar bolas fechadas em torno de seus elementos originais, ou seja, dado $\gamma > 0$, considerar o conjunto

$$X_\gamma \doteq \bigcup_{x \in X} \overline{B(x, \gamma)}. \tag{2.2}$$

É fácil ver que para qualquer $\gamma > 0$ o conjunto X_γ dado por (2.2) é um conjunto gordo, de acordo com a Definição 1.12. No Capítulo 1, discutimos sobre o Algoritmo Simplicíssimus (Algoritmo 1.4), capaz de resolver problemas sem derivadas em conjuntos gordos. Utilizando Simplicíssimus, o Teorema 2.1 a seguir descreve o espírito central do algoritmo estudado neste capítulo.

2.1 Teorema
Seja a sequência $\left\{x^k\right\}_{k \in \mathbb{N}}$, *onde* x^k *é o ponto limite da sequência gerada pelo Algoritmo Simplicíssimus aplicado no problema:*

$$\begin{aligned} \min \quad & f(x) \\ s.\ a \quad & x \in X_{\gamma_k}, \end{aligned} \tag{2.3}$$

com $\gamma_k > 0$, $\{\gamma_k\} \to 0$ *e* f *contínua. Se existe* $K \subseteq \mathbb{N}$ *tal que* $\lim_{k \in K} x^k = x^*$, *então, para todo* $\delta \in (0, \Delta)$, x^* *é um minimizador global do problema:*

$$\begin{aligned} \min_{x \in X} \quad & f(x) \\ s.\ a \quad & \|x - x^*\| \leq \delta. \end{aligned}$$

Demonstração:

Seja $\delta \in (0, \Delta)$ e $z \in \overline{B(x^*, \delta)} \cap X$. Então, para $k \in K$ suficientemente grande,

$$\|z - x^k\| \leq \|z - x^*\| + \|x^* - x^k\| < \Delta.$$

Para todo $\gamma_k > 0$, o problema (2.3) encaixa-se nas hipóteses do Teorema 1.11, que garante a convergência do Algoritmo Simplicíssimus a minimizadores globais do problema

$$\begin{array}{ll} \min\limits_{x \in X_{\gamma_k}} & f(x) \\ \text{s. a} & \|x - x^k\| \leq \delta. \end{array}$$

Logo, sabemos que $f(x^k) \leq f(z)$. Pela continuidade de f, podemos tomar o limite, para $k \in K$, e temos

$$f(x^*) \leq f(z).$$

A desigualdade acima vale para todo $z \in \overline{B(x^*, \delta)} \cap X$. Como δ foi escolhido arbitrariamente, o resultado segue. ∎

O Teorema 2.1 representa exatamente o que estamos propondo para resolver problemas do tipo (2.1). Existem, todavia, requisitos implícitos no teorema e que foram usados sem o seu devido cuidado. Em primeiro lugar, não temos como garantir que para todo problema da forma (2.3) Simplicíssimus conseguirá gerar uma sequência convergente. Em segundo lugar, podemos observar que Simplicíssimus necessita de um ponto inicial *viável*, e ainda não sabemos como encontrar tal ponto em cada iteração. Em terceiro lugar, a representação do conjunto viável do problema (2.3) ainda está muito abstrata e longe de poder ser implementada computacionalmente, além do fato de que nunca será possível gerar totalmente uma sequência de direções densa na bola de raio Δ. Ao longo deste capítulo, estudaremos cada inconveniente encontrado no método teórico do Teorema 2.1, sugerindo modificações e verificando sua aplicabilidade. Começaremos com a apresentação de um método, descrito em 1969, que também trabalha com relaxamentos do conjunto viável original.

2.1.1 O método da tolerância flexível

O método da tolerância flexível foi apresentado por Paviani e Himmelblau em 1969 [PH69], mas é no livro de Himmelblau [Him72], publicado em 1972, que o método ganhou o nome que o define e também uma detalhada explicação de seu funcionamento. De uma maneira geral e simplificada, podemos dizer que o método da tolerância flexível é uma adaptação do método de Nelder e Mead [NM65] para lidar com restrições de igualdade e desigualdade.

Para os autores, o problema de programação não linear

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & g(x) \leq 0 \\ & h(x) = 0, \end{array}$$

com $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ e $h : \mathbb{R}^n \to \mathbb{R}^p$, é convertido em um novo problema, com apenas uma restrição de desigualdade:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & T(x) - \Phi_k \leq 0, \end{array}$$

onde,

$$\begin{aligned} T(x) &= \sqrt{\sum_{i=1}^{p} h_i(x)^2 + \sum_{i=1}^{m} [g(x)_+]_i^2}, \\ [g(x)_+]_i &= \max\{0, g_i(x)\} \quad \text{e} \\ \Phi_k &= \min\{\Phi_{k-1}, \theta_k\}. \end{aligned}$$

O parâmetro Φ_k define quão tolerante o método será com respeito aos pontos inviáveis, onde Φ_0 é dado pelo tamanho inicial do simplex. O parâmetro θ_k representa a distância média dos vértices do simplex, na iteração k, ao seu centroide.

Para medir a inviabilidade utiliza-se a função T, que é sempre não negativa e ainda possui a propriedade que $T(x) = 0$ se e somente se x é um ponto viável. Utilizando Φ_k e T, dado um ponto x qualquer existem três tipos de classificação:

- viável, se $T(x) = 0$;
- quase viável, se $0 < T(x) \leq \Phi_k$;
- inviável, se $T(x) > \Phi_k$.

De forma similar à feita em [NM65], um simplex com $r + 1$ pontos é construído, onde $r = n - p$ representa o grau de liberdade do problema, com $p \leq n$, e lembrando que p é o número de restrições de igualdade.

Em cada iteração do método tenta-se substituir o vértice com o maior valor de função objetivo associado. Para isso, um novo vértice é calculado utilizando uma das operações: reflexão, expansão ou contração. Em cada um dos casos, verifica-se se o novo ponto é um ponto quase viável e, em caso negativo, um tipo de restauração é aplicado.

O processo de restauração consiste em encontrar um novo ponto viável ou quase viável que esteja, preferencialmente, próximo ao ponto considerado inviável. Para atingir esse objetivo, o método Nelder–Mead é aplicado na minimização (irrestrita) de T. Somente após o ponto ser considerado viável ou quase viável é que será avaliada a função objetivo, visando poupar o esforço computacional. Se, em algum momento do processo, for encontrado um ponto viável ou quase-viável com valor funcional menor do que o maior valor funcional do simplex, o novo ponto substitui aquele com maior valor e o processo recomeça. Caso nenhum dos pontos calculados seja considerado apto a entrar no simplex, é feita uma contração total, na qual o simplex todo reduz de tamanho em torno do ponto com menor valor da função objetivo e o processo recomeça.

A convergência do método a algum tipo de minimizador ou ponto estacionário não foi provada, mas o argumento de convergência dos autores baseia-se no mesmo argumento do método de Nelder e Mead: quando o simplex torna-se suficientemente pequeno, significa que o ponto possui alguma possibilidade de ser um minimizador.

O parâmetro de Φ_k define a tolerância com a qual pontos inviáveis são aceitos nas iterações do método. Esse parâmetro é não crescente e decresce apenas se o parâmetro θ_k diminuir. O parâmetro θ_k, por sua vez, está relacionado com o tamanho do simplex no final da iteração k. Como o método só trabalha com pontos viáveis e/ou quase viáveis e o politopo só diminui com uma contração ou uma contração total, os autores esperavam que, quando houvesse convergência para um ponto viável ($\Phi_k = 0$), então o método também estaria em um tipo de minimizador, pois o diâmetro do politopo seria 0.

Por fim, é mencionado em [Him72] que não é necessário aplicar o algoritmo Nelder–Mead para a fase de "restauração". Qualquer método para minimização de problemas irrestritos pode ser utilizado.

2.2 Otimização em conjuntos magros

Considere o seguinte problema de programação não linear:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & g(x) \leq 0, \end{array} \tag{2.4}$$

com $f : \mathbb{R}^n \to \mathbb{R}$ e $g : \mathbb{R}^n \to \mathbb{R}^m$. Denotamos por $\Omega \doteq \{x \in \mathbb{R}^n \mid g(x) \leq 0\}$ o conjunto viável de (2.4). Utilizamos Ω para este conjunto, para diferenciá-lo do conjunto X, utilizado até o momento para denotar um conjunto viável qualquer. A restrição de igualdade $h(x) = 0$, $h : \mathbb{R}^n \to \mathbb{R}^p$, pode ser naturalmente convertida em duas restrições de desigualdade da forma $h(x) \leq 0$ e $-h(x) \leq 0$, por isso trabalharemos apenas com restrições de desigualdade.

2.2 Definição (Conjunto engordado)
*Seja $\Omega \doteq \{x \in \mathbb{R}^n \mid g(x) \leq 0\}$ e $\gamma \geq 0$. O **engordamento de Ω por γ** é definido pelo conjunto*

$$\Omega_\gamma \doteq \{x \in \mathbb{R}^n \mid g(x) \leq \gamma\}.$$

Um ponto $x \in \Omega_\gamma$ é denominado γ-viável.

A diferença entre o engordamento geométrico que define o conjunto X_γ (definido pela equação (2.2)) e o engordamento algébrico Ω_γ dado na Definição 2.2 é clara: enquanto o primeiro se aplica diretamente no conjunto viável, o segundo está associado com a função g. O Exemplo 2.3 mostra essa diferença. Uma condição que permite relacionar os dois tipos de engordamento é a Lipschitz continuidade de g, como mostra a Proposição 2.4.

2.3 EXEMPLO
Considere a função $g(x) = \frac{x^4}{4} - x^3 + 1.1x^2$. *É fácil verificar que*

$$\Omega = \{x \in \mathbb{R} \mid g(x) \leq 0\} = \{0\} = X.$$

A Figura 2.1 mostra a comparação entre os dois tipos de engordamento discutidos. Tomando $\gamma = 1.5 + \frac{\sqrt{0.2}}{2}$ *temos que o engordamento dado pela Definição 2.2 não é apenas uma bola de raio* γ *em torno do ponto* 0, *como o engordamento dado pela equação* (2.2).

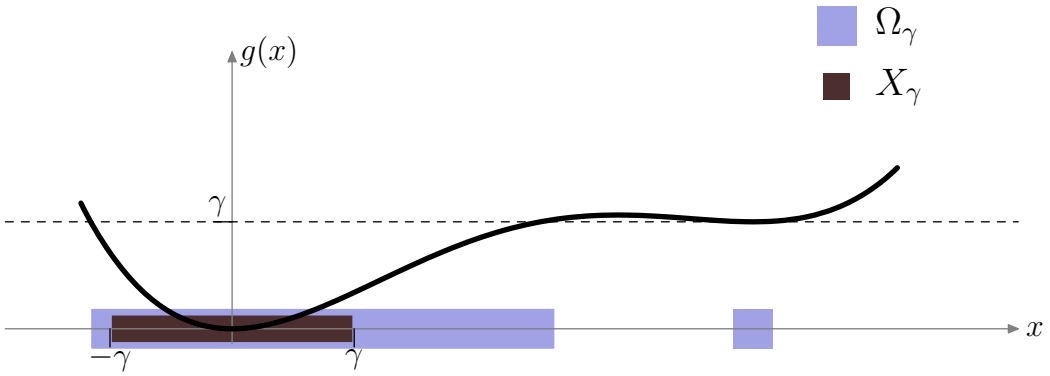


Figura 2.1: Diferença entre o engordamento geométrico e o engordamento algébrico.

2.4 PROPOSIÇÃO
Se a função $g : \mathbb{R}^n \to \mathbb{R}^m$ *é Lipschitz contínua, com constante de Lipschitz* L, *então para todo* $\gamma > 0$ *existe* $\varepsilon > 0$ *tal que*

$$X_\varepsilon \subset \Omega_\gamma.$$

Mais ainda, o valor de ε *é dado por* γ/L.

DEMONSTRAÇÃO:

Sabemos que $\|g(x) - g(y)\| \leq L\|x - y\|$. Seja $x \in X$, onde $X = \{x \in \mathbb{R}^n \mid g(x) \leq 0\} = \Omega$, e y tal que $\|x - y\| \leq \gamma/L$. Então, para $i = 1, \ldots, m$,

$$|g_i(x) - g_i(y)| \leq \|g(x) - g(y)\| \leq L\|x - y\| \leq \gamma,$$

supondo que a norma $\|\cdot\|$ satisfaça a primeira desigualdade. Logo,

$$g_i(y) \leq g_i(x) + \gamma \leq \gamma.$$

Portanto, para todo $y \in X_{\gamma/L}$ temos que $y \in \Omega_\gamma$. Como γ foi escolhido de forma arbitrária, temos o resultado. ■

A Proposição 2.4 também pode ser provada supondo-se Lipschitz continuidade de cada função $g_i(x)$, $i = 1, \ldots, m$. Para a prova, basta considerarmos $L = \max_{i=1,\ldots,m}\{L_i\}$, onde L_i é a constante de Lipschitz da respectiva restrição g_i. Embora a hipótese seja mais forte, a vantagem dessa abordagem é que não precisamos supor que a norma satisfaz a desigualdade

$$|x_i| \leq \|x\|, \quad i = 1, \ldots, n.$$

Podemos ver no Exemplo 2.3 que o conjunto Ω_γ não necessariamente é um conjunto gordo, embora contenha um conjunto gordo, pela Proposição 2.4. Assim, esperamos resolver o problema (2.4) aplicando o método sugerido no enunciado do Teorema 2.1, resolvendo, em cada passo, o problema engordado

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & g(x) \leq \gamma_k. \end{array} \tag{2.5}$$

Em outras palavras, conseguimos substituir a noção abstrata do conjunto viável X pela definição prática de (2.4).

No Capítulo 1 discutimos e estudamos algoritmos de otimização sem derivadas que constroem uma malha sobre o conjunto viável e avaliam a função objetivo em determinados pontos dessa malha. Conforme o algoritmo é iterado, essa malha torna-se cada vez mais fina. A malha e o seu refinamento são os conceitos que mais podemos associar à construção de um conjunto denso de direções, pois como podemos ver na Figura 2.2, quanto mais refinada é a malha, menor é a distância entre seus pontos e os pontos de todo o espaço, em particular os pontos do conjunto Ω.

Vamos lembrar que a malha utilizada pelos algoritmos de busca direcional é definida pelas direções de busca e pelo parâmetro de refinamento. As direções, que definem a estrutura da malha, não são alteradas durante as iterações, enquanto o refinamento poder aumentar ou diminuir de acordo com o valor de f. Ao utilizarmos o termo **Malha** a seguir, supomos implícita a dependência em um conjunto de direções de busca e em um parâmetro de refinamento. Não há necessidade de uma escolha específica do conjunto de direções de busca, embora as direções utilizadas na busca coordenada sejam interessantes para obtermos uma visualização da malha.

Computacionalmente, a malha também é um recurso interessante para ser usado. Se supomos, por exemplo, que todo o conjunto viável está contido em uma caixa, temos que uma malha construída sobre esse conjunto terá um número finito de pontos, por mais refinada que a malha possa estar. Mais ainda: ao invés de procurarmos todos os pontos viáveis na malha, podemos nos deter apenas naqueles que estão em uma vizinhança do ponto corrente, como, por exemplo, na bola de raio unitário, como mostra a Figura 2.2. Neste momento, estamos aptos a apresentar uma maneira mais realista de estudar um algoritmo como o Algoritmo Simplicíssimus.

As propriedades de convergência de Simplicíssimus foram demonstradas utilizando a hipótese de que o conjunto de direções de pesquisa é denso na bola de raio Δ. Porém, na prática devemos ter algum critério de parada que impeça o algoritmo de

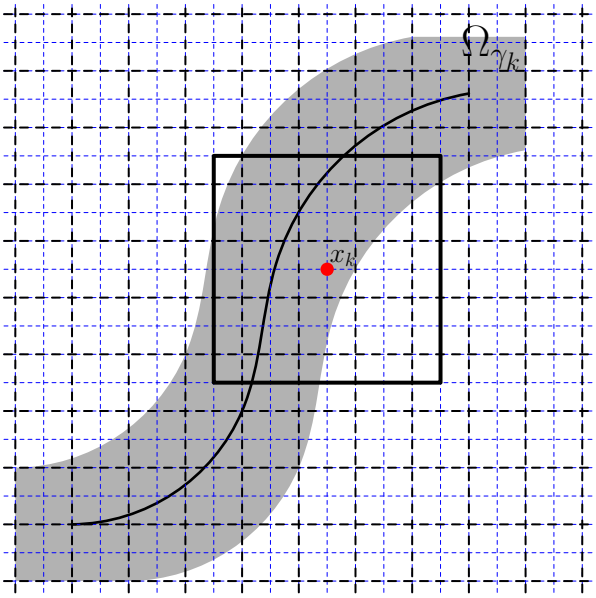


Figura 2.2: O refinamento da malha está intimamente ligado à construção de um conjunto denso de direções.

executar um número infinito de iterações. Portanto, ao final de sua execução, ele não terá utilizado um conjunto denso de direções e sim um conjunto **finito**. Essa simples observação em conjunto com a ideia da malha permite que troquemos o fato de x^k ser um minimizador local de um problema do tipo (2.3) (resultado da aplicação de Simplicíssimus sob as hipóteses do Teorema 1.11) por $x^k \in \Omega_{\gamma_k}$ tal que

$$f(x^k) \leq f(x) \quad \forall x \in B(x^k, \delta) \cap \Omega_{\gamma_k} \cap \text{Malha} \tag{2.6}$$

para valores adequados de δ, γ_k e para uma malha refinada em conformidade com o conjunto Ω_{γ_k}. Note que o número de pontos viáveis que estão na malha e em uma bola de raio δ em torno de x^k é finito, portanto, computável. Essa condição pode ser vista como uma condição necessária de otimalidade para um minimizador local do problema engordado (2.5).

Para que a condição (2.6) possa ser utilizada, deve ser satisfeita por algum ponto. Para isso, precisamos descrever um modo de refinar a malha que esteja em conformidade com o conjunto viável Ω_{γ_k}.

Hipótese M – Dado $\gamma > 0$, para todo $x \in \Omega_\gamma$ existe y na malha tal que

$$\|x - y\| \leq \gamma^2.$$

Para todo $\gamma > 0$, podemos facilmente satisfazer a Hipótese M se definimos uma matriz $D = [\pm e_1 \cdots \pm e_n]$ e consideramos uma malha descrita por todos os pontos da forma $x = x^0 + \min\{1, \gamma^2\} Dz$, para algum $x^0 \in \mathbb{R}^n$ e todo $z \in \mathbb{Z}^{2n}$. Esta malha teria um aspecto semelhante ao da Figura 2.2.

2.5 Lema (Refinamento da malha)
Suponha que a função g seja Lipschitz contínua e que o refinamento da malha satisfaça a Hipótese M. Então, dado $\beta \in [0,1)$, existe $\bar{\gamma} > 0$ tal que, para todo $0 < \gamma \leq \bar{\gamma}$ e para todo $x \in \Omega_{\beta\gamma}$ existe ao menos um ponto y na malha tal que

$$\|x - y\| \leq \gamma^2 \quad e \quad y \in \Omega_\gamma.$$

Demonstração:

Seja $\gamma > 0$ e $x \in \Omega_{\beta\gamma}$, ou seja, $g(x) \leq \beta\gamma$. Então, usando a Hipótese M, existe y na malha tal que

$$\|g(x) - g(y)\| \leq L\|x - y\| \leq L\gamma^2.$$

Logo, supondo que a norma $\|\cdot\|$ satisfaz a desigualdade $|x_i| \leq \|x\|$, temos

$$g_i(y) \leq g_i(x) + L\gamma^2 \leq \beta\gamma + L\gamma^2, \tag{2.7}$$

para todo $i = 1, \ldots, m$. Estamos interessados em saber para quais valores de γ a desigualdade:

$$\beta\gamma + L\gamma^2 \leq \gamma$$

é verdadeira, para que a equação (2.7) seja válida. Resolvendo a inequação, temos que $\bar{\gamma} = \dfrac{1-\beta}{L}$. Logo, se $\gamma \leq (1-\beta)/L$ então $g_i(y) \leq \gamma$, $i = 1, \ldots, m$, e concluímos que $y \in \Omega_\gamma$. ∎

A Proposição 2.4 mostrou que com a Lipschitz continuidade somos capazes de relacionar um problema engordado da forma (2.5) com a teoria de convergência de Simplicíssimus. Agora, o Lema 2.5 mostrou que sob a Hipótese M é possível gerar uma malha de pontos de forma que a intersecção desta com o conjunto engordado Ω_{γ_k} seja não vazia (necessária para a condição (2.6)). O Algoritmo 2.1 apresentado a seguir e a convergência demonstrada no Teorema 2.6 são os primeiros passos para a resolução prática de problemas de otimização que possuem o conjunto viável magro, descritos pelo problema (2.4). Não podemos deixar de ressaltar que a condição (2.6) também exige que x^k seja γ_k-viável, o que deixa implícito um problema de viabilidade, e que vai se tornando cada vez mais difícil, conforme o conjunto engordado Ω_{γ_k} converge para o conjunto viável do problema original. Lembremos que temos a nosso favor a suposição de que o custo computacional da avaliação das restrições é irrelevante se comparado ao da função objetivo, portanto o algoritmo em geral não é afetado pela crescente dificuldade em encontrar pontos viáveis.

2.6 Teorema
Seja f contínua, g Lipschitz contínua e suponha que a construção da malha, em cada iteração, satisfaz a Hipótese M para $\gamma = \gamma_k$. Se existe $K \subseteq \mathbb{N}$ tal que $\lim\limits_{k \in K} x^k = x^*$, onde

Algoritmo 2.1: Algoritmo de Engordamento com Malhas

Dados: $\gamma_k > 0$ tal que $\lim_{k\to\infty} \gamma_k = 0$, $\Delta > 0$ e $k \leftarrow 0$

1. Encontre $y^k \in \Omega_{\gamma_k}$.
2. Utilizando y^k como ponto inicial, calcule $x^k \in \Omega_{\gamma_k}$ tal que

$$f(x^k) \leq f(x)$$

   para todo $x \in \Omega_{\gamma_k} \cap B(x^k, \Delta)$ pertencente à malha.
3. Faça $k \leftarrow k + 1$ e volte para o passo 1.

$\{x^k\}$ foi gerada pelo Algoritmo 2.1, então x^* é solução global do problema:

$$\begin{array}{ll} \min & f(x) \\ s.\ a & g(x) \leq 0 \\ & \|x - x^*\| \leq \Delta. \end{array}$$

Demonstração:

Suponha por contradição que existe $z^* \in \Omega = \{x \in \mathbb{R}^n \mid g(x) \leq 0\}$ tal que $f(z^*) < f(x^*)$, mais especificamente, $f(z^*) = f(x^*) - \xi$, para algum $\xi > 0$. Pela continuidade de f, existe $\delta > 0$ tal que

$$f(x) < f(x^*) - \frac{\xi}{2} \quad \forall\, x \in B(z^*, \delta). \tag{2.8}$$

Vamos dividir esta demonstração em dois casos: (i) quando z^* está no interior da bola de raio Δ com centro em x^* e (ii) quando z^* está em sua borda.

(i) Suponha que $\|z^* - x^*\| < \Delta$. A sequência $\{\gamma_k\}$ tende a zero por definição e, usando o Lema 2.5 (com $\beta = 0$) sabemos que existe $N_1 > 0$ tal que para todo $k > N_1$, $k \in K$, existe z^k na malha tal que $z^k \in \Omega_{\gamma_k}$ e $\|z^k - z^*\| \leq \gamma_k^2 < \delta$, onde δ é dado por (2.8).

Suponha, sem perda de generalidade, que δ seja suficientemente pequeno de forma que $B(z^*, \delta) \subset B(x^*, \Delta)$. Então existe $N_2 > N_1$ tal que

$$B(z^*, \delta) \subset B(x^k, \Delta)$$

para todo $k > N_2$, $k \in K$.

Agora tomamos $k > N_3 > N_2$ tal que $|f(x^k) - f(x^*)| \leq \xi/3$. Temos que para todo $k > N_3$, $k \in K$, existe z^k na malha, $z^k \in \Omega_{\gamma_k} \cap B(x^k, \Delta)$, tal que

$$f(x^k) > f(x^*) - \frac{\xi}{3} > f(x^*) - \frac{\xi}{2} > f(z^k),$$

o que é uma contradição com a forma em que o algoritmo gera os pontos x^k.

(ii) Suponha agora que z^* é tal que $\|z^* - x^*\| = \Delta$. Pela Proposição 2.4, para todo $\gamma > 0$ temos que $B(z^*, \gamma/L) \subset \Omega_\gamma$, pela definição do conjunto $X_{\gamma/L}$, onde L é a constante de Lipschitz da função g. Assim, tomando $\gamma = \gamma_k/2$, para todo k podemos verificar que existe

$$z^k \in B(z^*, \gamma_k/2L) \cap B(x^*, \Delta) \cap \Omega_{\gamma_k/2},$$

pois $z^* \in \overline{B(x^*, \Delta)}$.

Como $z^k \in \Omega_{\gamma_k/2}$, usamos o Lema 2.5 com $\beta = 1/2$ para garantir que, para γ_k suficientemente pequeno, existe $y^k \in \Omega_{\gamma_k}$ na malha tal que

$$\|z^k - y^k\| \leq \gamma_k^2 < \delta$$

e, como $z^k \in B(x^*, \Delta)$,

$$\|y^k - x^*\| \leq \|y^k - z^k\| + \|z^k - x^*\| \leq \gamma_k^2 + \|z^k - x^*\| < \Delta,$$

ou seja, $y^k \in B(x^*, \Delta)$. Logo, podemos tomar $N_1 > 0$ suficientemente grande tal que para todo $k > N_1$, $k \in K$, existe um ponto na malha $y^k \in \Omega_{\gamma_k} \cap B(x^*, \Delta)$ tal que $\|z^* - y^k\| < \delta$, o que implica que

$$f(y^k) < f(x^*) - \xi/2,$$

em virtude de (2.8).

Por fim, existe $N_2 > N_1$ tal que para todo $k > N_2$, $k \in K$, temos que $y^k \in B(x^k, \Delta)$ e também

$$f(x^k) > f(x^*) - \frac{\xi}{3} > f(x^*) - \frac{\xi}{2} > f(y^k),$$

o que mais uma vez é uma contradição com a definição de x^k dada pelo Algoritmo 2.1.

■

O passo 1 do Algoritmo 2.1, como pode ser visto na demonstração do Teorema 2.6, é teoricamente irrelevante. A viabilidade no ponto limite é conseguida através das condições exigidas no passo 2 do algoritmo. A restauração é necessária, contudo, para que possamos utilizar algoritmos baseados em direções viáveis no passo 2, que necessitam de um ponto inicial viável. Nesse caso, utilizamos y^k como ponto γ_k-viável inicial e podemos aplicar a busca coordenada, MADS, GSS, entre outros.

Os resultados conseguidos até então são promissores. Já é possível vislumbrarmos uma implementação razoavelmente eficiente que consiga resolver problemas de otimização sem derivadas com restrições magras. Porém, há um inconveniente no Algoritmo 2.1, que impossibilita sua eficiência completa: calcular todos os pontos de uma

malha suficientemente refinada. Da mesma forma que construir um conjunto denso de direções em uma bola, gerar pontos em uma malha é uma tarefa impossível de ser realizada na prática. Muito mais interessante é considerarmos pequenas amostras de pontos nessa malha e fazer tal escolha de forma que, no limite, um conjunto denso de pontos tenha sido consultado. Podemos generalizar mais ainda essa escolha, de forma que os pontos não necessitem estar na malha, apenas associados com direções na bola de raio Δ. Essa é a principal inspiração para o Algoritmo 2.2.

Algoritmo 2.2: Algoritmo de Engordamento

Dados: $\beta \geq 1$, $\gamma_k \geq 0$ tal que $\{\gamma_k\} \to 0$, $\eta_k > 0$ tal que $\{\eta_k\} \to 0$ e $\{m_k\}$ tal que $m_k \in \mathbb{N}^*$.

$k \leftarrow 1$

1. **Fase de restauração**
   Escolha $y^k \in \Omega_{\gamma_k}$.
2. **Fase de minimização**
   Dado $\xi_k \in [\eta_k, \beta\eta_k]$ e utilizando y^k, escolha $x^k \in \Omega_{\gamma_k}$ tal que

$$f(x^k) \leq f(\varphi_k(x^k + d)) + \xi_k, \tag{2.9}$$

   para todo $d \in D_k$, onde $D_k = \{d^i \in \mathbb{R}^n, i = 1, \ldots, m_k \mid \varphi_k(x^k + d^i) \in \Omega_{\gamma_k}\}$.
   (O operador φ_k é definido de forma que $\varphi_k(x^k + d) \in \Omega_{\gamma_k}$ para todo $d \in D_k$.)
3. $k \leftarrow k + 1$ e volte para 1.

O diferencial do Algoritmo 2.2 está no passo 2. Em primeiro lugar porque não há mais busca em uma malha e sim em uma amostra limitada em torno de x^k, dada pelo conjunto D_k. Em segundo lugar, pelo aparecimento de algo que, com um certo abuso de linguagem, podemos chamar de "operador" $\varphi_k(\cdot)$. Tal "operador" é definido apenas para os pontos descritos pelos vetores de D_k e pode ser visto como uma projeção de $x^k + d$ no conjunto Ω_{γ_k}. Veja a Figura 2.3.

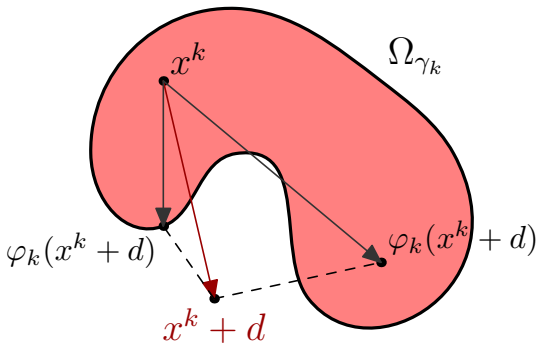


Figura 2.3: Duas (dentre infinitas) possibilidades para definir $\varphi_k(x^k + d)$.

Quando consideramos a função penalidade extrema [ADJ06]:

$$f_{\Omega_{\gamma_k}}(x) = \begin{cases} f(x), & \text{se } x \in \Omega_{\gamma_k} \\ +\infty, & \text{caso contrário} \end{cases} \tag{2.10}$$

no lugar de f, o uso de φ_k torna-se ainda mais evidente. Ao garantirmos que o ponto $\varphi_k(x^k + d)$ é γ_k-viável, garantimos também que o valor verdadeiro da função é usado na comparação com $f(x^k)$ em (2.9). Por conseguinte, temos uma amostra de valores de f em uma vizinhança de x^k que realmente nos fornece informações sobre a otimalidade do ponto x^k.

O Algoritmo 2.2 é bastante geral, mas não tem o Algoritmo 2.1 como seu caso particular. Uma das razões é o decréscimo suficiente da função objetivo, pedido em (2.9). A vantagem do novo algoritmo é que, ao demonstrarmos sua convergência, permitimos uma grande liberdade na maneira como o passo de minimização será feito, contanto que a condição (2.9) seja satisfeita. De fato, na Seção 2.3 apresentamos um modo de fazê-lo, que, ao ser acoplado no Algoritmo 2.2, resulta no algoritmo com o qual os resultados numéricos são discutidos, no Capítulo 3.

Outro aspecto do algoritmo é que não há a necessidade de utilizarmos $\gamma_k > 0$. Como ficará claro na demonstração do teorema de convergência, podemos tomar $\gamma_k = 0$, para todo k, que o algoritmo continua bem definido. Na prática, todavia, essa escolha não é aconselhável, dado que queremos usar algoritmos bem estabelecidos para conjuntos gordos. Ademais, restaurar no conjunto original ($\gamma = 0$) pode ser mais difícil do que no conjunto engordado ($\gamma > 0$).

Por fim, vale destacar que o parâmetro ξ_k utilizado no passo 2 do Algoritmo 2.2 pode ser substituído simplesmente por η_k, levando à consequente eliminação do parâmetro β. Em ambos os casos, estamos pedindo decréscimo suficiente da função objetivo. Optamos por descrever o algoritmo dessa forma para que seja evidenciada a possibilidade de relaxar a condição (2.9), ao permitirmos valores maiores que η_k. Já a limitação inferior de ξ_k não é utilizada na demonstração, mas será necessária para garantir que o algoritmo utilizado no passo de minimização termine em um número finito de iterações.

Como é de praxe na teoria dos métodos sem derivadas, a introdução do decréscimo suficiente traz uma grande simplificação da teoria, libertando os algoritmos do uso de malhas. Na literatura, podemos citar uma comparação entre MADS [ADJ06] e o algoritmo descrito em [VC10] e entre os algoritmos definidos em [LT00] e em [KLT03].

No Teorema 2.7 que se segue, demonstramos a convergência do Algoritmo 2.2 a minimizadores locais do problema (2.4). Além das hipóteses usuais que utilizamos até então, precisamos de uma hipótese adicional sobre o "operador" $\varphi_k(\cdot)$, descrita a seguir.

Hipótese S1 – Se $\{x^k\}_{k\in K}$, $K \subseteq \mathbb{N}$, é uma subsequência convergente gerada pelo Algoritmo 2.2, $z \in \Omega$, $d^k \in D_k$ para todo $k \in K$ e $\lim_{k\in K} x^k + d^k = z$, então

$$\lim_{k\in K} \varphi_k(x^k + d^k) = z.$$

A Hipótese S1 faz com que φ_k tenha um comportamento de continuidade que lembra a projeção ortogonal. Com isso, tentamos garantir que, quando o valor da função objetivo no ponto atual x^k é comparado com o valor em um ponto da vizinhança, na forma $x^k + d$, esse ponto esteja em Ω_{γ_k}. Utilizando a definição de $\varphi_k(x^k + d)$ e a Hipótese S1 temos que, se z é um ponto de acumulação de $\left\{x^k + d^k\right\}$, então pontos γ_k-viáveis arbitrariamente próximos de z foram consultados na comparação com os valores de $f(x^k)$.

O papel da Hipótese S1 no Teorema 2.7 é similar ao da Hipótese M no Teorema 2.6. No Algoritmo 2.1 a malha era usada para consultar a região em torno de x^k, portanto tornava-se necessário garantir que existiam pontos da malha, suficientemente próximos de x^k, que pertencessem ao conjunto engordado Ω_{γ_k}. Com a Hipótese S1 temos a mesma garantia sem a utilização de malhas.

Porém, na Hipótese M, as malhas também estavam atreladas à ideia da densidade. Quanto mais refinada fosse a malha, maior o número de pontos viáveis consultados na vizinhança de x^k. Esse é outro diferencial entre os algoritmos 2.1 e 2.2. No teorema de convergência do Algoritmo 2.2 também necessitamos da densidade dos pontos consultados, mas desejamos que tal densidade seja conseguida através da união dos conjuntos de direções de busca D_k, como feito no Algoritmo Simplicíssimus do Capítulo 1. A hipótese a seguir resume as características que as direções de busca devem ter quanto à densidade.

Hipótese S2 – A sequência $\{m_k\}_{k\in\mathbb{N}}$ é limitada e, para toda subsequência convergente $S \subset \mathbb{N}$, o conjunto

$$D = \bigcup_{k\in S} D_k$$

é denso na bola de raio $\Delta > 0$.

Com as devidas hipóteses descritas e esclarecidas, provamos convergência do Algoritmo 2.2 a minimizadores locais de (2.4). Se um valor de Δ suficientemente grande for escolhido, é possível provar convergência a minimizadores globais de (2.4), um resultado similar ao apresentado em [Ano72] para o algoritmo que gerava um conjunto denso e enumerável de pontos. Lembramos que o uso da densidade fatalmente acarreta na geração de um algoritmo ineficiente. A principal contribuição do Teorema 2.7 é a indicação de que todo o esquema de escolha de direções de busca descrito conduz à solução desejada.

2.7 Teorema

Sejam f e g funções contínuas e suponha que as hipóteses S1 e S2 são válidas. Se existe $K \subseteq \mathbb{N}$ tal que $\lim_{k\in K} x^k = x^*$, onde $\left\{x^k\right\}$ foi gerada pelo Algoritmo 2.2, então x^* é solução global do problema:

$$\begin{array}{ll} \min & f(x) \\ s.\ a & g(x) \leq 0 \\ & \|x - x^*\| \leq \Delta. \end{array} \tag{2.11}$$

Demonstração:

Pelo passo 2 do algoritmo, temos que $x^k \in \Omega_{\gamma_k} = \{x \in \mathbb{R}^n \mid g(x) \leq \gamma_k\}$. Pela continuidade de g e por $\gamma_k \to 0$, temos que $x^* \in \Omega$. Seja $z \in \Omega$ tal que $\|z - x^*\| \leq \Delta$. Pela Hipótese S2, temos que existe $K_1 \subseteq K$ tal que $\lim_{k \in K_1} d^k = z - x^*$, onde $d^k \in D_k$. Logo, $\lim_{k \in K_1} x^k + d^k = z$. Como a sequência converge em K, também converge em K_1. Portanto, pela Hipótese S1, temos que $\lim_{k \in K_1} \varphi_k(x^k + d^k) = z$.

A condição (2.9) no passo 2 do algoritmo garante que

$$\begin{aligned} f(x^k) &\leq f(\varphi_k(x^k + d^k)) + \xi_k \\ &\leq f(\varphi_k(x^k + d^k)) + \beta\eta_k, \end{aligned}$$

onde a segunda desigualdade é consequência da limitação de ξ_k. Aplicando o limite para $k \in K_1$ de ambos os lados, pela continuidade de f temos que $f(x^*) \leq f(z)$. Como z foi tomado arbitrariamente, x^* é um minimizador global de (2.11). ∎

O papel de $\varphi_k(\cdot)$ é essencial para a demonstração de convergência do Algoritmo 2.2. Sem a garantia de que os pontos consultados na vizinhança de x^k são viáveis, há exemplos de convergência a pontos que não são minimizadores locais de (2.4). Mesmo com a hipótese de densidade das direções consultadas, ainda é possível construir um exemplo no qual o conjunto torna-se magro mais rapidamente do que as direções de consulta tornam-se viáveis. Esse caso é esquematizado na Figura 2.4. No algoritmo baseado em malhas, a Hipótese M também garantia que o conjunto gordo não era emagrecido mais rapidamente do que a malha era refinada.

Na Figura 2.4 mostramos o que poderia acontecer com o Algoritmo 2.2, caso $\varphi_k(\cdot)$ não fosse utilizada. Primeiramente temos os conjuntos engordados $\Omega_{\gamma_1} \subset \Omega_{\gamma_2} \subset \Omega_{\gamma_3}$. Em cada iteração, consideramos um conjunto insuficiente de amostras dentro do conjunto engordado. Além disso, γ_k está convergindo para zero mais rapidamente do que as direções de consulta conseguem entrar no conjunto γ_k-viável. O resultado é a convergência a um ponto não ótimo.

2.3 Um algoritmo para a fase de minimização

Dadas as hipóteses usadas na demonstração do Teorema 2.7, fica a questão se existe algum algoritmo capaz de satisfazê-las. Nesta seção apresentamos um algoritmo para a fase de minimização do Algoritmo 2.2 que, sob hipóteses razoáveis, possui tais características.

O Algoritmo 2.3 apresentado a seguir parte do pressuposto de que temos bons algoritmos para resolver um problema com conjunto viável gordo (veja a Definição 1.12) e os utiliza para resolver uma sequência de subproblemas engordados, da forma (2.5). Neste momento não descrevemos quais algoritmos são efetivamente utilizados na implementação, deixando essa discussão para o Capítulo 3.

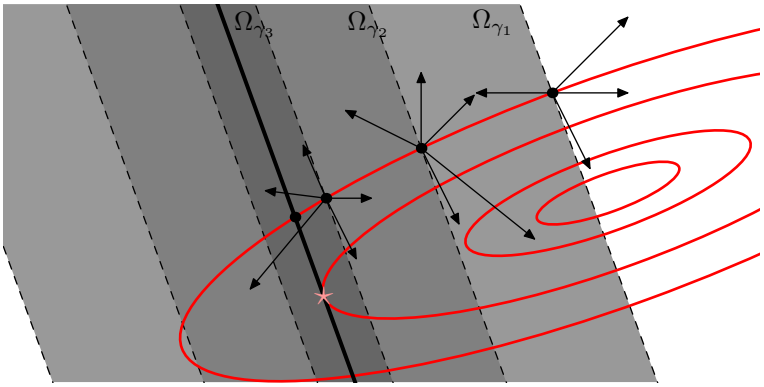


Figura 2.4: Quando as direções consultadas não estão no conjunto γ_k-viável, não há informação suficiente para verificar a otimalidade de x^k. Na figura, as elipses representam as curvas de nível da função, a reta em negrito representa a restrição original e $\star$ representa o minimizador global.

A razão pela qual não há necessidade de apresentarmos qual algoritmo será usado para resolver os subproblemas engordados é que eles não influenciam o funcionamento teórico do Algoritmo 2.3. Por esse motivo, e de forma análoga aos passos de busca e de consulta encontrados em [KLT03] e [ADJ06], o algoritmo para o passo de minimização possui duas fases: a fase de aprimoramento e a fase de consulta.

Na fase de aprimoramento (passo 2), teoricamente irrelevante, buscamos apenas por um ponto $z^{\ell+1}$ no conjunto engordado que reduza a função objetivo. Note que, assim como podemos simplesmente tomar $z^{\ell+1} = z^\ell$ nessa fase, podemos também aplicar um algoritmo bem estabelecido ao subproblema (2.5). Na prática, a fase de aprimoramento tem um papel crucial no bom desempenho do algoritmo.

Na fase de consulta (passo 3), o objetivo é verificar se o ponto corrente z^ℓ satisfaz a condição (2.9). Para isso, construímos o conjunto D_k com direções pseudo-aleatórias. Se z^ℓ não satisfaz a condição desejada, então existe $\varphi_k(z^\ell + d) \in \Omega_{\gamma_k}$ tal que

$$f(\varphi_k(z^\ell + d)) < f(z^\ell) - \eta_{k,\ell},$$

ou seja, um ponto que reduz suficientemente a função objetivo foi encontrado. O novo ponto torna-se o ponto central, descartamos D_k e o processo recomeça. Caso contrário, temos que D_k possui m_k direções que atestam que z^ℓ é o ponto com menor valor funcional dentre elas, ou seja, z^ℓ satisfaz (2.9). O fato das direções serem pseudo-aleatórias tem o objetivo de satisfazer a Hipótese S2.

No passo 3.2 do algoritmo de minimização ocorre a "avaliação" de $\varphi_k(\cdot)$. Quando o ponto $z^\ell + d$ pertence ao conjunto engordado Ω_{γ_k}, nada precisa ser feito. Porém, quando $z^\ell + d \notin \Omega_{\gamma_k}$, a informação fornecida por $f(z^\ell + d)$ não possui utilidade, principalmente se estivermos usando a função penalidade $f_{\Omega_{\gamma_k}}$ no lugar de f. Nesse momento, o papel de φ_k é essencial para que a amostra definida por D_k indique algum tipo de otimalidade de z^ℓ. O que fazemos é associar ao ponto $z^\ell + d$ um novo ponto

Algoritmo 2.3: Algoritmo para a fase de minimização

Dados: $\beta \geq 1$
Entrada: $\gamma_k \geq 0$, $y^k \in \Omega_{\gamma_k}$, $\eta_k > 0$, $\Delta > 0$ e $m_k \in \mathbb{N}^*$
Saída: x^k satisfazendo (2.9)

1 Faça $z^0 = y^k$ e $\ell \leftarrow 0$.

2 **Fase de aprimoramento**

Encontre $z^{\ell+1} \in \Omega_{\gamma_k}$ tal que $f(z^{\ell+1}) \leq f(z^\ell)$.
$\ell \leftarrow \ell + 1$

3 **Fase de consulta**

Escolha $\eta_{k,\ell} \in [\eta_k, \beta\eta_k]$.
$D_k \leftarrow \emptyset$
para $j = 1, \ldots, m_k$ **faça**
3.1 Calcule uma direção pseudo-aleatória $d \in \overline{B(0, \Delta)}$.
3.2 **se** $z^\ell + d \in \Omega_{\gamma_k}$ **então**
Defina $\varphi_k(z^\ell + d) = z^\ell + d$.
senão
Encontre por meio de restauração $\varphi_k(z^\ell + d)$ tal que $\varphi_k(z^\ell + d) \in \Omega_{\gamma_k}$.
Se falhou em encontrar $\varphi_k(z^\ell + d)$, vá para 3.4.
3.3 **se** $f(\varphi_k(z^\ell + d)) < f(z^\ell) - \eta_{k,\ell}$ **então**
$z^{\ell+1} = \varphi_k(z^\ell + d)$
$\ell \leftarrow \ell + 1$
Vá para o passo 2.
3.4 $D_k \leftarrow D_k \cup \{d\}$

4 $x^k \leftarrow z^\ell \quad (\xi_k \leftarrow \eta_{k,\ell})$

$\varphi_k(z^\ell + d)$ que seja γ_k-viável.

A forma como encontramos o novo ponto deve satisfazer a Hipótese S1. Uma estratégia comum para restaurar o ponto inviável é projetá-lo no conjunto viável Ω_{γ_k}. Infelizmente, na prática nem sempre é possível encontrar um ponto restaurado $\varphi_k(z^\ell + d)$. Nesse caso patológico, podemos ver no Algoritmo 2.3 que ainda assim d é adicionada ao conjunto D_k. Embora tal procedimento possa acarretar em um comportamento similar ao ilustrado anteriormente na Figura 2.4, esperamos que isso não aconteça com muita frequência. Por esse motivo, adicionamos a Hipótese R a seguir.

Hipótese R – Se $K \subset \mathbb{N}$ é tal que $\lim_{k \in K} x^k + d^k = z$, onde $z \in \Omega$ e $\{x^k\}$ e $\{d^k\}$, $d^k \in D_k$, foram geradas pelo Algoritmo 2.2 com o Algoritmo 2.3 no passo de minimização, então para k suficientemente grande, o passo 3.2 do Algoritmo 2.3 sempre será capaz de encontrar $\varphi_k(x^k + d^k)$.

Ainda no aspecto prático, há outra vantagem na utilização de $\varphi_k(\cdot)$. Se o

custo computacional de restaurar um ponto inviável for desprezível quando comparado ao custo de avaliar a função objetivo, então muito esforço pode ser aplicado nessa tarefa. Com isso, não é necessário um conjunto D_k com um grande número de elementos para garantir que uma boa vizinhança de z^ℓ foi consultada.

Com a discussão feita até o momento e utilizando a Hipótese R, temos que o Algoritmo 2.3, quando usado no passo de minimização do Algoritmo 2.2, está bem definido. Resta mostrarmos que ele encontra um ponto x^k em um número finito de iterações.

2.8 Proposição
Suponha que as funções f e g são contínuas e que, para todo $\gamma \geq 0$ a função f é limitada inferiormente em Ω_γ. Então um ponto x^k que satisfaz a condição (2.9) *é encontrado em um número finito de iterações.*

Demonstração:

Suponha por contradição que, dada uma iteração k do Algoritmo 2.2, o Algoritmo 2.3 não termina em um número finito de iterações. Pela descrição do algoritmo, isso significa que, para todo $\ell \geq 0$, temos

$$f(z^{\ell+1}) < f(z^\ell) - \eta_{k,\ell} \leq f(z^\ell) - \eta_k.$$

Pela continuidade de f, temos que $\lim_{\ell\to\infty} f(z^\ell) = -\infty$, o que é uma contradição com o fato de f ser limitada inferiormente em Ω_{γ_k}.

Logo, em algum momento o laço do passo 3 do algoritmo de minimização será executado até o fim, devolvendo um ponto x^k de acordo com a condição (2.9). ■

2.4 Considerações adicionais

Algumas observações adicionais devem ser feitas com respeito aos algoritmos 2.2 e 2.3 que foram propostos. Primeiramente, podemos notar que não foi usada hipótese alguma sobre as restrições, além da continuidade. Porém, pelas próprias ideias do algoritmo, é necessário que, para todo ponto, as restrições sejam capazes de devolver um valor numérico associado à viabilidade/inviabilidade. Isso significa que não podemos aplicar a técnica de engordamento a restrições que, dado um ponto, simplesmente indicam se ele PERTENCE ou NÃO PERTENCE ao conjunto viável.

As restrições do tipo PERTENCE/NÃO PERTENCE geralmente são definidas por caixas-pretas e são encontradas na literatura com o nome de virtuais [CST98] ou não-relaxáveis (*unrelaxable*) [ADJ09]. Uma maneira usual de trabalhar com restrições não-relaxáveis é através da função penalidade extrema (2.10), como foi feito em [LT00, ADJ06], pois assim evitamos caminhar por pontos inviáveis.

Através de uma simples modificação, podemos tornar o Algoritmo 2.2 capaz de lidar com restrições não-relaxáveis. Para isso, em primeiro lugar devemos separá-las das restrições relaxáveis. Suponha que o conjunto $\mathcal{X}$ represente as restrições não-relaxáveis, enquanto o conjunto Ω continua como o definimos neste capítulo. Em seguida, ao considerarmos um ponto no conjunto engordado Ω_{γ_k}, devemos também pedir que esteja em $\mathcal{X}$, ou seja, devemos sempre garantir que os pontos estejam em $\Omega_{\gamma_k} \cap \mathcal{X}$. Com essas modificações, se denominarmos por γ_k-viáveis os pontos em $\Omega_{\gamma_k} \cap \mathcal{X}$, todos os resultados teóricos apresentados continuam válidos.

Como não é possível restaurar no conjunto $\mathcal{X}$, tal conjunto deve ser gordo, para que possa ser explorado por métodos de busca direta e pelas direções de consulta do Algoritmo 2.3. Também não podemos permitir que o algoritmo considere pontos fora de $\mathcal{X}$, pois não há técnica que seguramente reencontre pontos viáveis. Finalmente, com a impossibilidade da restauração, o cálculo de $\varphi_k(\cdot)$ torna-se mais complexo, sendo necessário um algoritmo especializado em funções caixas-pretas.

Um exemplo simples de conjunto não-relaxável são restrições de caixa ($l \leq x \leq u$), quando estas representam condições que não podem ser violadas. Dependendo do problema, uma variável que represente a temperatura, por exemplo, nunca pode ser negativa. Um algoritmo que garanta sempre a viabilidade de tais restrições, como o apresentado em [ABMS07], pode ser usado para a restauração nesse caso.

O Algoritmo 2.2, em conjunto com o Algoritmo 2.3, oferece grande liberdade nos métodos que serão utilizados tanto no passo de restauração quanto no passo de melhoria. Com isso, ele é capaz de aproveitar as especificidades de cada problema. Se as derivadas das restrições estão disponíveis, podemos usar um algoritmo de programação não linear para encontrar pontos viáveis. Caso as derivadas não estejam disponíveis, podemos aplicar métodos sem derivadas para sistemas não lineares [ESV11], por exemplo. Em ambos os casos, devemos ter métodos que consigam lidar com restrições não-relaxáveis, caso estas estejam presentes no problema.

No Capítulo 3 apresentamos uma implementação do algoritmo proposto. Mostramos também que seu desempenho depende fortemente dos métodos que são utilizados em cada passo. Os resultados apresentados neste capítulo e no Capítulo 3 também encontram-se no manuscrito [MS11].

Capítulo 3

Algoritmo de engordamento: implementação e testes numéricos

Neste capítulo, discutimos todos os detalhes de implementação do Algoritmo 2.2, utilizando o Algoritmo 2.3 na fase de minimização. O algoritmo resultante dessa implementação será chamado, a partir deste momento, de *Skinny*, em alusão ao tipo de restrições para as quais foi desenvolvido.

Não estamos interessados em problemas com conjuntos viáveis gordos, pois, em tais casos, o algoritmo proposto claramente resolverá uma série de subproblemas desnecessários, enquanto que um algoritmo eficiente poderia ser aplicado diretamente ao problema original. Em virtude disso, realizamos diversos experimentos numéricos e comparações com outros algoritmos existentes na literatura e capazes de lidar com conjuntos magros.

Na Seção 3.1 relembramos rapidamente o algoritmo de engordamento e o algoritmo para a fase de minimização. Discutimos detalhadamente o que é feito em cada passo e qual a finalidade da escolha proposta. Na Seção 3.2 realizamos testes preliminares para escolher dois parâmetros considerados importantes em *Skinny* e para os quais não há uma escolha natural. Por fim, na Seção 3.3 comparamos *Skinny* com outros 3 algoritmos existentes na literatura e capazes de lidar (de forma heurística ou não) com conjuntos viáveis magros. Problemas padrão na literatura e outros com apelo geométrico são utilizados para mostrar as vantagens da nova abordagem.

3.1 Implementação

Começamos por relembrar o problema a ser resolvido:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & g(x) \leq 0 \\ & x \in \mathcal{X}, \end{array} \tag{3.1}$$

com $f : \mathbb{R}^n \to \mathbb{R}$ e $g : \mathbb{R}^n \to \mathbb{R}^m$. Denotamos por $\Omega \doteq \{x \in \mathbb{R}^n \mid g(x) \leq 0\}$ o conjunto das restrições que podem ser relaxadas e por $\mathcal{X}$ as restrições não-relaxáveis no

Algoritmo 3.1: Algoritmo de Engordamento (Algoritmo 2.2)

Dados: $x^0 \in \mathbb{R}^n$, $\beta \geq 1$, $\gamma_k \geq 0$ tal que $\{\gamma_k\} \to 0$, $\eta_k > 0$ tal que $\{\eta_k\} \to 0$ e $\{m_k\}$ limitada tal que $m_k \in \mathbb{N}^*$.

$k \leftarrow 1$

1. **Fase de restauração**
   Escolha $y^k \in \Omega_{\gamma_k} \cap \mathcal{X}$.
2. **Fase de minimização**
   Dado $\xi_k \in [\eta_k, \beta\eta_k]$ e utilizando y^k, escolha $x^k \in \Omega_{\gamma_k}$ tal que

$$f(x^k) \leq f(\varphi_k(x^k + d)) + \xi_k, \tag{3.2}$$

   para todo $d \in D_k$, onde $D_k = \{d^i \in \mathbb{R}^n, i = 1, \ldots, m_k \mid \varphi_k(x^k + d^i) \in \Omega_{\gamma_k}\}$.
   (O operador φ_k é definido de forma que $\varphi_k(x^k + d) \in \Omega_{\gamma_k}$ para todo $d \in D_k$.)
3. $k \leftarrow k + 1$ e volte para 1.

sentido de [ADJ09]. Na presente implementação, o conjunto $\mathcal{X}$ é dado pelas restrições de caixa ($l \leq x \leq u$), pois estas não são relaxadas pelo algoritmo e podem ser tratadas eficientemente com o algoritmo usado na fase de restauração.

Como definido anteriormente, o conjunto Ω engordado por γ_k é definido por

$$\Omega_{\gamma_k} = \{x \in \mathbb{R}^n \mid g(x) \leq \gamma_k\}$$

e um ponto x é γ_k-viável se $x \in \Omega_{\gamma_k} \cap \mathcal{X}$. Caso haja restrições de igualdade $h(x) = 0$, estas podem ser substituídas por duas desigualdades $h(x) \leq 0$ e $-h(x) \leq 0$, sem perda das propriedades teóricas e ocasionando apenas um aumento no número de restrições.

Optamos por colocar novamente neste capítulo o algoritmo principal, Algoritmo 2.2, e o algoritmo utilizado na fase de minimização, Algoritmo 2.3. Esses algoritmos estão representados pelos algoritmos 3.1 e 3.2, respectivamente. Basicamente, o processo de implementação de *Skinny*, consiste na implementação cuidadosa da fase de restauração do Algoritmo 3.1 e dos passos descritos no Algoritmo 3.2. Como mencionado na Seção 2.4, adicionamos o conjunto $\mathcal{X}$ na descrição dos algoritmos. As restrições que compõem $\mathcal{X}$ sempre são satisfeitas.

3.1.1 A fase de restauração

A fase de restauração (passo 1 do Algoritmo 3.1) não é teoricamente necessária para a convergência de *Skinny*, mas tem uma grande importância na prática. Muitos algoritmos (veja os algoritmos da Seção 1.1) que minimizam funções sem derivadas em conjuntos gordos necessitam que o ponto inicial seja viável. Com um ponto inicial viável em mãos, cada passo desses algoritmos é cuidadosamente tomado para que não caminhem em pontos inviáveis, pois seria necessário algum tipo de restauração.

Algoritmo 3.2: Algoritmo para a fase de minimização (Algoritmo 2.3).

Dados: $\beta \geq 1$
Entrada: $\gamma_k \geq 0$, $y^k \in \Omega_{\gamma_k}$, $\eta_k > 0$, $\Delta > 0$ e $m_k \in \mathbb{N}^*$
Saída: x^k satisfazendo (3.2)

1 Faça $z^0 = y^k$ e $\ell \leftarrow 0$.

2 **Fase de aprimoramento**

Encontre $z^{\ell+1} \in \Omega_{\gamma_k} \cap \mathcal{X}$ tal que $f(z^{\ell+1}) \leq f(z^\ell)$.
$\ell \leftarrow \ell + 1$

3 **Fase de consulta**

Escolha $\eta_{k,\ell} \in [\eta_k, \beta\eta_k]$.
$D_k \leftarrow \emptyset$
para $j = 1, \ldots, m_k$ **faça**
3.1 Calcule uma direção pseudo-aleatória $d \in \overline{B(0,\Delta)}$.
3.2 **se** $z^\ell + d \in \Omega_{\gamma_k} \cap \mathcal{X}$ **então**
Defina $\varphi_k(z^\ell + d) = z^\ell + d$.
senão
Encontre por meio de restauração $\varphi_k(z^\ell + d)$ tal que $\varphi_k(z^\ell + d) \in \Omega_{\gamma_k}$.
Se falhou em encontrar $\varphi_k(z^\ell + d)$, vá para o passo 3.4.
3.3 **se** $f(\varphi_k(z^\ell + d)) < f(z^\ell) - \eta_{k,\ell}$ **então**
$z^{\ell+1} = \varphi_k(z^\ell + d)$
$\ell \leftarrow \ell + 1$
Vá para o passo 2.
3.4 $D_k \leftarrow D_k \cup \{d\}$

4 $x^k \leftarrow z^\ell \quad (\xi_k \leftarrow \eta_{k,\ell})$

Quando o conjunto Ω do problema (3.1) é vazio, nada pode ser feito na restauração, dado que não podemos quantificar a viabilidade associada às restrições de $\mathcal{X}$. Nesse caso, precisamos de informações adicionais (vindas, por exemplo, por conhecedores do problema a ser resolvido) para conseguir pontos viáveis em $\mathcal{X}$. No caso de restrições de caixa, podemos simplesmente projetar o ponto inviável na caixa.

Por outro lado, quando há restrições que podem ser relaxadas, a fase de restauração permite que aproveitemos características especiais do problema. No caso particular em que temos as derivadas das restrições, dado um ponto $x^{k-1} \notin \Omega_{\gamma_k}$, o ponto restaurado $y^k \in \Omega_{\gamma_k} \cap \mathcal{X}$ é dado pela solução do seguinte problema de otimização:

$$\begin{aligned} \min \quad & \|y - x^{k-1}\|_2^2 \\ \text{s. a} \quad & g(y) \leq \tau\gamma_k \\ & y \in \mathcal{X}, \end{aligned} \tag{3.3}$$

onde $\tau \in [0,1]$ controla o esforço desejado para a restauração. A escolha $\tau = 0$ força y^k a estar contido no conjunto viável original. Dado que consideramos $\mathcal{X} = \{x \in$

$\mathbb{R}^n \mid l \leq x \leq u\}$, o problema (3.3) é um problema de programação não linear e pode ser resolvido com algoritmos eficientes. Em *Skinny*, utilizamos o algoritmo baseado em Lagrangianos Aumentados denominado Algencan [ABMS07, TAN], discutido na Seção 1.5. Os parâmetros padrão de Algencan foram utilizados na implementação, com critério de parada por otimalidade dado por 10^{-8} e o critério de viabilidade dado por $\varepsilon_{\text{viab}}$, a ser definido posteriormente nos testes comparativos.

Se as derivadas das restrições não estão disponíveis, para encontrar $y^k \in \Omega_{\gamma_k} \cap \mathcal{X}$ resolvemos o seguinte problema sem derivadas:

$$\begin{array}{ll} \min & \sum_{i=1}^{m} \max\{0, g_i(y) - \tau\gamma_k\}^2 \\ \text{s. a} & y \in \mathcal{X} \end{array} \tag{3.4}$$

com o Algoritmo 1.1 (busca coordenada), porém qualquer algoritmo sem derivadas pode ser utilizado. Uma outra forma de obter y^k nesse caso é através de algoritmos próprios para sistemas não lineares sem derivadas [ESV11]. O critério de parada para a busca coordenada é o refinamento da malha, quando este torna-se menor do que 10^{-8}. Também interrompemos o algoritmo caso $1000n$ avaliações da função objetivo de (3.4) tenham sido feitas.

O ponto inicial utilizado nos problemas (3.3) e (3.4) é o próprio ponto x^{k-1}, calculado na iteração anterior, ou x^0, fornecido na primeira iteração. O valor exato do parâmetro τ é discutido na Seção 3.2.

3.1.2 A fase de aprimoramento

A ideia principal da fase de aprimoramento, representada pelo passo 2 do Algoritmo 3.2, é utilizar qualquer meio eficiente para conseguir um ponto com melhor valor funcional dentro do conjunto $\Omega_{\gamma_k} \cap \mathcal{X}$. Assim como a fase de restauração, essa fase também é irrelevante do ponto de vista teórico, mas suas vantagens na prática são indispensáveis.

O ideal para essa fase é utilizar um algoritmo que consiga lidar com certa eficiência os problemas sem derivadas com restrições gordas. Na implementação de *Skinny*, utilizamos o algoritmo DFO [CST97b], discutido na Seção 1.6. Como já mencionamos, DFO não possui teoria de convergência para o caso restrito. Porém, ele foi especialmente implementado para considerar restrições em problemas sem derivadas (processo descrito em [CST98]) e é capaz de resolver eficientemente esses problemas, desde que tenhamos algoritmos eficientes para resolver os subproblemas quadráticos com restrições. No caso em que as restrições dos problemas possuem derivadas, tais subproblemas são problemas de programação não linear, motivo pelo qual também utilizamos Algencan para resolvê-los. Como mostramos na Seção 3.3, DFO em conjunto com Algencan apresentou excelentes resultados nos problemas-teste de Hock e Schittkowski, que motivaram seu uso na fase de aprimoramento.

Para acelerar a convergência de *Skinny* a um ponto do conjunto viável original, combinamos a função objetivo original com uma penalização fixa das restrições

de Ω, definida a seguir:

$$\bar{f}(x) = f(x) + \rho\|g(x)_+\|_2^2, \tag{3.5}$$

onde $[g(x)_+]_i = \max\{0, g_i(x)\}$, $i = 1, \ldots, m$, e

$$\rho = \max\{1, |f(x^0)|\} / \max\{1, \|g(x^0)\|_\infty\} \tag{3.6}$$

representa o parâmetro fixo de penalidade. Por causa do parâmetro fixo, o problema:

$$\begin{array}{ll} \min & \bar{f}(x) \\ \text{s. a} & g(x) \leq 0 \\ & x \in \mathcal{X} \end{array} \tag{3.7}$$

é equivalente ao problema (3.1). Supondo que a avaliação de f é consideravelmente mais custosa do que a avaliação das restrições, o custo da avaliação de $\bar{f}$ continua vinculado apenas ao custo de f.

Por essa razão, o problema (3.7) é o problema realmente considerado por *Skinny*, sobre o qual aplicamos a técnica de engordamento descrita no Capítulo 2. Observamos que a utilização de (3.7) em lugar de (3.1) resultou, de fato, na aceleração do emagrecimento do conjunto nas primeiras iterações. Isso significa que a penalização fixa das restrições relaxáveis atua como um corretor dos engordamentos que estão em desacordo com a realidade de tais restrições. Observe a atuação da função $\bar{f}$ no funcionamento de *Skinny*, ilustrada no Exemplo 3.1.

3.1 Exemplo (Penalidade fixa)
Suponha que deseja-se minimizar $f(x) = x$ sujeita à restrição $x = 1$. Os problemas engordados definidos por (2.5) *(sem penalização fixa) e por* (3.7) *(com penalização fixa) são, respectivamente:*

$$\begin{array}{ll} \min & x \\ \text{s. a} & 1 - \gamma_1 \leq x \leq 1 + \gamma_1 \end{array} \quad e \quad \begin{array}{ll} \min & x + \rho(1-x)^2 \\ \text{s. a} & 1 - \gamma_1 \leq x \leq 1 + \gamma_1 \end{array}$$

onde ρ é um valor fixo. A penalização fixa foi simplificada por motivos ilustrativos, pois, na realidade, as igualdades são definidas como duas desigualdades.

Pode-se observar na Figura 3.1(a) que, sem a penalização fixa, os minimizadores globais dos subproblemas engordados da forma (2.5) *encontram-se sempre no limite esquerdo do engordamento, ou seja, $x^k = 1 - \gamma_k$. Como $\gamma_k \to 0$, a solução $x^* = 1$ é encontrada. Por outro lado, no caso da penalidade fixa, ilustrada na Figura 3.1(b), o minimizador do subproblema engordado da forma* (3.7) *na primeira iteração já está suficientemente dentro do intervalo definido por γ_2. Logo, a penalidade fixa tem a função de corrigir o valor inicial de γ ao problema. Depois da primeira iteração, ajusta-se o valor de γ_2 para que x^1 seja γ_2-inviável (e estimule a procura por y^2 que seja γ_2-viável) e Skinny procede normalmente.*

Os subproblemas resolvidos por DFO são definidos pelo engordamento do problema (3.7) de maneira análoga ao subproblema (2.5). Além de DFO, *Skinny* é capaz de utilizar os seguintes algoritmos na fase de aprimoramento: busca coordenada, BOBYQA [Pow09] e Nelder–Mead [NM65]. Nenhum deles, todavia, tem a habilidade de lidar explicitamente com restrições, como o faz DFO. Para adaptá-los, é necessário

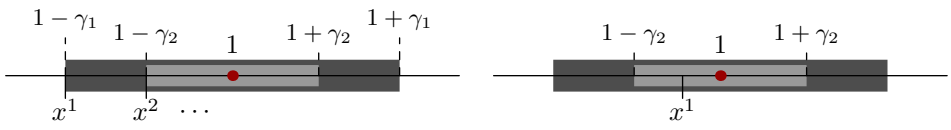


Figura 3.1: Diferença entre (a) sem penalização fixa e (b) com penalização fixa.

iniciar com um ponto γ_k-viável (razão para a existência da fase de restauração) e usar a função penalidade extrema aplicada à função $\bar{f}$, definida em (3.5):

$$\bar{f}_{\Omega_{\gamma_k}}(x) = \begin{cases} f(x) + \rho\|g(x)_+\|_2^2, & \text{se } x \in \Omega_{\gamma_k} \cap \mathcal{X} \\ 10^{99}, & \text{caso contrário,} \end{cases} \tag{3.8}$$

resolvendo o problema irrestrito de minimizar $\bar{f}_{\Omega_{\gamma_k}}$. A função penalidade extrema tenta impedir que os algoritmos saiam dos conjuntos Ω_{γ_k} e $\mathcal{X}$.

Um algoritmo baseado em busca direta, como a busca coordenada, não é prejudicado com o uso da função penalidade extrema. Isso se deve ao fato de que, supondo que ele parta de um ponto viável, sempre que um ponto com valor penalizado é consultado, este será automaticamente descartado pois o ponto corrente possui valor funcional menor. Esse é o motivo pelo qual necessitamos da fase de restauração em *Skinny*. Por outro lado, podemos esperar que o uso da função penalidade extrema cause grandes deformidades em métodos que trabalham com modelos da função objetivo. Isso porque, diferentemente da busca coordenada que nunca considera pontos γ_k-inviáveis, tais algoritmos vão inserir em seus modelos pontos extremamente penalizados.

No algoritmo Nelder–Mead, o simplex pode se tornar totalmente inútil se contiver muitos pontos com um valor extremo de função objetivo. De forma similar, os modelos quadráticos construídos por BOBYQA de forma alguma representam a função objetivo $\bar{f}$ se muitos pontos γ_k-inviáveis estão presentes no conjunto interpolador. Podemos construir um exemplo no qual todos os pontos são inviáveis e, consequentemente, o modelo em nada reflete a função objetivo, podendo resultar em um minimizador inviável. Felizmente, no caso de BOBYQA podemos fazer uma alteração que o torne semelhante ao algoritmo DFO (embora essa alteração artificial não resulte em um desempenho efetivamente superior). Na próxima subseção, descrevemos brevemente BOBYQA e apresentamos uma modificação que melhora a qualidade dos modelos quadráticos com os quais ele trabalha no caso de restrições gordas.

Cada algoritmo que pode ser utilizado na fase de aprimoramento possui parâmetros que modificam seu funcionamento. No caso de DFO mantivemos todos os parâmetros originais. Nos outros algoritmos, vimos a necessidade de associar alguns de seus parâmetros ao parâmetro γ_k, que controla o "relaxamento" do conjunto magro. Tal procedimento foi essencial para o seu funcionamento adequado, principalmente porque eles não foram desenvolvidos para o caso restrito.

Todos os algoritmos, incluindo DFO, foram associados ao critério de parada dado por $\varepsilon_{\text{otim}}$, que controla o esforço que devem fazer em busca de um minimizador do

problema engordado. Nos experimentos numéricos apresentados na Seção 3.3 utilizamos $\varepsilon_{\text{otim}} = 10^{-3}$ (padrão na maioria dos algoritmos comparados) ou $\varepsilon_{\text{otim}} = 10^{-8}$, de acordo com o que desejamos analisar.

3.1.3 Modificações em BOBYQA

Experimentos numéricos indicam que o algoritmo BOBYQA [Pow09] é capaz de resolver problemas de otimização sem derivadas com restrições de caixa usando um número bastante reduzido de avaliações da função objetivo. A ideia principal de BOBYQA é construir modelos quadráticos da função utilizando uma quantidade pré-fixada de pontos nos quais a função deve ser avaliada. Em cada passo do método, o modelo é minimizado sujeito às restrições de caixa e esse minimizador é o candidato a substituir um dos pontos já existentes para a construção do novo modelo. Somente quando já foi decidido que o ponto irá entrar no conjunto de pontos a função objetivo original é avaliada nesse ponto.

BOBYQA não foi projetado para restrições gerais, por isso neste trabalho utilizamos a função penalidade extrema e tratamos um problema geral como sendo irrestrito ou apenas com restrições de caixa. Claramente, qualquer algoritmo irrestrito aplicado à função penalidade extrema não funcionará quando o conjunto viável for magro, como já mencionamos anteriormente, o que motiva a utilização do algoritmo de engordamento. Porém, outro inconveniente surge no uso dessa função, oriundo da construção do modelo quadrático. A aproximação quadrática é feita com os valores da função calculada em um conjunto de pontos escolhido pelo algoritmo, independentemente se esse valor é o verdadeiro valor de f ou é o valor com a penalização extrema. Consequentemente, o modelo pode deformar-se e levar a pontos γ_k-inviáveis.

A correta observação do modo como os pontos para a aproximação inicial são escolhidos em BOBYQA é essencial para o bom desempenho do algoritmo com a função penalidade extrema. Dado um ponto inicial, os pontos utilizados para o modelo basicamente estão a uma distância `rhobeg` dele, na direção dada pelos vetores canônicos (e seus respectivos opostos). No caso em que o usuário desejar mais do que $2N + 1$ pontos, uma outra maneira é usada para os demais, seguindo um processo semelhante. Em ambos os casos, os pontos são escolhidos para satisfazer as restrições de caixa. Para o nosso algoritmo de engordamento, é necessário fazer com que `rhobeg` decresça de forma proporcional ao emagrecimento do conjunto (definido pelo parâmetro γ_k), fazendo com que os pontos para o modelo inicial estejam dentro do conjunto $\Omega_{\gamma_k} \cap \mathcal{X}$ e, desta forma, associados com os valores da função objetivo original, e não com o valor da penalidade extrema.

Um outro processo de BOBYQA também pode ser melhorado: o modo como o problema formado pela aproximação quadrática é resolvido. No algoritmo original, um método do tipo gradientes conjugados truncado é utilizado em conjunto com uma região de confiança. Dado o modelo Q_ℓ, o objetivo é encontrar d tal que $Q_\ell(x^\ell + d) < Q_\ell(x^\ell)$ e tal que $x^\ell + d$ satisfaça as restrições de caixa, com ℓ representando as iterações de BOBYQA e não do algoritmo de engordamento. Nossa modificação foi utilizar o

algoritmo de programação não linear Algencan para resolver o seguinte problema:

$$\begin{array}{ll} \min\limits_{d} & Q(x^\ell + d) \\ \text{s. a} & x^\ell + d \in \Omega_{\gamma_k} \\ & x \in \mathcal{X} \\ & \|d\|_2^2 \leq \Delta_\ell, \end{array} \tag{3.9}$$

onde Ω_{γ_k} representa o conjunto engordado no passo k de *Skinny*. Dessa forma, garantimos que toda solução do problema quadrático seja um ponto viável do subproblema engordado e consequentemente o valor da função penalidade extrema em $x^\ell + d$ será o valor da função verdadeira.

Lembremos que, de acordo com as hipóteses primordiais a respeito dos problemas para os quais o algoritmo de engordamento foi desenvolvido, embora a utilização de Algencan nesse processo aumente muito o número de avaliações das restrições, o algoritmo não é prejudicado, pois as restrições são consideradas computacionalmente baratas de serem calculadas. A utilização de Algencan só é possível em BOBYQA, assim como em DFO, porque as restrições têm derivadas disponíveis e $\mathcal{X}$ é dado por restrições de caixa.

Apesar das modificações feitas, BOBYQA realiza outros procedimentos, dentre eles um passo de Cauchy quando o ponto encontrado na resolução do problema (3.9) não satisfaz determinados critérios. Esses outros processos podem resultar em pontos γ_k-inviáveis inseridos no modelo, motivo pelo qual, mesmo com as modificações feitas, os resultados encontrados com BOBYQA foram inferiores aos encontrados por DFO.

3.1.4 A fase de consulta

A fase de consulta é a única fase responsável por toda a teoria de convergência de *Skinny*. Por carregar essa responsabilidade consigo, esta fase pode ser a causadora de um número bastante elevado de avaliações da função objetivo. Isso ocorre porque precisamos consultar um conjunto suficientemente grande de direções na bola de raio Δ, tentando simular uma construção de direções densas. Quanto maior o número de direções consultadas, mais próximos estamos da teoria e maior é o esforço computacional necessário. Essa troca entre qualidade da solução e esforço computacional deve ser medida e analisada por cada usuário, com o objetivo de resolver seu problema da melhor forma possível.

Embora na descrição dos algoritmos 3.1 e 3.2 apareça a função f, a função objetivo usada nesta fase é a função penalidade extrema com o parâmetro fixo de penalidade da inviabilidade, descrita por (3.8). Dado que a fase de consulta é um tipo de busca direta direcional, não desejamos que pontos γ_k-inviáveis sejam considerados nas comparações. Para evitar que as expressões fiquem muito carregadas, vamos utilizar a função f no lugar de $\bar{f}_{\Omega_{\gamma_k}}$, lembrando que tanto a penalidade fixa da inviabilidade quanto a penalidade extrema são apenas técnicas para trabalhar com restrições na prática e não influenciam no funcionamento teórico do algoritmo.

Na implementação da fase de consulta precisamos: (i) gerar conjuntos de direções de consulta tais que sua união seja densa na bola de raio Δ (Hipótese S2); e (ii) avaliar o "operador" $\varphi_k(\cdot)$ de forma que satisfaça a Hipótese S1, para $k \in \mathbb{N}^*$.

Para satisfazer (i), geramos direções pseudo-aleatórias na bola de raio Δ. Como vimos no Capítulo 2, qualquer norma pode ser usada para definir a bola. Para a criação dos números pseudo-aleatórios que formam as direções utilizamos o método de Schrage [Sch79]. A utilização de geradores de números pseudo-aleatórios é conhecida por ter uma grande desvantagem: não é possível gerar números realmente aleatórios e, em algum momento, os números gerados vão começar a se repetir, pois são descritos por uma sequência de passos bem definida. Tal desvantagem, ao menos para a finalidade de *Skinny*, é uma grande vantagem pois, pelo fato dos números serem gerados por passos bem definidos, dada uma mesma semente inicial, a sequência gerada será a mesma. Desta forma, *Skinny* utilizado duas vezes para um mesmo problema, em um mesmo ambiente, terá os mesmos resultados. Além disso, os geradores de números aleatórios bem estabelecidos na literatura (como o de Schrage) conseguem criar uma boa aleatoriedade de pontos antes de entrarem em repetição, desde que seja utilizada uma semente inicial razoável. A impossibilidade de repetir resultados foi o motivo pelo qual o algoritmo OrthoMads [AADJLD09] foi elaborado e substituiu a implementação aleatória LtMads [ADJ06] para a classe de algoritmos Mads.

Verificamos 3 maneiras de gerar direções pseudo-aleatórias:

1. com a norma $\|\cdot\|_\infty$ (distribuição uniforme no hipercubo de lados Δ);

2. com a norma $\|\cdot\|_2$ indiretamente, gerando uma direção v uniformemente no hipercubo de lados Δ e excluindo as direções tais que $\|v\|_2 > \Delta$;

3. com a norma $\|\cdot\|_2$ diretamente, através da metodologia combinada por [Mul59] e [BD93], descrita no Algoritmo 3.3.

A segunda metodologia é reconhecidamente ineficiente, pois a razão entre o volume da bola de raio Δ e do hipercubo de lados Δ vai para zero com o aumento da dimensão do espaço. Após alguns testes preliminares, verificamos que as metodologias 1 e 3 resultaram em comportamentos semelhantes e optamos pela terceira metodologia, descrita no Algoritmo 3.3, para que essa interessante técnica possa ser difundida.

Em [BM58] foi apresentada uma técnica para gerar pontos pseudo-aleatórios que seguem a distribuição normal utilizando pontos pseudo-aleatórios dados pela distribuição uniforme. Em [Mul59] afirmou-se que um vetor cujas n coordenadas são dadas por n distribuições normais independentes, quando normalizado, é um vetor uniformemente distribuído na hiperesfera unitária. No Algoritmo 3.3, combinamos essas duas técnicas e as estendemos com conhecidos resultados de probabilidade (veja [BD93]) para gerar pontos pseudo-aleatórios na bola de raio Δ.

Com uma direção pseudo-aleatória d na bola de raio Δ em mãos (dada pelo Algoritmo 3.3), temos que definir $\varphi_k(z^\ell + d)$ e decidir se adicionamos ou não d no conjunto D_k. Se $z^\ell + d \in \Omega_{\gamma_k} \cap \mathcal{X}$ nada precisa ser feito e definimos $\varphi_k(z^\ell + d) = z^\ell + d$,

Algoritmo 3.3: Geração de direções pseudo-aleatórias.

Entrada: $n \geq 1$, $\Delta > 0$

Saída: $d \in \mathbb{R}^n$ uniformemente distribuída na bola Euclidiana de raio Δ.

1 $i \leftarrow 1$

2 **enquanto** $i \leq n$ **faça**

  Gere dois números $a \sim U(0,1)$ e $b \sim U(0,1)$ com o algoritmo de Schrage.

  $v_i = \sqrt{-2\ln(a)}\cos(2\pi b)$

  $i \leftarrow i + 1$

  se $i \leq n$ **então**

    $v_i = \sqrt{-2\ln(a)}\sin(2\pi b)$

    $i \leftarrow i + 1$

3 Gere $c \sim U(0,1)$ com o algoritmo de Schrage.

$d = \left(\Delta c^{1/n}\right) \dfrac{v}{\|v\|_2}$

caso contrário um dos seguintes problemas de restauração é resolvido:

$$\begin{array}{lll} \min & \|y - (z^\ell + d)\|_2^2 & \\ \text{s. a} & g(y) \leq \gamma_k & \text{ou} \\ & y \in \mathcal{X} & \end{array} \qquad \begin{array}{lll} \min & \sum_{i=1}^m \max\{0, g_i(y) - \gamma_k\}^2 & \\ & y \in \mathcal{X}\,. & \end{array} \tag{3.10}$$

Para resolvê-los, repetimos as mesmas técnicas utilizadas para os problemas de restauração com derivadas das restrições (3.3), ou sem derivadas das restrições (3.4). Se uma solução y^* é encontrada, definimos $\varphi_k(z^\ell + d) = y^*$.

Mesmo que o problema de restauração considerado em (3.10) tenha solução, há a possibilidade de não encontrarmos sequer um ponto viável, devido a dificuldades numéricas. Nesse caso patológico, definimos $\varphi_k(z^\ell + d) = z^\ell + d$, o que acarreta em $f(\varphi_k(z^\ell + d)) = 10^{99}$ (com a função penalidade extrema) e em nenhuma informação adicional sobre a vizinhança de z^ℓ. Ainda nesse caso, adicionamos a direção d ao conjunto D_k. A explicação para tal atitude é que talvez toda a dificuldade em encontrar um ponto γ_k-viável para ser consultado esteja associada a algum tipo de otimalidade no ponto z^ℓ. Reiteramos que se este comportamento ocorrer com muita frequência durante a execução de *Skinny*, pode causar convergência a um ponto que não é minimizador ou sequer é um ponto viável, como mencionado no Capítulo 2.

No caso não patológico, adicionamos d ao conjunto D_k sempre que

$$f(\varphi_k(z^\ell + d)) \geq f(z^\ell) - \eta_{k,\ell} \geq f(z^\ell) - \beta\eta_k,$$

onde η_k controla o decréscimo suficiente. Caso algum ponto consultado possua valor funcional suficientemente menor do que $f(z^\ell)$, então todas as direções de D_k são descartadas, o ponto corrente z^ℓ é substituído por $\varphi_k(z^\ell + d)$ e um novo conjunto D_k é construído. Esse processo é repetido até que D_k possua m_k direções. O objetivo final

da fase de consulta é garantir que o ponto x^k devolvido pelo Algoritmo 3.2 satisfaça, para todo $d \in D_k$:

$$f(x^k) \leq f(\varphi_k(x^k + d)) + \beta\eta_k.$$

O termo η_k controla o decréscimo suficiente, necessário para a convergência de *Skinny* e para a boa definição do Algoritmo 3.2, e deve ser atualizado de forma que convirja para zero conforme $k \to \infty$. Observe que o parâmetro η_k é um dado de entrada do Algoritmo 3.2 e permanece fixo durante as iterações internas do mesmo.

Na prática, porém, percebemos que nas iterações iniciais podemos exigir um decréscimo maior do que nas iterações finais. Além disso, entendemos que quanto mais próximo do conjunto viável original (γ_k próximo de 0), mais próximo *Skinny* está de um possível minimizador, e o decréscimo funcional deve ser pequeno. Por esse motivo, utilizamos o parâmetro fixo $\beta \geq 1$. Para todo $k \in \mathbb{N}^*$, definimos um decréscimo fixo para η_k dado por

$$\eta_k = \frac{10^{-8}}{k}. \tag{3.11}$$

Escolhemos $\beta = 10^{16}$, pois ao pedirmos

$$\eta_{k,\ell} \in [\eta_k, \beta\eta_k]$$

temos uma grande liberdade de escolha, o que nos permite aproveitar as informações obtidas iterativamente, tanto dos valores fornecidos na entrada (η_k e γ_k), quanto dos valores encontrados nas iterações (internas) do Algoritmo 3.2. Assim, definimos $\eta_{k,\ell}$, para $k \geq 1$ e $\ell \geq 1$, da seguinte forma

$$\eta_{k,\ell} = \max\{\eta_k, \min\{\|g(z^\ell)\|_\infty, \eta'_k, \beta\eta_k\}\}, \tag{3.12}$$

onde

$$\eta'_k = \begin{cases} 0.01 \max\{1, |f(x^0)|\}, & \text{se } k = 1 \\ \min\{0.5\eta'_{k-1}, \gamma_k\}, & \text{se } k > 1 \end{cases}$$

é independente de ℓ.

Com a atualização dada por (3.12), asseguramos as condições teóricas de convergência para o decréscimo suficiente, dado que η_k tende inexoravelmente a zero, por (3.11). Ainda na fórmula (3.12), utilizamos $\|g(z^\ell)\|_\infty$ em função do explicado no parágrafo anterior, pois se z^ℓ é viável, não esperamos um decréscimo muito grande na sua vizinhança, a menos que as restrições sejam irrelevantes no problema. Por fim, o termo η'_k tem a função de relacionar $\eta_{k,\ell}$ com o parâmetro de engordamento γ_k que, como veremos em seguida, é o principal critério de parada de *Skinny*.

3.1.5 O parâmetro de engordamento

O parâmetro de engordamento γ_k permite que um problema do tipo (3.1), com restrições magras relaxáveis do tipo $g(x) \leq 0$, seja resolvido com algoritmos bem estabelecidos que lidam com conjuntos gordos. Intuitivamente, sabemos que γ_k deve

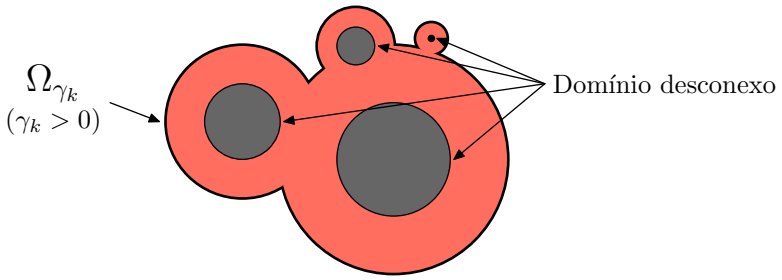


Figura 3.2: Conectando as partes de um domínio desconexo através de um engordamento adequado.

convergir para zero, para que no final o ponto devolvido por *Skinny* seja viável com respeito às restrições relaxáveis do problema original (3.1).

A relação de γ_k com a viabilidade dos iterados é evidente, por isso, pode ser usado como indicador de parada de *Skinny*. Se γ_k está suficientemente próximo de zero, temos que o ponto x^k é viável de acordo com alguma tolerância. Se o algoritmo usado na fase de minimização for cuidadoso na busca por minimizadores dos subproblemas engordados, também esperamos que x^k seja um minimizador do problema original.

A menos que os minimizadores de (3.1) estejam no interior do conjunto viável original, é de se esperar que os minimizadores dos subproblemas engordados sejam inviáveis, só atingindo a viabilidade quando $\gamma_k = 0$. Com a utilização da penalidade fixa, discutida na Subseção 3.1.2 e ilustrada na Figura 3.1, em associação a uma atualização inteligente de γ_k, aceleramos um pouco o processo de convergência a pontos viáveis. Para isso, definimos γ_k inicialmente da seguinte maneira:

$$\gamma_1 = \max\{\gamma_{\text{ini}}, \|g(x^0)\|_\infty\}, \tag{3.13}$$

ou seja, o engordamento inicial está associado ao valor funcional das restrições g no ponto inicial x^0 e, no caso em que $g(x^0) = 0$, ao parâmetro γ_{ini}. Dependendo do problema, uma boa escolha de γ_{ini} é essencial para eliminar a complexidade do conjunto viável original. A Figura 3.2 mostra que um valor adequado de engordamento pode conectar conjuntos viáveis desconexos, por exemplo. Na presente implementação, tomamos $\gamma_{\text{ini}} = 10$.

No final da iteração $k \geq 1$, definimos

$$\gamma_{k+1} = \max\{\varepsilon_{\text{viab}}, \theta \min\{\gamma_k, \|g(x^k)\|_\infty\}\}, \tag{3.14}$$

onde θ define a taxa de emagrecimento do conjunto e $\varepsilon_{\text{viab}}$ é a tolerância máxima que permitimos para a inviabilidade. Valores de θ próximos de zero aceleram a convergência do conjunto Ω_{γ_k} ao conjunto original Ω, mas podem fazer com que os iterados fiquem distantes uns dos outros, o que costuma trazer inconvenientes na prática. Por outro lado, valores de θ próximos de 1 resultam em um grande número de avaliações da função objetivo.

A equação (3.13) garante ainda que $g(x^0) \leq \gamma_1$, ou seja, que nada precise ser feito na primeira restauração. A atualização dada por (3.14) garante que, em todo o início de iteração $k > 1$, o ponto x^{k-1} encontrado na iteração anterior seja γ_k-inviável (a menos que x^{k-1} já seja viável) e deve ser restaurado. Nesse momento, é interessante que o problema de projeção (3.3) seja resolvido, na tentativa de mantermos a otimalidade da iteração anterior.

Dada a importância do parâmetro de engordamento, θ deve ser escolhido cuidadosamente. Por isso, na Seção 3.2 variamos esse parâmetro em conjunto com o parâmetro τ (também considerado importante) e verificamos a melhor combinação para *Skinny*.

O caso $\gamma_k = 0$

Embora pareça pouco intuitivo, a convergência de *Skinny* permanece inalterada se tomarmos $\gamma_k = 0$ para todo $k \geq 1$. Isso se deve essencialmente ao "operador" φ_k que garante, teoricamente, a viabilidade do ponto $\varphi_k(x^k + d)$, caso $x^k + d$ não o seja. Na prática, o caso merece um pouco mais de discussão.

Em primeiro lugar, a escolha $\gamma_k = 0$ para todo k implica em lidarmos diretamente com o conjunto original. Como o foco deste trabalho são conjuntos viáveis magros, estamos impossibilitados de utilizar na fase de aprimoramento grande parte dos algoritmos existentes atualmente.

Em segundo lugar, supondo que a restauração seja bem sucedida sempre que requerida, temos que todos os iterados pertencem a Ω e a $\mathcal{X}$. Pelo critério de parada natural de *Skinny* ($\gamma_k = 0$), fatalmente o algoritmo será encerrado após a primeira iteração. Nesse caso, ao menos uma das duas situações devem ocorrer para que encontremos bons resultados: há bons algoritmos que lidam (mesmo que heuristicamente) com problemas sem derivadas com restrições magras na fase de aprimoramento; ou um número suficientemente grande de direções pseudo-aleatórias na fase de consulta é utilizado. Caso nada seja feito na fase de aprimoramento e um número pequeno de direções seja consultado, o ponto viável corrente é declarado solução de (3.1) sem que tenhamos informações suficientes sobre a sua vizinhança.

Por fim, em terceiro lugar, é possível que encontrar pontos viáveis no conjunto viável original (possivelmente magro) seja uma tarefa mais difícil do que encontrar pontos viáveis no conjunto engordado. Logo, a restauração, que é um processo essencial tanto para a própria fase de restauração quanto para o cômputo de φ_k, pode ficar comprometida.

Apesar de todos os fatores contra a utilização de $\gamma_k = 0$, achamos importante deixar tal possibilidade em aberto. De fato, se as restrições do problema (3.1) possuem derivadas (ou seja, a restauração pode ser feita com algoritmos clássicos de otimização) então existem bons algoritmos para a fase de aprimoramento, como DFO. Seu bom resultado permite que um número reduzido de direções precise ser consultado. Na Seção 3.3 mostramos resultados numéricos associados à escolha $\gamma_k = 0$.

3.1.6 Critérios de parada

Supondo que o procedimento utilizado na fase de minimização cumpriu corretamente o seu papel, caso um ponto viável seja encontrado é razoável que o mesmo seja declarado como solução do problema. Assim, *Skinny* declara **sucesso** se após a fase de minimização (passo 2 do Algoritmo 3.1) um ponto $x^k \in \mathcal{X}$ tal que $g(x^k) \leq \varepsilon_{\text{viab}}$ foi encontrado.

O algoritmo declara **fracasso** no caso em que $\gamma_k \leq \varepsilon_{\text{viab}}$ e um ponto γ_k-viável não foi encontrado ou quando o número máximo de 1000 iterações ou de 10^6 avaliações da função objetivo foi atingido.

3.1.7 Outros parâmetros

Além dos parâmetros que já discutimos, há outros cujos valores foram escolhidos baseados na intuição e na observação. São descritos a seguir:

1. A semente inicial usada pelo método de Schrage para números pseudo-aleatórios é dada pelo número primo 12345701. Toda vez que *Skinny* é utilizado, a semente é a mesma, o que implica na mesma sequência de números pseudo-aleatórios gerada.

2. O raio no qual é feito o passo de consulta é dado por $\Delta = 1$.

3. O número de direções de consulta que compõem cada conjunto D_k é dado por $m_k = 2n$, onde n é a dimensão do problema a ser resolvido.

4. Toda vez que um algoritmo é utilizado na fase de aprimoramento, limitamos o número de avaliações da função objetivo em $1000n$ por utilização.

Como apresentado no Capítulo 2, o parâmetro Δ regula a otimalidade local dos pontos encontrados por *Skinny*. Um valor de Δ grande o suficiente para englobar todo o conjunto viável do problema original é, teoricamente, garantia de que minimizadores globais sejam encontrados. Na prática, porém, o processo ocorre de forma diferente e a grande razão para isso é o número finito de iterações, direções, avaliações, etc..., que são de fato consideradas.

Dessa forma, para nos aproximarmos ao máximo da teoria, se estamos interessados em encontrar minimizadores globais do problema, além de valores suficientemente grandes de Δ devemos também aumentar o número de direções consultadas em cada iteração (m_k). Também podemos permitir que o algoritmo execute um número indefinido de iterações, de forma que a densidade imposta pela Hipótese S2 seja respeitada.

3.2 Ajuste dos parâmetros

Antes de compararmos os algoritmos propostos com outros métodos existentes na literatura é necessário que encontremos os parâmetros ideais que resultem

na melhor qualidade das soluções obtidas. Os problemas utilizados para realizar tal otimização de performance foram 47 problemas do conjunto de testes de Hock e Schittkowski [HS81], discutidos no Apêndice A.2. Dos 105 problemas discutidos no apêndice, os 47 problemas escolhidos possuem conjuntos viáveis magros, pois todos possuem ao menos uma restrição de igualdade. Cada problema dessa coleção possui disponível o valor mínimo da função objetivo e alguns dos minimizadores globais conhecidos. Nos casos em que a solução não é conhecida, os valores disponibilizados são os melhores conhecidos.

Na Tabela 3.1 descrevemos as características dos 47 problemas selecionados. Nas primeiras quatro colunas, Prob. representa o número do problema, de acordo com o descrito em [HS81], Var. representa o número de variáveis, Des. representa o número de restrições de desigualdade e Ig. representa o número de restrições de igualdade. Os valores entre parênteses indicam quantas das restrições são lineares. Nos outros dois conjuntos de quatro colunas o significado é o mesmo.

Prob.	Var.	Des.	Ig.	Prob.	Var.	Des.	Ig.	Prob.	Var.	Des.	Ig.
6	2	0	1	49	5	0	2(2)	74	4	2(2)	3
7	2	0	1	50	5	0	3(3)	75	4	2(2)	3
8	2	0	2	51	5	0	3(3)	77	5	0	2
9	2	0	1(1)	52	5	0	3(3)	78	5	0	3
14	2	1	1(1)	53	5	0	3(3)	79	5	0	3
26	3	0	1	54	6	0	1(1)	80	5	0	3
27	3	0	1	55	6	0	6(6)	81	5	0	3
28	3	0	1(1)	56	7	0	4	87	6	0	4
32	3	1	1(1)	60	3	0	1	99	7	0	2
39	4	0	2	61	3	0	2	107	9	0	6
40	4	0	3	62	3	0	1(1)	109	9	4(2)	6
41	4	0	1(1)	63	3	0	2(1)	111	10	0	3(3)
42	4	0	2	68	4	0	2	112	10	0	3
46	5	0	2	69	4	0	2	114	10	8(4)	3(1)
47	5	0	3	71	4	1	1	119	16	0	8(8)
48	5	0	2(2)	73	4	2(1)	1(1)				

Tabela 3.1: Descrição dos 47 problemas de Hock e Schittkowski com domínios magros. Os valores em parênteses representam quantas das restrições são lineares.

Para analisar a qualidade das soluções encontradas pelas diversas versões de *Skinny*, temos que definir primeiramente o que significa resolver um problema. Supondo que $\bar{x}$ é o ponto encontrado pelo algoritmo e que $\bar{f} = f(\bar{x})$, dizemos que o algoritmo *resolveu* o problema se as seguintes condições foram satisfeitas:

$$g(\bar{x}) \leq \varepsilon_{\text{viab}}, \quad \bar{x} \in \mathcal{X} \quad \text{e} \quad \frac{\bar{f} - f_{\text{L}}}{\max\left\{1, |\bar{f}|, |f_{\text{L}}|\right\}} \leq \varepsilon_{\text{comp}}, \tag{3.15}$$

onde f_{L} é a melhor solução encontrada para o problema pelos algoritmos que estão sendo comparados no momento. Para o critério de viabilidade, utilizamos a mesma

tolerância do critério de parada $\varepsilon_{\text{viab}}$ e para o critério de comparação do valor funcional utilizamos $\varepsilon_{\text{comp}} = 10^{-1}$ ou $\varepsilon_{\text{comp}} = 10^{-8}$, dependendo da qualidade da solução que desejamos analisar. A fórmula

$$\frac{\bar{f} - f_{\text{L}}}{\max\left\{1, |\bar{f}|, |f_{\text{L}}|\right\}},$$

que relaciona o valor funcional obtido com o melhor encontrado, pode ser encontrada, por exemplo, nos testes numéricos de [GK10].

Há dois parâmetros principais que definem o comportamento de *Skinny*: θ, representando a taxa de emagrecimento do conjunto, e τ, representando o esforço para a fase de restauração do Algoritmo 3.1. Para escolher a melhor combinação dessas duas características, utilizamos o conjunto de Hock e Schittkowski, discutido anteriormente.

Como valores possíveis para a taxa de emagrecimento do conjunto, as possibilidades foram:

$$\theta \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$$

e para o esforço na fase de restauração variamos

$$\tau \in \{0, 0.1, 1\}.$$

O algoritmo *Skinny* foi implementado em Fortran 90, de acordo com o que foi discutido na Seção 3.1. Utilizamos nos testes um computador com processador Intel Core i7 2.67GHz, 8GB de memória RAM e sistema operacional Linux (Ubuntu). O compilador `gfortran-4.2` foi utilizado com a opção de otimização `-O4`. Algencan e DFO, utilizados nos processos internos de *Skinny*, também foram compilados com `gfortran-4.2` e com as opções de compilação padrão fornecidas por seus desenvolvedores. As bibliotecas BLAS e LAPACK (versão 3.2.1) foram necessárias para DFO. O critério de viabilidade é $\varepsilon_{\text{viab}} = 10^{-8}$ e o critério de otimalidade, utilizado pelos algoritmos na fase de aprimoramento, é $\varepsilon_{\text{otim}} = 10^{-8}$.

Para realizar as comparações utilizamos os gráficos de perfil de desempenho [DM02] e *data profile* [MW09]. Como medida de desempenho escolhemos o número de avaliações da função, por se tratar da operação mais custosa dos problemas que estamos interessados. Também tomamos a liberdade de dizer que um algoritmo foi mais rápido do que outro se foi o que gastou menos avaliações da função objetivo.

Nos gráficos de perfil de desempenho (*performance profiles*) duas informações são mostradas simultaneamente: a eficiência e a robustez dos algoritmos. Denotando por $\sigma_{\mathcal{A},i}$ o número de avaliações da função objetivo gastos pelo algoritmo $\mathcal{A}$ para resolver o problema i (de acordo com o critério (3.15) que definimos anteriormente) e por σ_i^* o menor número de avaliações que um algoritmo (dentre os comparados) gastou para resolver i, calculamos o seguinte número:

$$\text{n}_{\mathcal{A}}(\rho) = \left|\left\{i \mid \frac{\sigma_{\mathcal{A},i}}{\sigma_i^*} \leq \rho\right\}\right|,$$

onde $|\cdot|$ representa o número de elementos do conjunto.

		$\varepsilon_{\text{comp}} = 10^{-1}$		$\varepsilon_{\text{comp}} = 10^{-8}$	
θ	τ	Eficiência	Robustez	Eficiência	Robustez
0.1	0.0	**20/47**	45/47	10/47	33/47
0.1	0.1	17/47	**47/47**	**21/47**	**45/47**
0.1	1.0	16/47	**46/47**	**24/47**	44/47
0.3	0.0	17/47	45/47	7/47	35/47
0.3	0.1	7/47	**47/47**	7/47	**46/47**
0.3	1.0	7/47	**46/47**	7/47	44/47
0.5	0.0	**19/47**	44/47	8/47	33/47
0.5	0.1	6/47	**47/47**	7/47	**46/47**
0.5	1.0	6/47	45/47	6/47	**45/47**
0.7	0.0	17/47	43/47	7/47	31/47
0.7	0.1	6/47	45/47	6/47	44/47
0.7	1.0	6/47	44/47	6/47	44/47
0.9	0.0	**19/47**	44/47	7/47	31/47
0.9	0.1	6/47	44/47	6/47	43/47
0.9	1.0	6/47	43/47	6/47	41/47

Tabela 3.2: Análise de performance para 15 variações de *Skinny*. Os valores em negrito representam os 2 melhores resultados em cada coluna.

O número $\mathrm{n}_{\mathcal{A}}(\rho)$ indica quantos problemas o algoritmo $\mathcal{A}$ resolveu gastando até ρ vezes o número de avaliações de função do algoritmo que resolveu mais rapidamente o mesmo problema. Se tomamos $\rho = 1$ temos o número de problemas nos quais $\mathcal{A}$ foi o mais eficiente, ou seja, foi aquele que gastou menos avaliações funcionais. Por outro lado, se tomamos $\rho = \infty$, temos o número total de problemas que $\mathcal{A}$ resolveu, também conhecido como robustez. Observe que $\mathrm{n}_{\mathcal{A}}(\cdot)$ é crescente e se temos, por exemplo, $\mathrm{n}_{\mathcal{A}}(1) = 10$ e $\mathrm{n}_{\mathcal{A}}(2) = 15$, significa que há 5 problemas para os quais $\mathcal{A}$ foi até 2 vezes mais lento do que o algoritmo que os resolveu mais rapidamente.

Impossibilitados de apresentar um gráfico de perfil de desempenho para essa primeira comparação, dado que há 15 variações de *Skinny* no total, mostramos na Tabela 3.2 apenas os valores de eficiência e robustez. De acordo com o critério (3.15), analisamos o caso em que procuramos uma solução aproximada ($\varepsilon_{\text{comp}} = 10^{-1}$) e o caso em que desejamos a melhor solução ($\varepsilon_{\text{comp}} = 10^{-8}$). As entradas em negrito representam os dois melhores valores para cada critério.

Analisando a Tabela 3.2 percebemos que a eficiência diminui na medida que aumentamos o valor de θ. Surpreendentemente, a robustez não aumenta. Podemos afirmar que os melhores resultados estão associados a $\theta = 0.1$. Para escolher o melhor valor de τ vamos utilizar os gráficos de perfil de desempenho e também os *data profiles*.

O *data profile* foi apresentado em [MW09] para comparar algoritmos sem derivadas irrestritos. Os autores partem do princípio que as avaliações da função são as responsáveis pelo custo computacional do problema. Para um algoritmo $\mathcal{A}$ e um

número máximo ρ de avaliações da função objetivo, calculamos o número de problemas que foram resolvidos (de acordo com o critério (3.15)) com até ρ avaliações. De forma similar aos perfis de desempenho, quando $\rho = \infty$ temos o número total de problemas resolvidos por cada algoritmo. Em [MW09] os autores utilizam critérios diferentes de (3.15), dado que resolvem problemas irrestritos. Além disso, o número de avaliações da função gasto por cada algoritmo é divido por $n+1$, onde n é a dimensão do problema, numa tentativa de equilibrar os resultados. Optamos neste trabalho por apresentar diretamente os valores funcionais.

É interessante observar que os algoritmos não são comparados entre si no *data profile*. Com este tipo de gráfico podemos ter ideia do esforço computacional (em termos de número de avaliações da função objetivo) necessário para resolver o conjunto de problemas.

Comparando $\theta = 0.1$ com diferentes valores de τ e utilizando $\varepsilon = 10^{-1}$ mostramos na Figura 3.3 os gráficos de perfil de desempenho e *data profile*. Podemos ver no perfil de desempenho que $\tau = 0$ é o mais eficiente em 20 problemas, mas além disso, o gráfico nos mostra que essa variação resolveu 45 problemas (todos os que conseguiu) utilizando até 5 vezes o número de avaliações que o melhor algoritmo gastou. As outras variações precisaram de até 40 vezes para chegar a esse número. Por outro lado, podemos ver no *data profile* que se permitirmos até 2000 avaliações da função, as três variações resolvem aproximadamente o mesmo número de problemas, ou seja, todas utilizaram um baixo número de avaliações da função.

Porém, o que a Figura 3.4 nos mostra é que a escolha $\tau = 0$ não chega aos menores valores funcionais em cada problema. Percebemos tal fenômeno quando utilizamos $\varepsilon_{\text{comp}} = 10^{-8}$ na condição (3.15) e geramos o perfil de desempenho. Esse comportamento já havia sido mostrado na Tabela 3.2. Por outro lado, as variações $\tau = 0.1$ e $\tau = 1.0$ mantiveram basicamente os mesmos resultados. Com base nessas observações e na Tabela 3.2, escolhemos a configuração

$$\theta = 0.1 \quad \text{e} \quad \tau = 0.1$$

como a configuração padrão de *Skinny*, com a qual o comparamos com os outros algoritmos existentes na literatura.

A crescente eficiência de *Skinny* em virtude do decréscimo de θ originou a questão de que valores menores do que $\theta = 0.1$ poderiam gerar melhores resultados. De fato, fixando $\tau = 0.1$, a Figura 3.5 mostra claramente uma mudança no comportamento do algoritmo: com a redução de θ a eficiência aumenta e a robustez diminui. Para a comparação realizada, utilizamos apenas $\varepsilon_{\text{comp}} = 10^{-8}$, pois estamos mais interessados em mostrar o comportamento do que realmente escolher uma melhor configuração.

De acordo com a Figura 3.5, os ganhos de eficiência pela escolha de $\theta = 10^{-8}$ superam as perdas de robustez. Porém, essa configuração não será escolhida como a padrão de *Skinny*, pois, como mostraremos na Seção 3.3, quanto menor o valor de θ mais o algoritmo fica parecido com aquele usado na fase de aprimoramento, no nosso caso DFO. Assim, nossa escolha continua sendo $\theta = 0.1$ e $\tau = 0.1$, que aparentam ser os valores que separam diferentes comportamentos de *Skinny*.

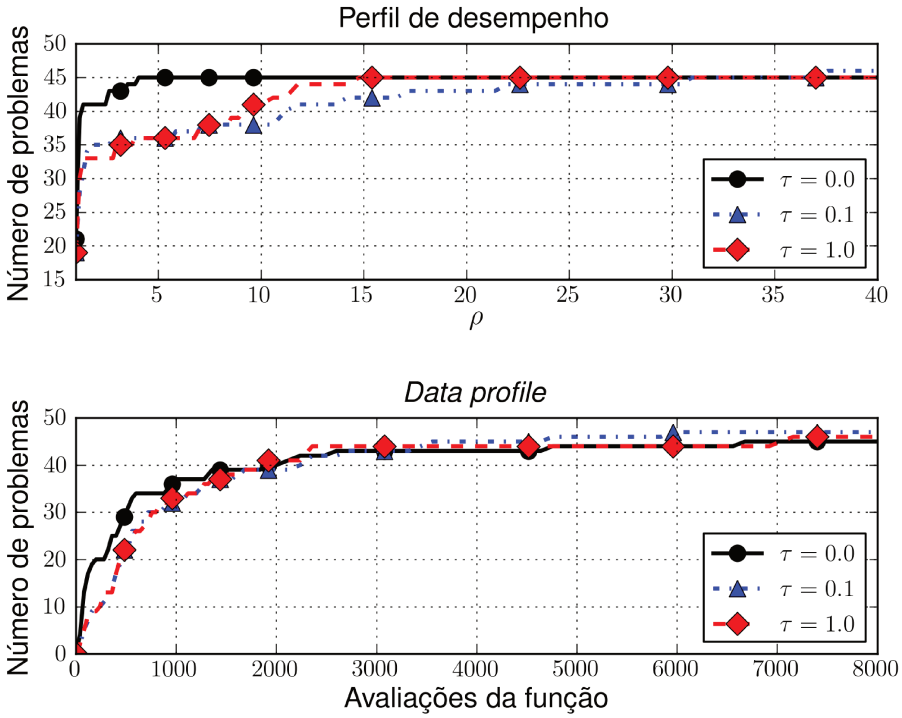


Figura 3.3: Comparação para $\theta = 0.1$ fixo e as variações $\tau \in \{0, 0.1, 1\}$, utilizando $\varepsilon_{\text{comp}} = 10^{-1}$.

3.3 Experimentos numéricos

Após termos incorporado os resultados encontrados na Seção 3.2 no algoritmo *Skinny*, realizamos comparações com 3 algoritmos existentes na literatura:

1. NOMAD [LD11]: implementação em C$_{++}$ da classe de algoritmos MADS [ADJ06, AADJLD09], com diversas outras estratégias de otimização, como a barreira progressiva [ADJ09]. MADS foi discutido na Seção 1.2;

2. HOPSPACK [Pla09]: implementação em C$_{++}$ do algoritmo baseado em Lagrangianos Aumentados descrito em [KLT06b, GK10] e discutido na Seção 1.5;

3. DFO: um algoritmo em Fortran 77 baseado em regiões de confiança, apresentado em [CST97b, CST97a, CST98], e que foi discutido na Seção 1.6.

Dentre os três algoritmos, apenas HOPSPACK possui uma teoria de convergência que inclui domínios magros. Os outros dois algoritmos podem ser considerados heurísticas aplicadas aos problemas-teste.

Os problemas escolhidos para os testes foram os 47 problemas de Hock e Schittkowski utilizados na Seção 3.2. Os problemas têm conjunto viável magro, poucas variáveis e as restrições possuem derivadas. Dessa forma, estamos nas condições ideais para aplicarmos *Skinny*.

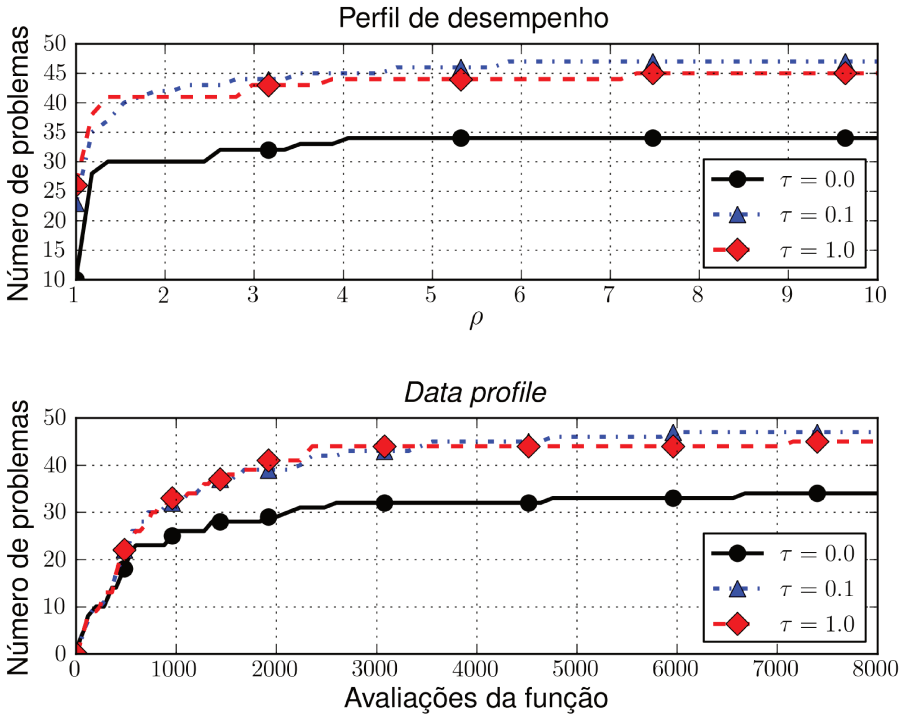


Figura 3.4: Comparação para $\theta = 0.1$ fixo e as variações $\tau \in \{0, 0.1, 1\}$, utilizando $\varepsilon_{\text{comp}} = 10^{-8}$.

Escolhemos comparar nosso método com DFO, pois podemos utilizar o algoritmo Algencan para resolver os subproblemas de região de confiança com restrições, dado que as restrições têm derivadas disponíveis. Essa combinação resultou em excelentes resultados.

A escolha de NOMAD é devida à sua semelhança com *Skinny* com respeito à busca direta direcional baseada em hipóteses de densidade. Se fosse utilizar a penalidade extrema de MADS, a menos de raras exceções, NOMAD estaria fadado ao fracasso em problemas com restrições magras. O que nos motiva a utilizá-lo nos testes a seguir é a barreira progressiva [ADJ09], que permite com que o algoritmo percorra pontos inviáveis do conjunto Ω, penalizando um pouco a inviabilidade. Apesar disso, a teoria de MADS, que sustenta NOMAD, não abrange problemas com restrições de igualdade, dado que o cone hipertangente é vazio para todo ponto viável.

Assim como *Skinny* possui uma fase de aprimoramento, na qual qualquer procedimento pode ser utilizado para reduzir o valor da função objetivo, NOMAD possui o passo de busca. No passo de busca, utiliza-se qualquer procedimento para procurar em um número finito de pontos da malha um ponto que reduza o valor funcional. Infelizmente, não podemos usar DFO no passo de busca, como é feito no passo de aprimoramento, dado que não há garantia de que a solução encontrada pelo algoritmo estará localizada na malha. Nesse aspecto, *Skinny* foi beneficiado por desassociar-se da

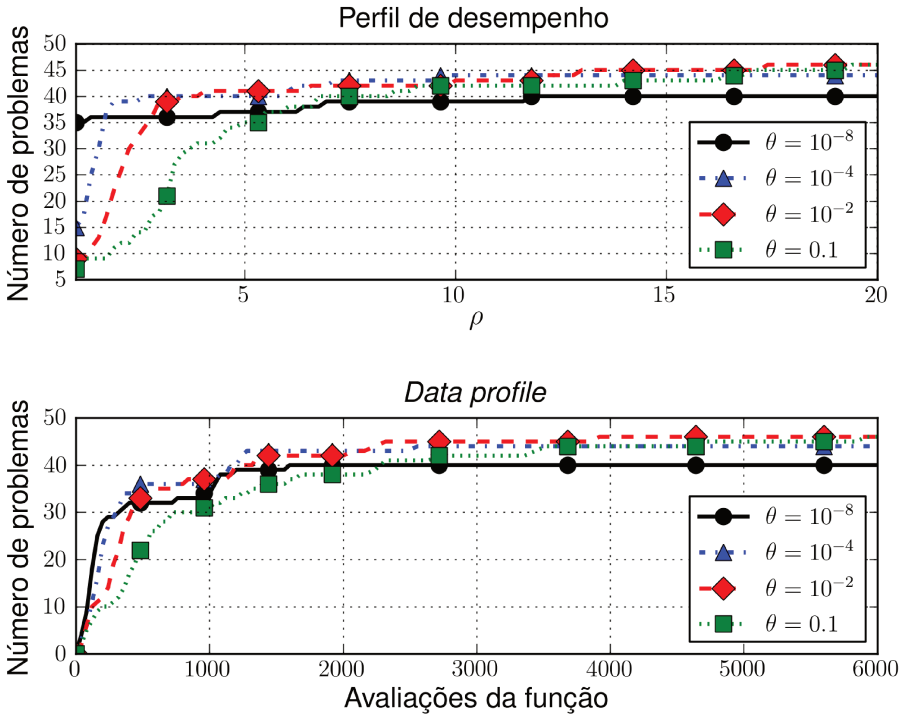


Figura 3.5: Comparação de valores pequenos de θ para $\tau = 0.1$ fixo, com $\varepsilon_{\text{comp}} = 10^{-8}$.

teoria que usa o decréscimo simples e, consequentemente, malhas.

Com exceção de NOMAD, todos os algoritmos testados estão com os valores originais de seus parâmetros. Para uma compatibilidade com os demais, modificamos o critério de parada por otimalidade de NOMAD para $\Delta_k^p \leq 10^{-3}$, onde Δ_k^p, como explicado na Seção 1.2, representa o grau de refinamento da malha. Todos os algoritmos tiveram seu critério de viabilidade alterado para 10^{-8}, exceto NOMAD, cujo funcionamento piorou consideravelmente quando a opção padrão era alterada. Para que *Skinny* fosse também padronizado com os demais algoritmos, utilizamos $\varepsilon_{\text{viab}} = 10^{-8}$ e, na fase de aprimoramento, $\varepsilon_{\text{otim}} = 10^{-3}$.

Para as comparações, utilizamos o mesmo critério adotado na Seção 3.2, dado por (3.15), com $\varepsilon_{\text{viab}} = 10^{-8}$ e $\varepsilon_{\text{comp}} = 10^{-1}$. O tempo máximo de 1 hora de CPU foi permitido para cada problema. Para analisar os resultados foram usados os gráficos de perfil de desempenho e *data profiles*.

Na Tabela 3.3 apresentamos os resultados obtidos. A primeira coluna (Prob.) contém o número do problema de acordo com [HS81], cujas características podem ser vistas na Tabela A.2 do Apêndice A.2. Nas demais colunas, f representa o valor funcional na solução e #AF o número de avaliações da função utilizadas por cada algoritmo. As entradas da tabela que possuem '−' representam os casos em que o tempo máximo de CPU foi utilizado e não obtivemos informação sobre o último ponto calculado. O valores marcados com * representam soluções inviáveis e aqueles marcados com $^\bullet$ representam

soluções viáveis, porém cujo valor funcional não está próximo do menor encontrado, de acordo com o critério (3.15).

Podemos observar que *Skinny* encontrou soluções viáveis em todos os problemas e em apenas 3 problemas não encontrou o melhor valor de função objetivo.

DFO não foi capaz de encontrar soluções viáveis em 6 problemas. Isso ocorreu porque o algoritmo necessita de ao menos 2 pontos viáveis iniciais para construir o primeiro modelo linear da função objetivo. Nos problemas 61, 71, 74, 75, 87 e 109 os pontos iniciais são inviáveis e, por dificuldades numéricas, não foi possível resolver os problemas de "restauração" que DFO necessita para encontrar pontos viáveis. O problema foi declarado inviável e apenas uma avaliação da função foi feita.

Os algoritmos HOPSPACK e NOMAD tiveram um comportamento muito parecido, embora os resultados de HOPSPACK sejam um pouco melhores. HOPSPACK falhou em encontrar soluções viáveis de 22 problemas e excedeu o limite de tempo computacional em 3. No total, foi capaz de encontrar os menores valores em 16 problemas. Surpreendentemente, NOMAD não conseguiu encontrar pontos viáveis em apenas 21 problemas e atingiu o menor valor em 15 problemas. Há muitos casos nos quais NOMAD e HOPSPACK não conseguiram encontrar pontos viáveis e acreditamos que um dos motivos pode ser o critério de parada de 10^{-3}, que é pouco rigoroso.

Na Figura 3.6 comparamos o desempenho de cada algoritmo, com respeito ao número de avaliações da função. Podemos ver na Figura 3.6(b) que *Skinny* é o mais robusto, perdendo em eficiência apenas para DFO. Esse comportamento é esperado, dado que *Skinny* precisa avaliar a função um maior número de vezes para garantir a convergência global do método. A perda de eficiência, porém não é muito elevada, pois na Figura 3.6(a) percebemos que *Skinny* e DFO resolvem aproximadamente o mesmo número de problemas se permitimos até 1000 avaliações da função objetivo. Ainda no limite de 1000 avaliações, verificamos que o número de problemas resolvidos é mais do que o dobro do conseguido por HOPSPACK e DFO.

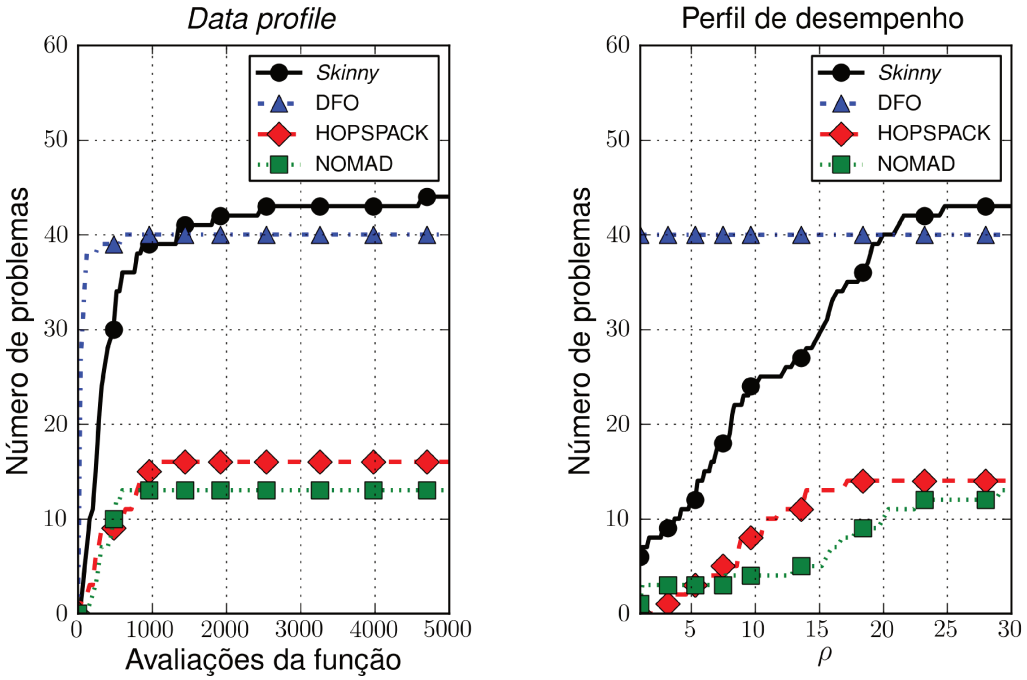
Apenas para ilustrarmos que a modificação do critério de parada por otimalidade de 10^{-3} para 10^{-8} aumenta o número de pontos viáveis encontrados por HOPSPACK e NOMAD, apresentamos a Figura 3.7. Para *Skinny* isso significa utilizar $\varepsilon_{\text{otim}} = 10^{-8}$. Dado que ambos caminham pela inviabilidade e, no caso de HOPSPACK, só atingem a viabilidade no limite, fazê-los parar antecipadamente poderia resultar em pontos inviáveis. Podemos ver que o número de problemas resolvidos por cada um aumentou, enquanto o comportamento de *Skinny* e DFO permaneceu inalterado, apenas aumentando o número de avaliações da função.

3.3.1 O caso $\gamma_k = 0$

Somando o fato de DFO ter sido o algoritmo mais eficiente aos resultados com respeito ao parâmetro θ mostrados na Seção 3.2, podemos nos questionar sobre os benefícios do engordamento. O excelente desempenho de DFO está fortemente ligado à habilidade de Algencan em resolver problemas de restauração. Da mesma forma, o bom desempenho de *Skinny* está ligado ao bom desempenho de DFO e, quanto menor o valor

Prob.	*Skinny* f	#AF	HOPSPACK f	#AF	NOMAD f	#AF	DFO f	#AF
6	8.8304E−08	97	* 4.8400E+00	151	* 3.3265E+00	83	2.2009E−23	14
7	−1.7321E+00	150	• 6.9315E−01	325	*−1.7172E+00	73	−1.7321E+00	10
8	−1.0000E+00	56	*−1.0000E+00	187	*−1.0000E+00	117	−1.0000E+00	3
9	−5.0000E−01	109	−5.0000E−01	26	−5.0000E−01	186	−5.0000E−01	12
14	1.3935E+00	142	• 1.3941E+00	202	1.3944E+00	151	1.3935E+00	9
26	• 2.1160E+01	33	• 2.1160E+01	585	• 3.6000E−01	300	1.2942E−06	40
27	4.0000E+00	269	* 4.0006E+00	1358	4.0000E+00	252	4.0000E+00	33
28	3.6892E−27	43	7.7034E−08	264	1.1055E−06	508	2.3039E−18	25
32	1.0000E+00	209	1.0000E+00	51	• 1.9904E+00	599	1.0000E+00	11
39	−1.0000E+00	302	−1.0000E+00	830	*−1.0109E+00	997	−1.0000E+00	23
40	−2.5000E−01	215	*−2.5056E−01	897	*−2.4985E−01	147	−2.5000E−01	14
41	1.9259E+00	348	1.9259E+00	292	1.9259E+00	600	1.9259E+00	34
42	1.3858E+01	254	1.4000E+01	779	1.4000E+01	276	1.3858E+01	14
46	2.1090E−02	501	• 3.3376E+00	777	• 3.3376E+00	246	2.9829E−07	77
47	1.4548E−08	302	• 1.2495E+01	901	• 1.2495E+01	343	1.8714E−06	50
48	1.4969E−16	76	1.1174E−06	497	• 2.4574E+00	317	1.2711E−18	29
49	3.7388E−05	261	1.4294E−04	1002	• 2.6600E+02	408	2.7801E−05	72
50	7.7030E−06	246	5.2937E−07	290	• 1.7416E+04	369	2.1369E−07	46
51	8.4671E−17	88	1.2537E−06	142	• 6.4060E−01	384	1.1025E−18	16
52	5.3267E+00	337	* 5.3267E+00	311	5.3366E+00	449	5.3266E+00	20
53	4.0930E+00	295	4.0930E+00	216	4.3744E+00	284	4.0930E+00	18
54	•−1.5581E−01	335	–	–	−9.0805E−01	8074	•−1.5399E−01	20
55	6.3333E+00	223	* 6.0000E+00	1	6.7068E+00	301	6.6667E+00	9
56	•−1.0000E+00	93	•−1.0000E+00	2075	•−1.0000E+00	452	−3.4560E+00	45
60	3.2570E−02	236	* 5.4709E−02	465	* 4.5322E+00	139	3.2571E−02	29
61	−1.4365E+02	196	−1.4300E+02	621	−1.4300E+02	203	* 0.0000E+00	1
62	−2.5698E+04	33	−2.6272E+04	233	−2.6272E+04	418	−2.6273E+04	32
63	9.6172E+02	159	* 9.6261E+02	317	* 9.6599E+02	128	9.6172E+02	10
68	−9.2041E−01	439	*−8.4354E−01	1316	•−3.0774E−01	346	−9.2042E−01	106
69	−9.5671E+02	581	*−9.5665E+02	2471	•−8.4604E+02	486	−9.5107E+02	73
71	1.7014E+01	398	* 1.7031E+01	1939	* 1.8177E+01	107	* 1.6000E+01	1
73	2.9894E+01	305	3.0160E+01	223	3.0000E+01	464	2.9894E+01	16
74	5.1265E+03	279	* 5.1447E+03	46145	* 5.1706E+03	47347	* 0.0000E+00	1
75	5.1744E+03	2453	* 5.2331E+03	22678	* 5.1265E+03	9883	* 0.0000E+00	1
77	2.4151E−01	598	* 4.6807E+00	1904	* 2.3979E+02	358	2.4151E−01	85
78	−2.9197E+00	368	*−2.8917E+00	869	*−2.1482E+00	213	−2.9197E+00	27
79	7.8777E−02	495	* 2.4186E−01	1054	* 8.8562E+01	203	7.8777E−02	40
80	5.3950E−02	458	* 1.0000E+00	557	* 2.0909E−01	210	5.3950E−02	23
81	5.3950E−02	481	* 9.9999E−01	557	* 3.3160E−01	196	5.3950E−02	23
87	8.9276E+03	493	* 9.3254E+03	16244	9.1436E+03	498	* 4.2090E+04	1
99	−8.3108E+08	777	*−7.4573E+08	729	*−8.1588E+08	311	−8.3108E+08	82
107	5.0550E+03	858	* 5.0628E+03	7232	* 5.2336E+03	795	5.0550E+03	23
109	5.3621E+03	775	* 5.5010E+03	57551	* 5.2059E+03	244247	* 0.0000E+00	1
111	−4.7761E+01	4607	–	–	*−4.3270E+01	3255	−4.7760E+01	561
112	−4.7761E+01	1815	−4.7761E+01	730	*−4.1352E+01	524	−4.7760E+01	85
114	−1.7688E+03	1359	–	–	−1.0576E+03	2233	−1.7688E+03	276
119	2.4490E+02	1331	2.4493E+02	944	* 7.9130E+02	1236	2.4490E+02	90

Tabela 3.3: Comparação entre *Skinny*, HOPSPACK, NOMAD e DFO nos 47 problemas com conjunto viável magro. Os valores de função marcados representam problemas que não foram considerados resolvidos: * significa falta de viabilidade e • significa falta de otimalidade da solução.



(a) (b)

Figura 3.6: Comparação de desempenho entre *Skinny*, HOPSPACK, NOMAD e DFO.

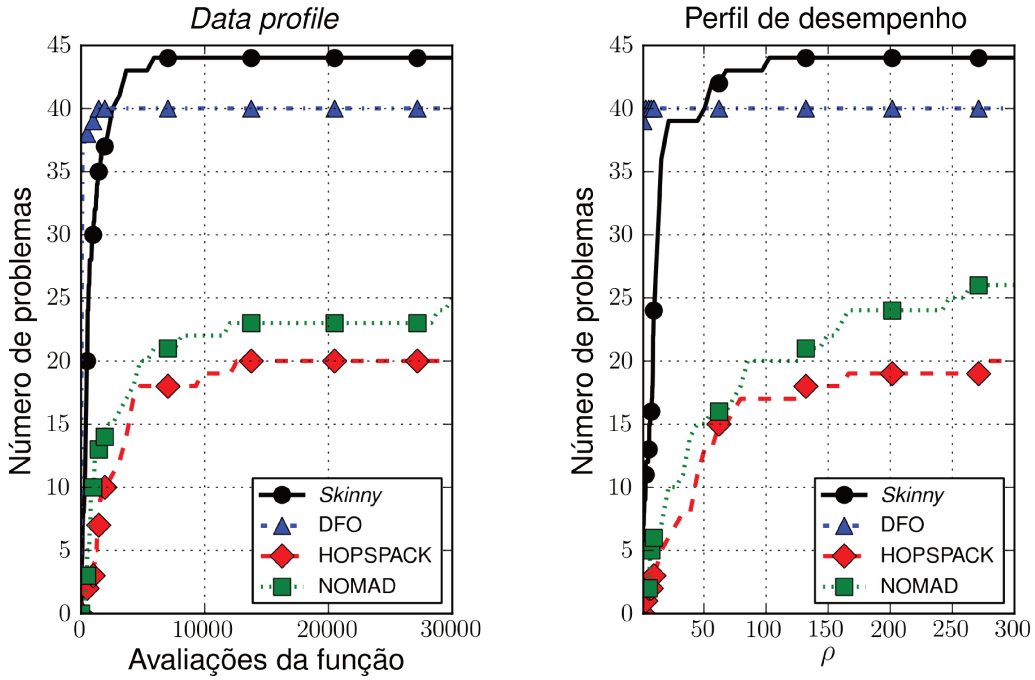
de θ, mais confiança depositamos sobre as fases de aprimoramento e de restauração.

Na Tabela 3.4 verificamos o comportamento de *Skinny* ao tomarmos $\gamma_k = 0$ para todo k. Aplicamos os testes de perfil de desempenho para DFO, *Skinny* original e *Skinny* com $\gamma_k = 0$. Para cada algoritmo temos que Viab. representa o número de problemas nos quais o algoritmo encontrou um ponto viável, Conv. representa o número de problemas resolvidos pelo algoritmo (no sentido do critério (3.15)) e Ef. representa a quantidade de problemas resolvidos nos quais o algoritmo foi o mais rápido. Nas última três colunas mostramos o número de problemas para os quais foram gastas menos que 10^2, entre 10^2 e 10^3 e mais do que 10^3 avaliações da função objetivo, respectivamente.

Na prática, dado que o critério de parada é $\gamma_k \leq \varepsilon_{\text{viab}}$, com $\varepsilon_{\text{viab}} = 10^{-8}$, *Skinny* irá realizar apenas uma iteração, ou seja, uma restauração e uma minimização. Na Tabela 3.4, podemos ver claramente que o comportamento de *Skinny* com $\gamma_k = 0$ é muito semelhante ao de DFO, com um ganho extra de robustez, que está ligado à fase de consulta. Os resultados justificam a razão do ganho de eficiência e da perda de robustez ao reduzirmos o valor de θ na Seção 3.2.

Apesar de todos os resultados incentivarem o uso de $\gamma_k = 0$, mencionamos na Subseção 3.1.5 que um engordamento adequado do conjunto pode conectar conjuntos desconexos. Apresentamos aqui um exemplo no qual verificamos tal propriedade.

Suponha que temos um conjunto viável e que temos pelo menos dois con-



(a) (b)

Figura 3.7: Comparação de desempenho entre *Skinny*, HOPSPACK, NOMAD e DFO quando o critério de parada é modificado para 10^{-8}.

	Viab.	Conv.	Ef.	Av. da função		
				$[0, 10^2]$	$(10^2, 10^3]$	$(10^3, 10^4]$
DFO	41	41	34	38/41	3/41	0/41
Skinny $\gamma_k = 0$	47	46	13	42/47	3/47	2/47
Skinny	47	45	1	8/47	34/47	5/47

Tabela 3.4: *Skinny* com $\gamma_k = 0$ foi melhor do que *Skinny* em conjuntos engordados e teve um comportamento parecido com o comportamento de DFO.

juntos abertos disjuntos nos quais ele está contido. Esse conjunto é conhecido como desconexo [Lim70]. Logo, começando em uma das partes desconexas desse conjunto estamos destinados a continuar nela até o fim. Porém, como mostrou a Figura 3.2, um bom valor de engordamento, pode conectar os conjuntos. Nesse momento, o passo de minimização pode se deslocar por todo o domínio, em busca do minimizador global. Esse comportamento é mostrado no Exemplo 3.2.

3.2 Exemplo (Conjuntos desconexos)
Suponha que deseja-se resolver o seguinte problema:

$$\begin{array}{ll} \min & x^2 \\ s.\ a & \prod_{i=1}^{5}(x-i)=0. \end{array}$$

A restrição do problema é a equivalente contínua da restrição $x \in \{1,2,3,4,5\}$, ou seja, há restrições de integralidade. Desta forma, o problema possui conjunto viável desconexo com valor mínimo no ponto $x^* = 2$. Seja $x^0 = 5$ o ponto inicial.

Com $\gamma_k = 0$, Skinny não precisa fazer nada na restauração, dado que x^0 é viável. Na fase de minimização, nada é feito por DFO, dado que o ponto é um minimizador local. Na fase de consulta, todas as direções são inviáveis se Δ é suficientemente pequeno, e a projeção resulta no próprio ponto x^0. Logo, $x^0 = 5$ é declarado solução.

Tomando γ_k suficientemente grande, todas as desconexidades desaparecem e o problema engordado resume-se a minimizar x^2 em um intervalo (contínuo) contendo os pontos $\{1,2,3,4,5\}$. Logo, DFO encontra o minimizador global irrestrito $x^1 = 0$ e, nos passos seguintes, encontra a solução global $x^* = 1$.

Observação: Embora não seja o objetivo desta discussão, se o algoritmo utilizado nos problemas de restauração de *Skinny* souber lidar eficientemente com as restrições do problema, valores adequados de Δ e de m_k na fase de consulta também são capazes de "pular" desconexidades. Isso pode ser uma alternativa, caso deseje-se utilizar $\gamma_k = 0$. Na Figura 3.8 mostramos que uma direção de consulta pode ser gerada próxima a uma parte inacessível do conjunto viável. Ao restaurarmos tal ponto inviável (para encontrar $\varphi_k(z^\ell + d)$) podemos encontrar um ponto viável nesta parte do conjunto, possivelmente com menor valor da função objetivo.

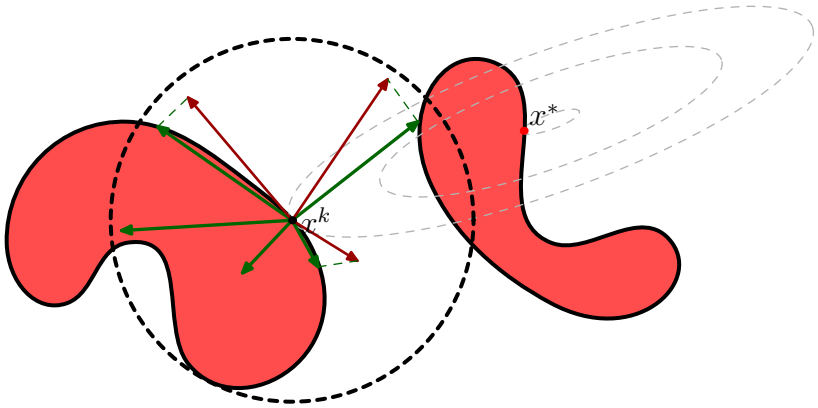


Figura 3.8: Pulando desconexidades na fase de consulta, com a ajuda de bons algoritmos de restauração e de valores adequados de Δ e m_k. As elipses pontilhadas representam as curvas de nível da função objetivo.

3.3.2 Problemas vorazes

Sob aspectos teóricos, além de *Skinny*, os algoritmos baseados em penalidades são os únicos conhecidos pelos autores capazes de resolver problemas sem derivadas com conjunto viável magro. Vamos mostrar que o presente algoritmo não sofre com um fenômeno que acomete os algoritmos baseados em função de penalidade: a voracidade.

A voracidade foi discutida em [CMMS10, BCMM11] e está basicamente associada à forte atração que os minimizadores irrestritos da função de mérito ou de penalidade exercem sobre as primeiras iterações de algoritmos baseados em função de mérito (como, por exemplo, Lagrangianos Aumentados). Utilizamos 7 problemas que apresentam esse fenômeno, discutidos em [CMMS10]. Os problemas podem ser encontrados no Apêndice A.3.

A Tabela 3.5 apresenta os resultados das comparações. Nas 4 primeiras colunas indicamos o problema considerado, o número de variáveis, o número de restrições de desigualdade e o número de restrições de igualdade, respectivamente. Para cada algoritmo, f representa o valor da função objetivo associado à solução e #AF representa o número de avaliações da função utilizado.

				Skinny		HOPSPACK	
Problema	Var.	Des.	Ig.	f	#AF	f	#AF
1 [CMMS10]	100	100	0	* 4.8990E+05	1	*−3.43000E+04	1001
3 [CMMS10]	2	0	1	−5.31732E+00	68110	0.00000E+00	63022
4 [CMMS10]	2	0	1	−2.28486E+01	188	*−1.21825E+01	105
5 [CMMS10]	50	1	0	−5.00000E+00	2104	*−2.22900E+23	50001
6 [CMMS10]	100	1	0	−5.00008E−03	1	*−1.00000E+30	25442
40 [HS81]	4	0	3	−0.25000E+00	215	*−2.50563E−01	897
56 [HS81]	7	0	4	−3.45600E+00	832	*−8.18800E+09	154022

Tabela 3.5: Comparação entre *Skinny* e HOPSPACK em 7 problemas que apresentam o fenômeno da voracidade. Os valores marcados com * representam soluções inviáveis.

Devido ao processo de separação entre viabilidade e otimalidade utilizado por *Skinny*, 6 problemas foram resolvidos. HOPSPACK, por outro lado, sofreu com a voracidade em pelo menos 4 problemas: 1, 5, 6 e 56.

Nos problemas 1 [CMMS10] e 6 [CMMS10], *Skinny* excedeu o limite de 1 hora de tempo de CPU, devido ao fato de DFO ter dificuldades em problemas com um número grande de variáveis. Se limitamos o número máximo de iterações de DFO na fase de aprimoramento de *Skinny* para n (ao invés de $1000n$), então encontramos $f(\bar{x}) = 0.37883$ gastando 37456 avaliações da função, no problema 1, e $f(\bar{x}) = -0.61562$ gastando 2417 avaliações da função, no problema 6.

3.3.3 Aplicação

Apesar de serem excelentes como testes, os problemas de Hock e Schittkowski podem ser facilmente resolvidos por algoritmos clássicos de programação não

linear. Desejamos mostrar aqui a capacidade de *Skinny* em resolver problemas que realmente possuem o espírito sem derivadas. Para isso, vamos aplicá-lo a uma variação do problema da área introduzido em [Ped09, DEMP11]. Essa variação foi denominada "Problema do Vestibular" e é completamente discutido no Apêndice A.1.

No problema do vestibular com r disciplinas, são dados n_{H} hiperplanos para que sejam separados n_{O} pontos em $\mathbb{R}^r$, representando, cada um, r notas obtidas no vestibular por cada estudante. Além disso, há outros n_{D} pontos que devem ficar suficientemente dentro do politopo resultante e n_{B} pontos que devem ficar na sua borda. Estes dois últimos tipos de pontos representam alunos (aprovados no vestibular anterior) pré-selecionados segundo seu desempenho na faculdade.

A Tabela 3.6 apresenta informações sobre a dimensão das instâncias do problema do vestibular utilizadas nos testes. As primeiras 5 colunas foram explicadas no parágrafo anterior. Nas três últimas colunas, Var. representa o número de variáveis, Des. representa o número de restrições de desigualdade e Ig. representa o número de restrições de igualdade. Os valores entre parênteses indicam quantas das restrições são lineares. Maiores detalhes de cada instância são fornecidos no Apêndice A.1.

Instância	r	n_{H}	n_{B}	n_{D}	n_{O}	Var.	Des.	Ig.
SVM1-2	2	2	3	1	3	6	12 (8)	3
SVM1-3	2	3	3	1	3	9	18 (12)	3
SVM2-3	2	3	3	1	10^5	9	18 (12)	3
SVM2-4	2	4	3	1	10^5	12	24 (16)	3
SVM2-4'	2	4	3	1	10^5	12	24 (16)	3
SVM3-3	2	3	3	9	15000	9	66 (36)	3
SVM3-4	2	4	3	9	15000	12	88 (48)	3
SVM3-5	2	5	3	9	15000	15	110(60)	3
SVM4-4	2	4	4	4	1000	12	52 (32)	4

Tabela 3.6: Instâncias do problema do vestibular. Os valores entre parênteses representam o número de restrições que são lineares.

A função objetivo é dada pela contagem de pontos do tipo O que estão no interior do politopo formado pelos hiperplanos. Podemos ver que ela não possui derivadas e seu custo está associado a n_{O}, que pode ser grande, dependendo da instância escolhida. Todas as restrições são suaves e possuem as derivadas disponíveis. Apenas na instância SVM3 os dados são realmente baseados no vestibular para as disciplinas de matemática e português ($r = 2$), encontrados em [CON02]. Nas instâncias SVM2-4 e SVM2-4' apenas mudamos o parâmetro ε_{D} de 0.5 para 2, que indica o quão *suficientemente dentro* do politopo devem estar os pontos do tipo D (veja o Apêndice A.1).

Resolvemos as instâncias com *Skinny*, HOPSPACK e DFO. Foram feitas 10 tentativas para cada instância, sendo que todos os pontos iniciais foram os mesmos para os três algoritmos. Na Tabela 3.7 apresentamos os resultados encontrados. Para cada algoritmo, temos que f representa o valor da função objetivo na solução, Viab. representa a norma infinito das restrições e #AF representa o número de avaliações da

função objetivo utilizadas para encontrar a solução. Os valores em negrito indicam as soluções com menor valor funcional.

	Skinny			HOPSPACK			DFO		
Instância	f	Viab.	#AF	f	Viab.	#AF	f	Viab.	#AF
SVM1-2	**1**	7.0E−11	334	**1**	4.4E−13	2175	**1**	3.0E−13	27
SVM1-3	**0**	3.0E−10	593	2	0.0	1811	**0**	9.0E−11	27
SVM2-3	**29295**	5.0E−09	908	**29295**	1.1E−22	4797	**29295**	6.0E−09	88
SVM2-4	**28889**	6.0E−11	3244	**28889**	1.8E−10	3651	**28889**	8.0E−11	205
SVM2-4'	**47518**	6.0E−11	1340	50119	9.8E−09	4564	47535	3.0E−09	127
SVM3-3	**1708**	6.0E−10	1595	1709	9.6E−15	3504	1838	3.0E−10	60
SVM3-4	**1708**	5.0E−09	2325	1709	9.6E−15	3504	1838	3.0E−10	55
SVM3-5	1838	2.0E−09	432	**1691**	1.1E−14	5591	1819	2.0E+00	1
SVM4-4	**116**	1.0E−09	981	**116**	7.1E−10	3803	129	4.0E−09	2

Tabela 3.7: Resultados encontrados para o problema do vestibular.

Podemos verificar que novamente *Skinny* herdou a eficiência de DFO e a robustez da estratégia usada na fase de consulta. Exceto na instância SVM3-5, *Skinny* sempre encontrou as melhores soluções conhecidas, gastando menos avaliações da função do que HOPSPACK. Nos casos em que n_O possui valor elevado, isso significou uma grande redução no tempo computacional. É possível ver também que HOPSPACK atingiu bons valores de viabilidade, explicados pela sua habilidade em tratar diretamente as restrições lineares. Apenas em uma instância (SVM3-5) DFO não foi capaz de encontrar uma solução viável.

Nas figuras 3.9, 3.10, 3.11, 3.12 e 3.13 mostramos as soluções encontradas por *Skinny* e, no caso da instância SVM3-5, também mostramos a solução encontrada por HOPSPACK. Todas as configurações parecem relacionadas com a solução ótima, embora só tenhamos certeza disso nas instâncias SVM1-2 e SVM1-3. Em todos os desenhos, as setas apontam para o interior do poliedro. Os quadrados (azuis) estão associados aos pontos B, que devem estar na borda do poliedro. Os pontos maiores (verdes) são os pontos do tipo D, que devem estar suficientemente dentro do poliedro. Por fim, os pontos menores estão associados aos pontos do tipo O, os quais queremos que o menor número possível esteja no interior do poliedro formado pelos hiperplanos.

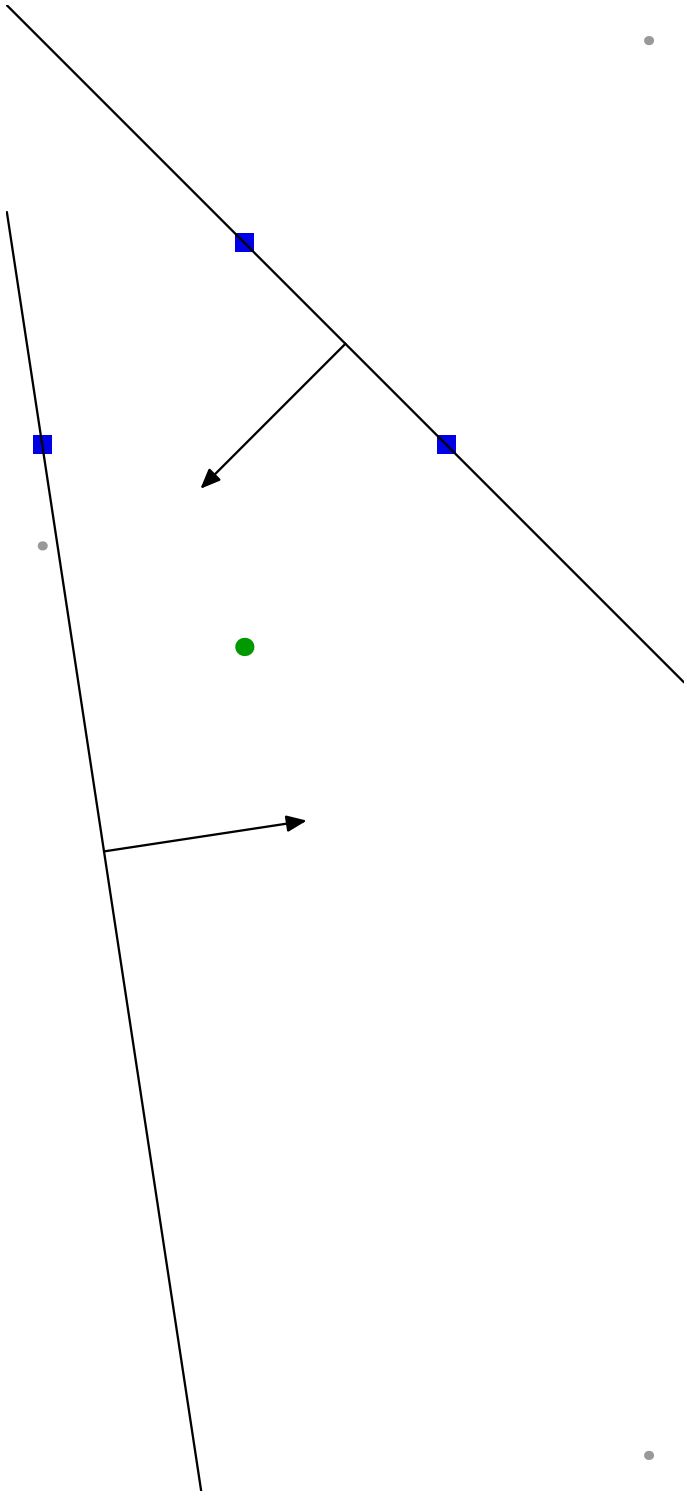


Figura 3.9: Instância SVM1-2, solução ótima encontrada. Os quadrados representam os pontos B, os pontos circulares grandes representam os pontos D e os menores representam os pontos O. As setas apontam para o interior do poliedro.

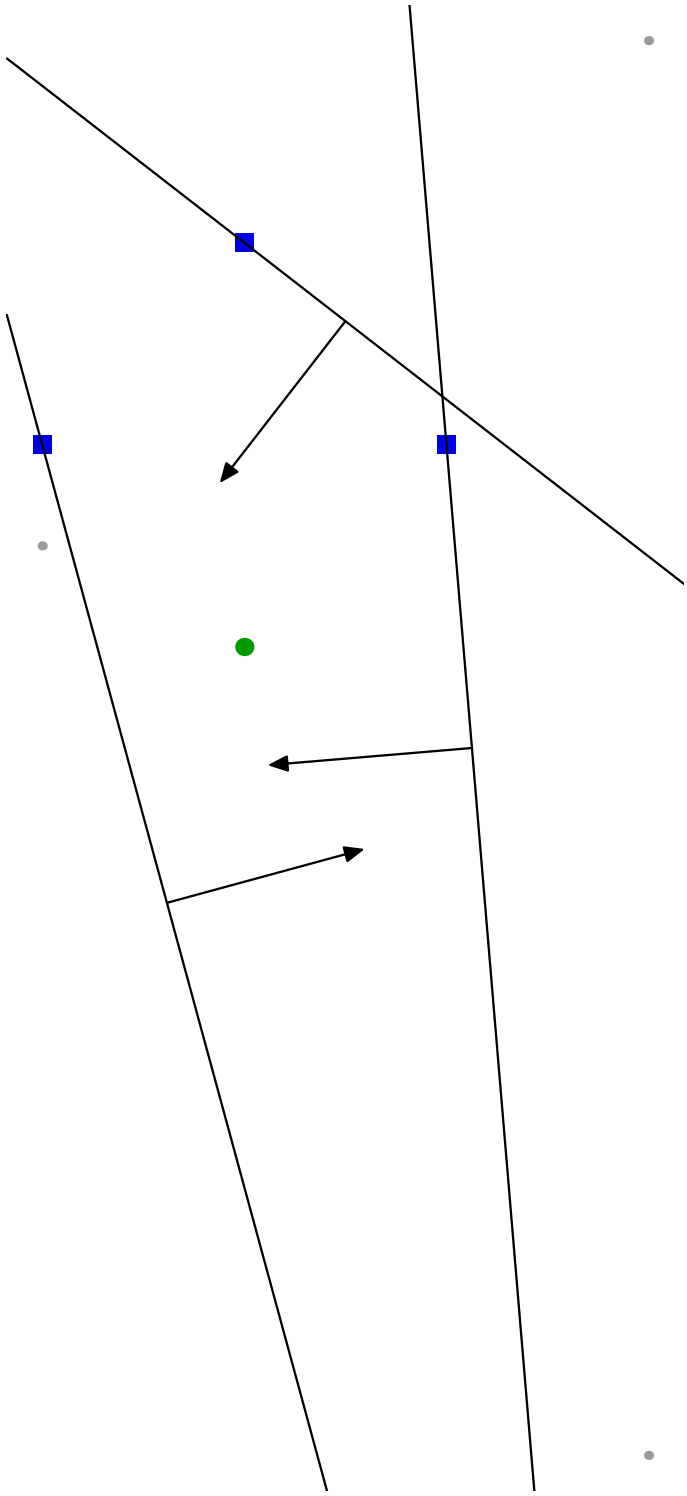


Figura 3.10: Instância SVM1-3, solução ótima encontrada. Os quadrados representam os pontos B, os pontos circulares grandes representam os pontos D e os menores representam os pontos O. As setas apontam para o interior do poliedro.

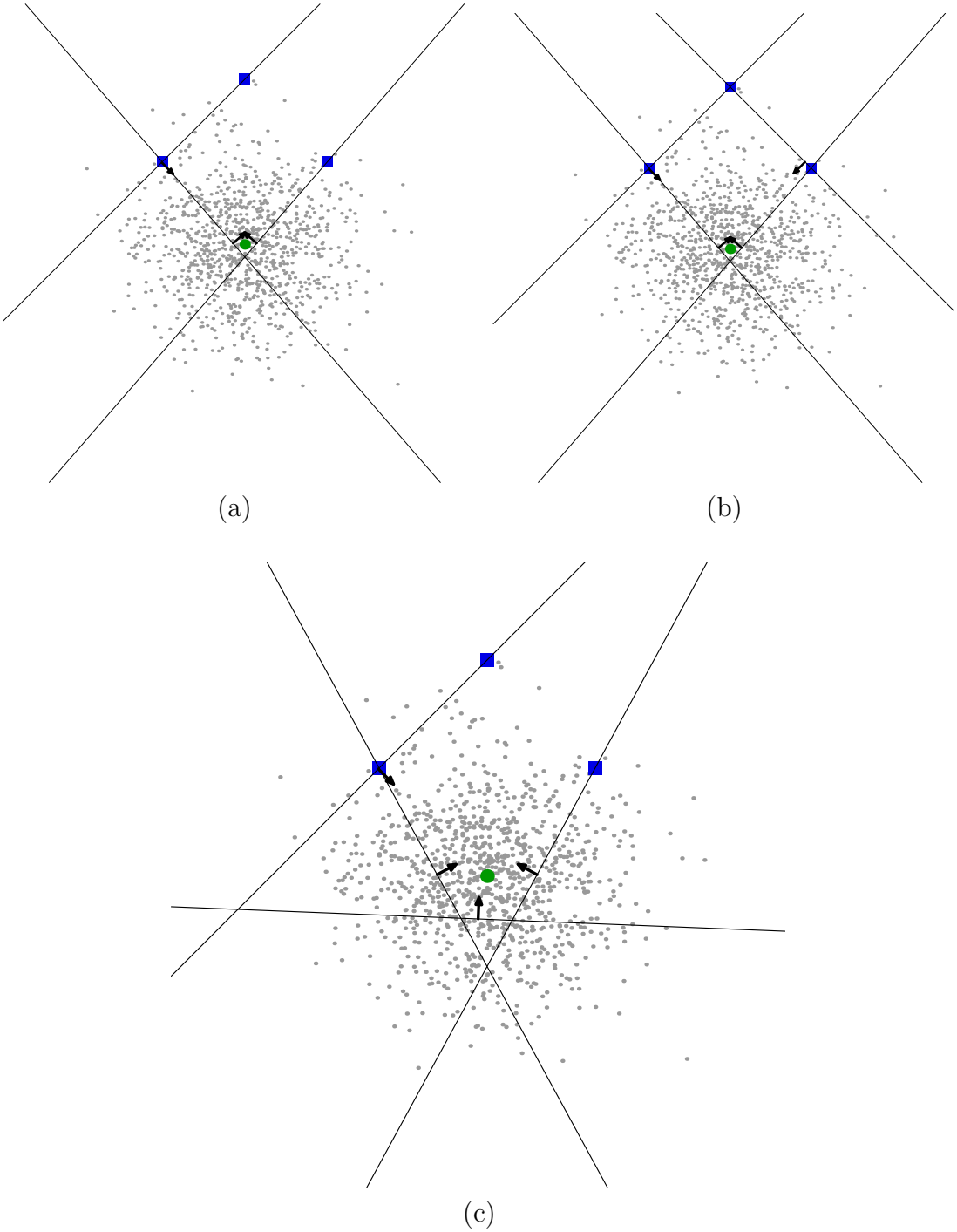


Figura 3.11: Instâncias (a) SVM2-3, (b) SVM2-4 e (c) SVM2-4'.

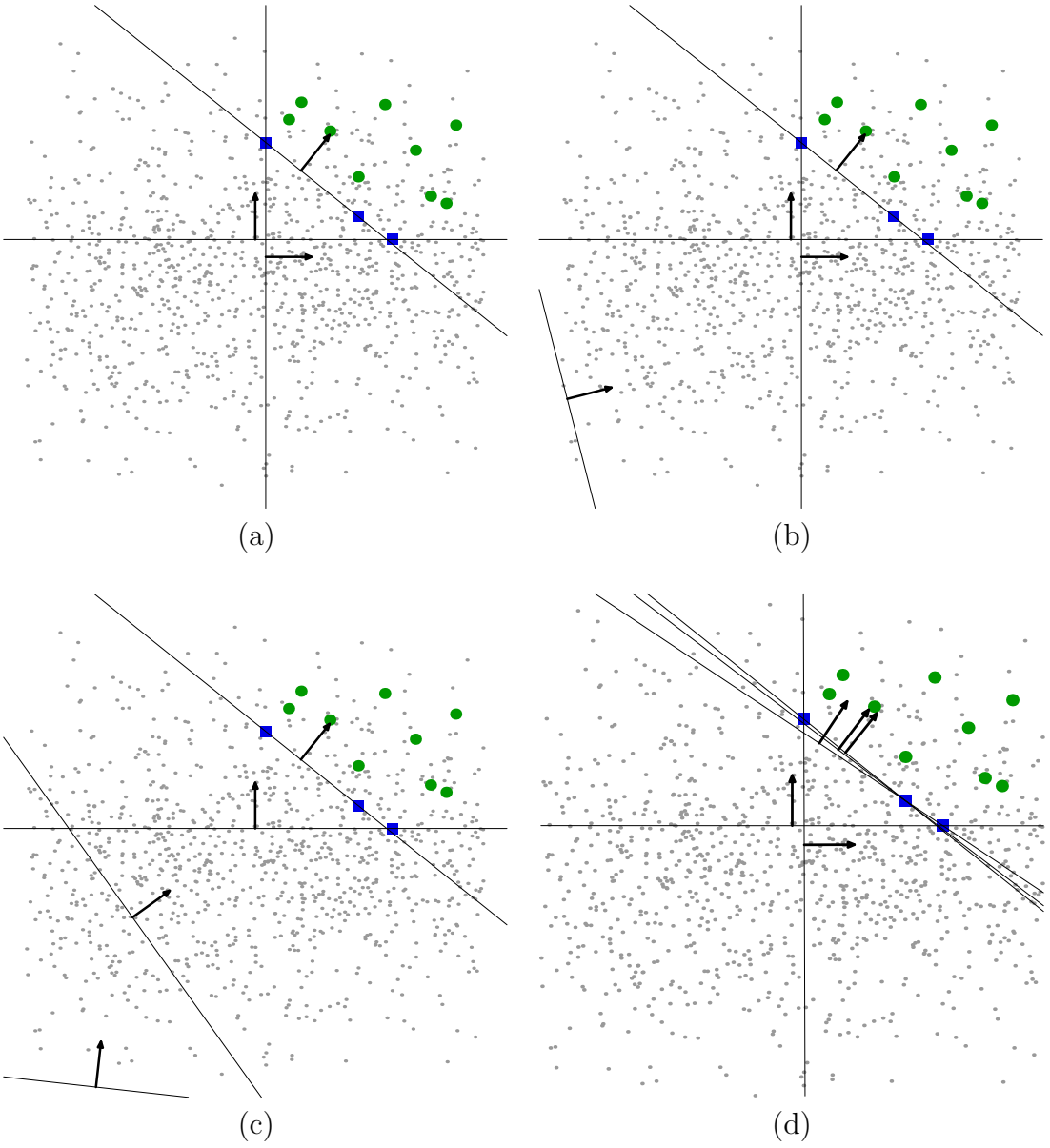


Figura 3.12: Instâncias (a) SVM3-3, (b) SVM3-4, (c) SVM3-5 com *Skinny* e (d) SVM3-5 com HOPSPACK (melhor). As coordenadas x estão associadas às notas de matemática e y às notas de português. Os eixos não foram desenhados.

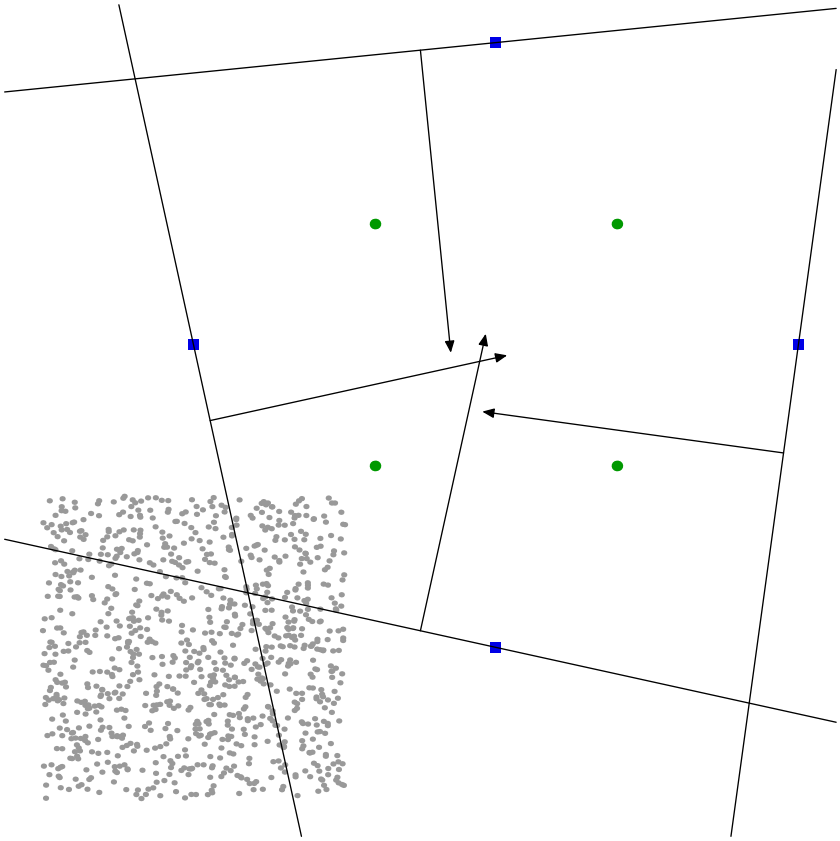


Figura 3.13: Instância SVM4-4.

Capítulo 4

Restauração Inexata sem derivadas

O Algoritmo 2.2 traz consigo uma combinação dos métodos de Restauração Inexata descritos em [Mar98, MP00, Mar01] com o método da tolerância flexível, de Paviani e Himmelblau [PH69]. Porém, o algoritmo proposto aproveita apenas as ideias principais, não utilizando nenhuma das ferramentas realmente existentes na teoria desenvolvida para métodos de Restauração Inexata.

Neste capítulo, preenchemos essa lacuna, apresentando um algoritmo para otimização sem derivadas que utiliza toda a estrutura de Restauração Inexata apresentada recentemente em [FF10]. Nosso foco são os problemas de otimização nos quais a função objetivo não possui derivadas disponíveis, mas as restrições possuem. Além disso, também supomos que o custo computacional de avaliar a função objetivo é muito maior do que o custo de avaliar as restrições, o que sustenta a ideia de que minimizar é um processo mais caro do que encontrar um ponto viável.

Nos algoritmos de Restauração Inexata, cada iteração é dividida em 2 fases. Na fase de restauração, o objetivo é encontrar um ponto menos inviável sem perder demais a otimalidade conseguida até o momento. Na fase de otimização, para não perder muito a viabilidade, explora-se o espaço tangente das restrições em busca de um ponto que minimize a função objetivo (ou função Lagrangiana, etc...). Para aceitar o novo ponto utiliza-se uma função de mérito [MP00, Mar01], direção de descida [FF10], entre outros meios.

Com essa abordagem, os algoritmos baseados em Restauração Inexata conseguem explorar melhor a estrutura do problema quando comparados com outros métodos que trabalham simultaneamente com otimalidade e viabilidade, como Lagrangianos Aumentados [CGT91, LT02, ABMS07], por exemplo. A vantagem pode tornar-se ainda maior quando a tarefa de encontrar um ponto viável exige cálculos complexos.

Embora a separação em fases torne, sob certas condições, a Restauração Inexata mais recomendada do que Lagrangianos Aumentados, os dois algoritmos se assemelham nas propriedades de convergência. Ao estendermos adequadamente o algoritmo de Fischer e Friedlander [FF10], herdamos a habilidade de gerar uma sequência satisfazendo condições sequenciais de otimalidade [MS03], como as obtidas para Lagrangianos Aumentados [ABMS07, AHM11]. Desta forma, o algoritmo proposto neste

capítulo enquadra-se em resultados recentes de convergência baseados em hipóteses bastante fracas [AHSS11a, AHSS11b].

Na Seção 4.1 apresentamos uma motivação para os algoritmos de Restauração Inexata e a razão pela qual os consideramos adequados para problemas sem derivadas. Na Seção 4.2 apresentamos o algoritmo de Restauração Inexata sem derivadas e os resultados de convergência a pontos estacionários, descritos detalhadamente em [Bue11, BFMS11].

4.1 Motivação

Consideremos o seguinte problema:

$$\begin{array}{ll} \min & f(x) \\ \text{s. a} & h(x) = 0 \\ & x \in \mathcal{X}, \end{array} \tag{4.1}$$

onde $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$ e $\mathcal{X} \subseteq \mathbb{R}^n$. A exemplo do Capítulo 2, o conjunto $\mathcal{X}$ representa as restrições que sabemos lidar de forma explícita (como caixas) ou que não podem ser relaxadas e, por isso, sempre deverão ser completamente satisfeitas.

Os algoritmos de Restauração Inexata (RI) surgiram na tentativa de contornar conhecidos inconvenientes dos algoritmos clássicos de otimização com restrições. Em uma iteração básica do algoritmo de Programação Quadrática Sequencial (PQS), por exemplo, para encontrar uma direção d de decréscimo, resolve-se o seguinte subproblema, gerado através de uma aproximação de (4.1):

$$\begin{array}{ll} \min\limits_{d} & f(x^k) + \nabla f(x^k)^T d + \frac{1}{2} d^T H_k d \\ \text{s. a} & h(x^k) + \nabla h(x^k)^T d = 0 \\ & x^k + d \in \mathcal{X} \\ & \|d\| \leq \Delta, \end{array} \tag{4.2}$$

onde $\nabla h : \mathbb{R}^n \to \mathbb{R}^{n \times p}$ representa a transposta da matriz Jacobiana de h, Δ está associado com uma região de confiança e H_k é uma aproximação da Hessiana de f ou da função Lagrangiana de (4.1) em x^k. Com base em uma função de mérito, decide-se se o novo ponto $x^k + d$ é aceito ou não [MS95]. A Figura 4.1(a) descreve uma iteração do método.

O primeiro inconveniente que surge com o algoritmo de Programação Quadrática Sequencial é que o problema (4.2) pode ser inviável. Para isso, basta que a linearização das restrições não possua intersecção com a região de confiança, como mostra a Figura 4.1(b). Essa situação foi discutida em [MS95].

Outro inconveniente aparece na construção dos modelos de f e h. Ambos são construídos sobre pontos que podem estar distantes do conjunto viável original. Além disso, não há nenhuma tentativa explícita de aumentar a viabilidade de x^k, caso este seja inviável.

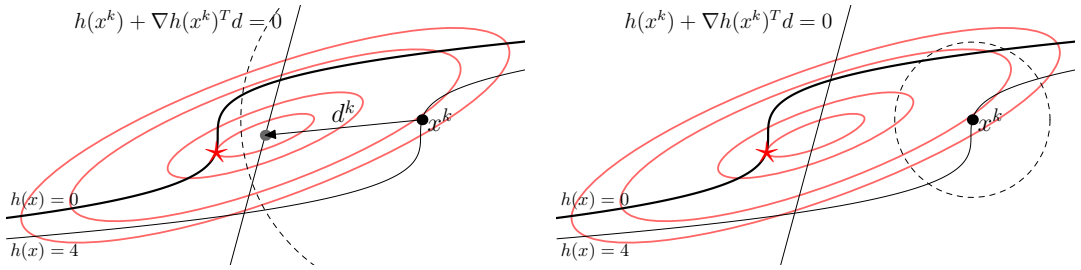


Figura 4.1: PQS aplicada à minimização de uma quadrática convexa com uma restrição. (a) Uma iteração do método é bem sucedida. (b) O subproblema pode ser inviável quando a linearização das restrições não intersecta a região de confiança. O minimizador global é dado por ★.

Por fim, o terceiro inconveniente, que é compartilhado com os algoritmos de penalidade, vem do fato da otimalidade e viabilidade serem consideradas simultaneamente em cada iteração. Como podemos verificar no subproblema (4.2), a aproximação de f em x^k tem a finalidade de encontrar uma direção de decréscimo para a função objetivo e a linearização de h, por sua vez, baseia-se em um esquema tipo Newton para melhorar a viabilidade.

Quando conhecemos características especiais de um problema que facilitam o processo de minimização ou de viabilidade, não podemos introduzi-las facilmente em métodos que se baseiam exclusivamente na resolução de (4.2). Por conseguinte, tais métodos realizam um esforço desnecessário. Partindo dessa ideia, os métodos de Restauração Inexata dividem a iteração principal em duas partes: a fase de **restauração** e a fase de **otimização**.

Na fase de restauração, o objetivo é encontrar um ponto y^k que reduza a inviabilidade associada às restrições h. O ponto restaurado não deve acarretar em um aumento descontrolado da função objetivo.

Na fase de otimização, o objetivo é reduzir o valor de f sem perder muito a viabilidade obtida na fase de restauração. Uma maneira de fazer isso é resolver um problema similar a (4.2):

$$\begin{aligned} \min_{d} \quad & f(y^k) + \nabla f(y^k)^T d + \tfrac{1}{2} d^T H_k d \\ \text{s. a} \quad & h(y^k) + \nabla h(y^k)^T d = h(y^k) \\ & y^k + d \in \mathcal{X}, \end{aligned} \tag{4.3}$$

com a diferença que é construído em torno do *ponto restaurado* y^k. Essa mudança tenta solucionar os dois primeiros inconvenientes citados, dado que (4.3) tem sempre solução ($d = 0$) e o ponto y^k é obtido através da fase de restauração. Note que a linearização de h tangencia a curva de nível dada por $h(y^k)$ e não mais dada por 0, como em (4.2). Uma iteração típica de Restauração Inexata é ilustrada na Figura 4.2.

Encontrada a direção d^k, solução do subproblema (4.3), o novo ponto $y^k + d^k$ é aceito ou não com base em diferentes critérios encontrados na literatura: região de

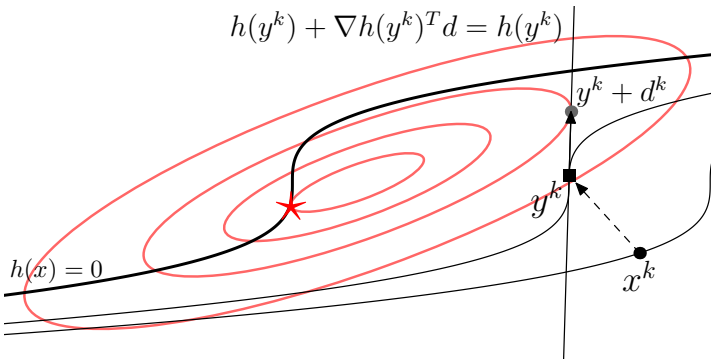


Figura 4.2: Uma iteração do algoritmo de Restauração Inexata aplicada na minimização de uma quadrática convexa com uma restrição. O ponto x^k é restaurado no ponto y^k. Em seguida, o ponto $y^k + d^k$ é aquele que minimiza a função objetivo na linearização da restrição em y^k. O minimizador global é dado por $\star$.

confiança [MP00, Mar01], busca linear [FF10] ou filtros [GKV03, KGR10]. A flexibilidade dos algoritmos de Restauração Inexata pode ser observada no fato de que qualquer procedimento pode ser usado nas fases de restauração e otimização. O objetivo sempre é explorar ao máximo as características específicas do problema.

Diversos problemas possuem características especiais que os tornam ideais para o esquema do tipo Restauração Inexata. Um exemplo, apresentado em [ACC+09], são os problemas de programação em dois níveis. Nesses tipos de problema, um subconjunto das variáveis possui restrições associadas a outro problema de otimização, chamado problema do segundo nível. Por isso, minimizar a função objetivo é mais fácil do que encontrar um ponto viável, tarefa que envolve a resolução de um problema de otimização.

Suponha que o custo computacional da avaliação da função objetivo supera consideravelmente o custo de avaliação das restrições. Suponha também que estas formam um conjunto viável topologicamente complicado[1]. Nessas condições, estamos no campo ideal de aplicação dos algoritmos de Restauração Inexata. A fase de restauração é particularmente importante, pois somos capazes de melhorar a viabilidade sem o custo extra das avaliações de f. Dada a complexidade do conjunto viável, a superioridade ante os algoritmos que consideram simultaneamente otimalidade e viabilidade torna-se ainda mais evidente.

Devido à sua natureza prática, os problemas sem derivadas podem possuir função objetivo extremamente complexa e custosa. Problemas com tais tipos de função, aliados a conjuntos viáveis topologicamente complicados, formados por restrições simples, encaixam-se perfeitamente na filosofia da Restauração Inexata. O algoritmo de Restauração Inexata com busca linear apresentado recentemente por Fischer e Friedlander [FF10] (doravante denominado Fischer–Friedlander) simplificou e generalizou a

[1]No nosso sentido, totalmente informal, em um conjunto topologicamente complicado, a tarefa de encontrar um ponto que o pertença pode ser uma tarefa computacionalmente complexa.

teoria existente até o momento [MP00, Mar01]. Porém, na fase de otimização, um subproblema similar a (4.3) deve ser resolvido e uma condição associada à função objetivo deve ser satisfeita nos pontos calculados na fase de restauração.

Dada a ausência das derivadas da função objetivo, algumas modificações devem ser feitas no algoritmo Fischer–Friedlander para considerar problemas sem derivadas. No restante deste capítulo descrevemos o algoritmo baseado em Restauração Inexata para problemas sem derivadas com restrições.

4.2 Algoritmo de Restauração Inexata para problemas sem derivadas com restrições

O problema em consideração é o mesmo da seção anterior, copiado a seguir por praticidade:

$$\begin{aligned} \min \quad & f(x) \\ \text{s. a} \quad & h(x) = 0 \\ & x \in \mathcal{X}\,. \end{aligned} \tag{4.1}$$

Porém, supomos que as derivadas das restrições estão disponíveis e as derivadas da função objetivo, embora existam, não podem ser utilizadas. Também supomos que o conjunto $\mathcal{X}$ é um politopo limitado descrito por:

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid a_i^T x \leq b_i,\ i = 1, \ldots, m\}.$$

Note que as restrições de caixa também encontram-se em $\mathcal{X}$. A razão para tal tipo de especificidade é que os algoritmos adequados para resolver os subproblemas gerados na fase de otimização do algoritmo de Restauração Inexata têm facilidade com restrições lineares. Dado que as restrições de $\mathcal{X}$ serão tratadas explicitamente, sem aproximações e relaxamentos, também podemos relacionar este conjunto com os conjuntos não-relaxáveis de [ADJ09].

Baseado no algoritmo Fischer–Friedlander [FF10], o Algoritmo 4.1 descrito a seguir possui modificações essenciais para problemas sem derivadas. Supondo que avaliar as restrições é consideravelmente menos custoso do que avaliar a função objetivo, tentamos evitar ao máximo as avaliações de f quando procuramos por um ponto mais viável. Por esse motivo, a função objetivo só é utilizada na fase de otimização, quando necessitamos algum tipo de decréscimo do valor funcional.

No começo de toda iteração do Algoritmo 4.1 (passo 1), temos em mãos o ponto x^k que foi calculado na iteração anterior (ou fornecido pelo usuário na iteração inicial). Estamos na fase de restauração, na qual o algoritmo procura por um ponto $y^k \in \mathcal{X}$ que satisfaça as seguintes condições:

$$\|h(y^k)\|_2 \leq r\|h(x^k)\|_2 \tag{4.4}$$

$$\|y^k - x^k\|_2 \leq \beta\|h(x^k)\|_2, \tag{4.5}$$

onde $r \in [0, 1)$ e $\beta > 0$ são duas constantes algorítmicas. Qualquer algoritmo pode ser usado para encontrar tal ponto, mas devemos evitar aqueles que necessitam da função objetivo. Sob tais condições esperamos que o ponto restaurado não esteja muito distante de x^k e, de certa forma, mantenha a otimalidade conseguida até o momento, reduzindo a inviabilidade.

Em [FF10] foram utilizadas condições um pouco mais fracas, mas que necessitavam da avaliação da função objetivo para serem verificadas. No mesmo trabalho, os autores forneceram condições suficientes para (4.4) e (4.5), ou seja, condições que garantem a boa definição da fase de restauração. Tais condições são, basicamente, que h seja continuamente diferenciável em $\mathcal{X}$ e que a condição de qualificação de Magasarian-Fromovitz [MF67] seja válida em todos os pontos do conjunto viável.

Para ser possível provar a convergência e a boa definição do Algoritmo 4.1 precisamos garantir que uma combinação adequada da função objetivo com a inviabilidade é decrescida ao longo das iterações. Uma medida que relaciona esses dois critérios (viabilidade e otimalidade) está associada com a **função de mérito**:

$$\Phi(x, \theta) = \theta f(x) + (1 - \theta)\|h(x)\|_2,$$

para $\theta \in (0, 1)$. O parâmetro θ pode ser visto como um parâmetro de controle que, se próximo de 1, preza por uma redução na função objetivo e, ao se aproximar de zero, prioriza por ganhos na viabilidade.

Com os pontos x^k (vindo da iteração anterior) e y^k calculados, no passo 2 o parâmetro θ_k é atualizado. O ponto x^k está associado a θ_k, que também é proveniente da iteração anterior. Logo, para calcular x^{k+1}, precisamos encontrar primeiramente θ_{k+1}. Para a atualização, verificamos se x^k, θ_k e y^k satisfazem

$$\Phi(y^k, \theta_k) - \Phi(x^k, \theta_k) \leq \frac{1}{2}(1 - r)\left(\|h(y^k)\|_2 - \|h(x^k)\|_2\right), \tag{4.6}$$

onde r é o mesmo da condição (4.4). Se (4.6) for satisfeita, nada precisa ser feito, ou seja, $\theta_{k+1} = \theta_k$, caso contrário θ_{k+1} é dado por

$$\theta_{k+1} = \frac{(1 + r)\left(\|h(x^k)\|_2 - \|h(y^k)\|_2\right)}{2\left[f(y^k) - f(x^k) + \|h(x^k)\|_2 - \|h(y^k)\|_2\right]}. \tag{4.7}$$

A equação de atualização (4.7) descreve exatamente o valor que θ_k deveria ter para, em conjunto com x^k e y^k, satisfazer a condição (4.6). Com (4.7), conseguimos demonstrar que a sequência $\{\theta_k\}$ é não crescente. Mais ainda: θ_k decresce de forma mais lenta do que se o processo descrito em [FF10] fosse utilizado.

No passo 3, entramos na fase de otimização. Utilizamos x^k, y^k e θ_{k+1} para encontrar o ponto x^{k+1} que resulte em um decréscimo controlado da função de mérito. Também necessitamos de um decréscimo suficiente da função objetivo, comandado pela constante algorítmica $\gamma > 0$. O ponto x^{k+1}, por conseguinte, é definido por $y^k + d^k$. Para calcular a direção d^k, dados y^k e o termo regularizador $\mu \geq \gamma$, primeiramente

encontramos uma solução aproximada $d \in \mathbb{R}^n$ do subproblema:

$$\begin{aligned} \min_{d} \quad & f(y^k + d) + \mu \|d\|_2^2 \\ \text{s. a} \quad & \nabla h(y^k)^T d = 0 \\ & y^k + d \in \mathcal{X}\,. \end{aligned} \tag{4.8}$$

O subproblema (4.8) é um problema de otimização sem derivadas com restrições lineares. As restrições lineares são formadas pelo conjunto $\mathcal{X}$ e pela linearização de h no ponto restaurado y^k:

$$h(y^k) + \nabla h(y^k)^T d = h(y^k).$$

A introdução do termo regularizador μ na função objetivo permite-nos encontrar uma direção de decréscimo da função objetivo sem resolver problemas do tipo (4.3), para os quais as derivadas de f são necessárias.

Com uma solução aproximada d do subproblema (4.8) em mãos, verificamos se as seguintes condições são satisfeitas:

$$f(y^k + d) \leq f(y^k) - \gamma \|d\|_2^2 \tag{4.9}$$

$$\Phi(y^k + d, \theta_{k+1}) \leq \Phi(x^k, \theta_{k+1}) + \frac{1}{2}(1 - r)\left(\|h(y^k)\|_2 - \|h(x^k)\|_2\right). \tag{4.10}$$

A condição (4.9) pede um decréscimo suficiente da função objetivo (pois $\gamma > 0$), enquanto a condição (4.10) pede um decréscimo suficiente da função de mérito. Note que o termo $\|h(y^k)\|_2 - \|h(x^k)\|_2$ é negativo, pela condição (4.4). Se (4.9) e (4.10) são satisfeitas, então $d^k = d$ e o ponto x^{k+1} está calculado, marcando o fim de uma iteração.

Por outro lado, pode ocorrer de uma das duas condições não ser satisfeita. Observe que o subproblema (4.8) possui ao menos um ponto viável: $d = 0$. Além disso, $d = 0$ satisfaz (4.9) e (4.10) (basta verificar a maneira como atualizamos θ_{k+1} para satisfazer a condição (4.6)). Percebe-se também que μ está inversamente associado a $\|d\|_2$. Desta forma, é de se esperar que, para valores suficientemente grandes de μ, exista d suficientemente pequeno satisfazendo as condições necessárias. Por isso, neste caso de fracasso, aumentamos o valor de μ, de acordo com a fórmula

$$\mu \in [\alpha_l \mu, \alpha_u \mu], \tag{4.11}$$

para $\alpha_u \geq \alpha_l > 1$, e resolvemos novamente o subproblema (4.8).

Devido ao alto custo da função objetivo (por definição), esse processo é o gargalo do Algoritmo 4.1 e deve ser tratado com cuidado. Felizmente, há algoritmos eficientes capazes de resolver problemas sem derivadas do tipo (4.8), como GSS [KLT03, KLT06b], descrito na Seção 1.4. São exatamente as características do algoritmo GSS que são necessárias para a análise de convergência, discutida a seguir. Uma implementação desse algoritmo também será usada na implementação numérica do Algoritmo 4.1.

Além de resolver o subproblema de forma eficiente, para reduzir o custo computacional da fase de otimização devemos escolher cuidadosamente os valores de μ. Valores grandes trazem maior garantia de que as condições (4.9) e (4.10) serão aceitas

Algoritmo 4.1: Restauração Inexata sem derivadas

Dados: $r \in [0,1)$, $\beta > 0$, $\bar{\mu} \geq \gamma > 0$ e $1 < \alpha_l \leq \alpha_u$
Entrada: $x^0 \in \mathcal{X}$ e $\theta_0 \in (0,1)$
$k \leftarrow 0$

1. **Fase de restauração**
   Encontre $y^k \in \mathcal{X}$ que satisfaça as condições (4.4) e (4.5).
2. **se** a condição (4.6) for satisfeita por x^k, θ_k e y^k **então**
   | Faça $\theta_{k+1} = \theta_k$.
   senão
   | Atualize θ_{k+1} de acordo com (4.7).
3. **Fase de otimização**
   Escolha $\mu \in [\gamma, \bar{\mu}]$
3.1. Encontre uma solução aproximada $d \in \mathbb{R}^n$ do subproblema (4.8).
3.2. **se** as condições (4.9) e (4.10) são satisfeitas **então**
   | $d^k = d$
   | $\mu_k = \mu$
   senão
   | Atualize μ de acordo com (4.11) e volte para o passo 3.1.
4. $x^{k+1} = x^k + d^k$
   Faça $k \leftarrow k + 1$ e volte para o passo 1.

rapidamente, mas estão associadas a pequenos passos ($\|d\|_2$ pequena). Em contrapartida, valores pequenos de μ resultam em passos grandes e uma possível convergência rápida, sob o risco de muitas resoluções do subproblema (4.8) serem necessárias. Consideramos a escolha correta de μ crucial no bom desempenho de qualquer implementação do Algoritmo 4.1. No Capítulo 5, apresentamos uma maneira adequada de escolhermos seus valores.

Após termos descrito todos os passos do algoritmo de Restauração Inexata sem derivadas, discutimos a seguir suas propriedades teóricas. Apresentamos neste trabalho apenas os enunciados dos teoremas associados ao Algoritmo 4.1, pois as demonstrações completas foram feitas na tese [Bue11] e também no manuscrito [BFMS11]. A principal contribuição do presente trabalho encontra-se na implementação correta do Algoritmo 4.1 e nos experimentos numéricos realizados no Capítulo 5.

Nossa análise de convergência segue o mesmo caminho de [FF10], começando com a demonstração de que todos os passos do Algoritmo 4.1 estão bem definidos. Para isso, necessitamos de duas hipóteses iniciais: que sempre é possível encontrar um ponto na fase de restauração que satisfaça as condições (4.4) e (4.5), e que a função objetivo satisfaz uma condição de Lipschitz.

Hipótese R1 – Sempre é possível encontrar um ponto $y^k \in \mathcal{X}$, no passo 1, tal que as condições (4.4) e (4.5) são satisfeitas.

Hipótese R2 – Existe $L_f > 0$ tal que

$$|f(x) - f(y)| \leq L_f \|x - y\|_2,$$

para todo $x, y \in \mathcal{X}$.

4.1 Lema

Suponha que as hipóteses R1 e R2 são válidas. Então, para todo $k \in \mathbb{N}$, os passos 1 e 2 estão bem definidos. Além disso, a sequência $\{\theta_k\}$ é não crescente, a desigualdade

$$\Phi(y^k, \theta_{k+1}) - \Phi(x^k, \theta_{k+1}) \leq \frac{1}{2}(1 - r)\left(\|h(y^k)\|_2 - \|h(x^k)\|_2\right)$$

é válida para todo k e existe $\bar{\theta} > 0$ tal que $\theta_k \downarrow \bar{\theta}$.

O Lema 4.1 garante a boa definição dos passos 1 e 2 do Algoritmo 4.1. Além disso, ele também mostra que o termo θ_k sempre encontra-se suficientemente longe de 0, o que implica que nunca estaremos considerando apenas a viabilidade na função de mérito. Um resultado auxiliar do Lema 4.1, que aparece durante as demonstrações, é que o valor de $\bar{\theta}$ está inversamente relacionado com β e L_f, ou seja, quanto maiores os valores de β e L_f, mais próximo de 0 está $\bar{\theta}$.

O Lema 4.2, apresentado a seguir, garante a boa definição do passo 3. Para demonstrá-lo, são necessárias duas hipóteses adicionais. A Hipótese R3 pede que as derivadas de h satisfaçam uma condição de Lipschitz.

Hipótese R3 – Existe $L_h > 0$ tal que

$$\|h(y + d)\|_2 \leq \|h(y)\|_2 + L_h \|d\|_2^2$$

para todo $d \in \mathbb{R}^n$ e todo $y \in \mathcal{X}$ tais que $y + d \in \mathcal{X}$ e $\nabla h(y)^T d = 0$.

A outra hipótese é necessária para garantir que o valor da função objetivo no ponto $y^k + d$, onde d é uma solução aproximada do subproblema (4.8), não seja estritamente maior do que $f(y^k)$. Tal hipótese é fácil de ser satisfeita, dado que $d = 0$ é uma solução válida. Uma maneira simples de satisfazê-la na prática é utilizarmos $d = 0$ como ponto inicial na resolução do subproblema (4.8) e aplicarmos um método de resolução aproximada que não devolva pontos com valores funcionais maiores que o do ponto inicial.

Hipótese R4 – Para todo $k \in \mathbb{N}$, a solução aproximada $d \in \mathbb{R}^n$, calculada no passo 3.1 do Algoritmo 4.1, é viável do subproblema (4.8) e satisfaz

$$f(y^k + d) + \mu \|d\|_2^2 \leq f(y^k).$$

4.2 Lema

Suponha que as hipóteses R1–R4 são válidas. Então, para cada iteração do Algoritmo 4.1, após um número finito de aumentos de μ (dados por (4.11)), as condições (4.9) e (4.10) são satisfeitas. Além disso, existe $\mu_{\max} > 0$ tal que $\mu_k \leq \mu_{\max}$, para todo $k \in \mathbb{N}$.

Ao garantir que μ é aumentado um número finito de vezes em cada iteração do Algoritmo 4.1, o Lema 4.2 também garante que o subproblema (4.8) é resolvido um número finito de vezes. Esse resultado é importante, pois a fase de otimização é a mais custosa do algoritmo. Além disso, como dissemos na discussão do algoritmo, se um valor suficientemente grande de μ for escolhido, digamos $\mu = \mu_{\max}$, as condições (4.9) e (4.10) serão sempre satisfeitas na primeira verificação. Infelizmente, tal escolha acarreta em passos pequenos a cada iteração.

Após mostrarmos que o Algoritmo 4.1 está bem definido, mostramos no Teorema 4.3 que os pontos limites são viáveis.

4.3 Teorema

Suponha que as hipótese R1–R4 são válidas. Se $\{x^k\}$, $\{y^k\}$ e $\{d^k\}$ são sequências geradas pelo Algoritmo 4.1, então:

$$\lim_{k\to\infty} \|h(x^k)\|_2 = \lim_{k\to\infty} \|h(y^k)\|_2 = \lim_{k\to\infty} \|d^k\|_2 = 0.$$

Uma consequência do Teorema 4.3 é que, se y^* é um ponto de acumulação da sequência $\{y^k\}$, gerada pelo Algoritmo 4.1, então, pela equação (4.5), temos que y^* também é um ponto de acumulação de $\{x^k\}$. Por esse motivo, como resultado do principal teorema de convergência do Algoritmo 4.1, mostramos que y^* é um ponto KKT do problema (4.1). Consequentemente, x^* também o é.

Para este teorema final, são necessárias mais duas hipóteses. A primeira é bastante comum na literatura de otimização sem derivadas e supõe que ∇f existe e é Lipschitz com constante L_g, embora não utilizemos o gradiente na prática. A segunda, dada pela Hipótese R5, descreve exatamente o que queremos dizer com "resolver aproximadamente" o subproblema (4.8).

Hipótese R5 – A direção $d^k \in \mathbb{R}^n$, calculada no passo 3 do Algoritmo 4.1, é um ponto viável e ε_k-KKT do subproblema (4.8), ou seja, $y^k + d^k \in \mathcal{X}$ e existem $u^k \in \mathbb{R}^m_+$ e $v^k \in \mathbb{R}^p$ tais que

$$\begin{aligned}
\left\| \nabla f(y^k + d^k) - 2\mu_k d^k + \nabla h(y^k) v^k + \sum_{i=1}^{m} u_i^k a_i \right\|_2 &\leq \varepsilon_k \\
\nabla h(y^k)^T d^k &= 0 \\
a_i^T (y^k + d^k) - b_i \leq 0 \quad i = 1, \ldots, m & \\
a_i^T (y^k + d^k) - b_i < -\varepsilon_k \;\Rightarrow\; u_i = 0 \quad i = 1, \ldots, m. &
\end{aligned}$$

A Hipótese R5 seria inadequada se fosse impossível de ser verificada. Sabemos, todavia, pela Seção 1.4, que o algoritmo GSS encontra pontos ε-KKT, onde ε está relacionado ao critério de parada $\alpha_{\min}$ do próprio algoritmo. Além disso, GSS foi desenvolvido para lidar eficientemente com problemas sem derivadas com restrições lineares. Desta forma, para satisfazer a Hipótese R5, apenas precisamos tomar uma sequência $\left\{\alpha_{\min}^{(k)}\right\}_{k\in\mathbb{N}} \to 0$, onde $\alpha_{\min}^{(k)}$ é o critério de parada para resolver os subproblemas (4.8) com GSS na iteração k, e escolher adequadamente ε_k relacionado com $\alpha_{\min}^{(k)}$. Essa é a ideia por trás do Teorema 4.4 apresentado a seguir.

4.4 Teorema
Suponha que as hipóteses R1–R5 são válidas e que ∇f é Lipschitz contínuo com constante L_g. Se y^* é um ponto limite da sequência $\{y^k\}$ gerada pelo Algoritmo 4.1, então y^* é AKKT [AHM11].

A condição AKKT em conjunto com uma condição de qualificação das restrições é uma condição de otimalidade para problemas de otimização contínua, discutida em [AHM11]. Por [AMS05], se y^* satisfaz também a condição de qualificação CPLD, então y^* é um ponto KKT do problema original (4.1). Como CPLD é uma condição de qualificação fraca, implica que condições mais fortes, como independência linear ou Mangasarian-Fromovitz, possam ser utilizadas. O resultado do Teorema 4.4 permite que outras condições de qualificação, mais fracas do que CPLD, sejam utilizadas [AHSS11a, AHSS11b]. Em [BFMS11] o Teorema 4.4 é enunciado e demonstrado utilizando a condição de otimalidade sequencial AGP [MS03].

Na demonstração do Teorema 4.4 pode-se obter uma informação importante: ε_k, definido na Hipótese R5, está relacionado com a viabilidade do ponto restaurado y^k e com o tamanho do passo d^k, calculado na fase de otimização. Essa informação será útil durante a implementação numérica no Capítulo 5, para sugerir critérios de parada e de atualização inteligentes.

O uso de uma hipótese como a Hipótese R5 concede-nos liberdade na escolha do algoritmo que será utilizado para resolver aproximadamente o subproblema (4.8). O algoritmo GSS tem a vantagem de ter sido desenvolvido para problemas sem derivadas e, portanto, preza por evitar ao máximo avaliações desnecessárias da função objetivo. Consequentemente, utilizando GSS podemos resolver problemas nos quais as derivadas da função objetivo não estão disponíveis, embora as derivadas das restrições sejam necessárias.

É importante destacar que o Algoritmo 4.1 não é um algoritmo exclusivo para problemas sem derivadas. Ele parte do pressuposto que a fase de restauração é computacionalmente mais fácil do que a fase de otimização. No contexto deste trabalho, o uso em problemas sem derivadas é apenas uma possibilidade de aplicação. De fato, em problemas nos quais a avaliação de f é simples, mas o cômputo de ∇f é custoso, o algoritmo proposto neste capítulo também é uma alternativa viável. Basta encontrar métodos eficientes para lidar com os subproblemas (4.8), satisfazendo as hipóteses necessárias.

Em [Bue11], o mesmo algoritmo é apresentado sob o nome de Restauração Inexata sem busca linear, em comparação com o algoritmo de Fischer–Friedlander que realiza busca linear na fase de otimização.

Capítulo 5

Restauração Inexata sem derivadas: implementação e testes numéricos

Neste capítulo apresentamos uma implementação do Algoritmo 4.1 apresentado no Capítulo 4. Nosso objetivo é discutir alternativas eficientes para a implementação de cada passo, além de comparar a implementação resultante com outros algoritmos descritos na literatura.

Poucos são os algoritmos existentes na literatura que lidam com problemas sem derivadas com restrições gerais [LT02, LST02, ADJ06, AADJLD09, LLS10, LT10]. Esse número torna-se menor se consideramos aqueles cujas implementações estão disponíveis publicamente para serem utilizadas [CST98, BB05, Pla09, GK10, LD11]. Percebemos que os métodos baseados em Restauração Inexata são mais difíceis de ser disponibilizados completamente, por sua implementação adequada e eficiente se basear na utilização de outros algoritmos para resolver seus subproblemas. Na implementação que discutimos neste capítulo, os problemas foram eficientemente resolvidos, porém dois algoritmos internos foram utilizados: um na fase de restauração e outro na fase de otimização. A liberdade oferecida por esse tipo de algoritmo permite-nos adequá-lo corretamente ao tipo de problema que será resolvido.

Este capítulo segue uma estrutura semelhante à do Capítulo 3 e, para evitar repetições desnecessárias, como, por exemplo, a explicação dos gráficos de perfil de desempenho e *data profiles*, frequentemente nos referenciamos a ele. Primeiramente, na Seção 5.1, especificamos os tipos de comparações que foram feitos durante a implementação e nos testes. Na Seção 5.2, discutimos todos os detalhes práticos de cada passo do Algoritmo 4.1, que é colocado novamente neste capítulo, sob o nome de Algoritmo 5.1, para facilitar a consulta. Na Seção 5.3, apresentamos os resultados da comparação com algoritmos existentes na literatura e uma aplicação, cujas características são ideais para o algoritmo proposto.

5.1 Métodos de comparação e análise de resultados

Dois tipos de comparação são realizadas neste capítulo: entre diferentes versões preliminares do algoritmo de Restauração Inexata proposto no Capítulo 4, com o objetivo de escolhermos valores adequados para os seus parâmetros, e entre diferentes algoritmos desenvolvidos para problemas sem derivadas com restrições.

Os problemas utilizados para realizar tais comparações são os 105 problemas do conjunto de testes de Hock e Schittkowski [HS81], discutidos no Apêndice A.2. Há, originalmente, 116 problemas nessa coleção, mas retiramos 11 que não possuem restrições e/ou que estão incorretamente programados. Cada problema dessa coleção possui disponível o valor mínimo da função objetivo e alguns dos minimizadores globais conhecidos. Nos casos em que a solução não é conhecida, os valores disponibilizados são os melhores conhecidos.

Na Tabela 5.1, descrevemos as características dos 105 problemas selecionados. Nas primeiras quatro colunas, Prob. representa o número do problema, de acordo com o descrito em [HS81], Var. representa o número de variáveis, Des. representa o número de restrições de desigualdade e Ig. representa o número de restrições de igualdade. Os valores entre parênteses indicam quantas das restrições são lineares. Nos outros dois conjuntos de quatro colunas o significado é o mesmo.

Para analisar a qualidade das soluções encontradas, utilizamos a mesma definição da Seção 3.2. Supondo que $\bar{x}$ é o ponto encontrado pelo algoritmo e que $\bar{f} = f(\bar{x})$, dizemos que o algoritmo *resolveu* o problema se as seguintes condições foram satisfeitas:

$$\|h(\bar{x})\|_2 \leq \varepsilon_{\text{viab}}, \quad \bar{x} \in \mathcal{X} \quad \text{e} \quad \frac{\bar{f} - f_{\text{L}}}{\max\left\{1, |\bar{f}|, |f_{\text{L}}|\right\}} \leq \varepsilon_{\text{comp}}, \tag{5.1}$$

onde f_{L} é a melhor solução encontrada pelos algoritmos que estão sendo comparados e $\varepsilon_{\text{viab}}$ é o critério de viabilidade utilizado por todos os algoritmos para decidir se um ponto é viável ou não. Para o critério de comparação do valor funcional utilizamos $\varepsilon_{\text{comp}} = 10^{-1}$ ou $\varepsilon_{\text{comp}} = 10^{-8}$, dependendo do que desejamos analisar. A fórmula

$$\frac{\bar{f} - f_{\text{L}}}{\max\left\{1, |\bar{f}|, |f_{\text{L}}|\right\}},$$

que relaciona o valor funcional com o melhor encontrado, foi utilizada, por exemplo, nos testes numéricos de [GK10].

Para analisar e visualizar os resultados, utilizamos os gráficos de perfil de desempenho [DM02] e *data profile* [MW09]. O critério de desempenho escolhido foi o número de avaliações da função objetivo. Uma explicação detalhada do seu funcionamento encontra-se na Seção 3.2.

Convém mencionarmos que o perfil de desempenho compara o desempenho do algoritmo em determinado problema com o melhor desempenho para o mesmo problema, gerando medidas de eficiência e robustez. No *data profile* não há tais comparações e podemos ter ideia do esforço necessário para o algoritmo resolver um determinado número de problemas.

Prob.	Var.	Des.	Ig.	Prob.	Var.	Des.	Ig.	Prob.	Var.	Des.	Ig.
6	2	0	1	43	4	3	0	80	5	0	3
7	2	0	1	44	4	6(6)	0	81	5	0	3
8	2	0	2	46	5	0	2	83	5	6	0
9	2	0	1(1)	47	5	0	3	84	5	6	0
10	2	1	0	48	5	0	2(2)	86	5	10(10)	0
11	2	1	0	49	5	0	2(2)	87	6	0	4
12	2	1	0	50	5	0	3(3)	88	2	1	0
13	2	1	0	51	5	0	3(3)	89	3	1	0
14	2	1	1(1)	52	5	0	3(3)	90	4	1	0
15	2	2	0	53	5	0	3(3)	91	5	1	0
16	2	2	0	54	6	0	1(1)	92	6	1	0
17	2	2	0	55	6	0	6(6)	93	6	2	0
18	2	2	0	56	7	0	4	95	6	4	0
19	2	2	0	57	2	1	0	96	6	4	0
20	2	3	0	58	2	3	0	97	6	4	0
21	2	1(1)	0	59	2	3	0	98	6	4	0
22	2	2(1)	0	60	3	0	1	99	7	0	2
23	2	5(1)	0	61	3	0	2	100	7	4	0
24	2	3(3)	0	62	3	0	1(1)	101	7	6	0
26	3	0	1	63	3	0	2(1)	102	7	6	0
27	3	0	1	64	3	1	0	103	7	6	0
28	3	0	1(1)	65	3	1	0	104	8	6	0
29	3	1	0	66	3	2	0	105	8	1 (1)	0
30	3	1	0	68	4	0	2	106	8	6 (3)	0
31	3	1	0	69	4	0	2	107	9	0	6
32	3	1	1(1)	70	4	1	0	108	9	13	0
33	3	2	0	71	4	1	1	109	9	4 (2)	6
34	3	2	0	72	4	2	0	111	10	0	3
35	3	1(1)	0	73	4	2(1)	1(1)	112	10	0	3(3)
36	3	1(1)	0	74	4	2(2)	3	113	10	8 (3)	0
37	3	2(2)	0	75	4	2(2)	3	114	10	8 (4)	3(1)
39	4	0	2	76	4	3(3)	0	116	13	15 (5)	0
40	4	0	3	77	5	0	2	117	15	5	0
41	4	0	1(1)	78	5	0	3	118	15	29(29)	0
42	4	0	2	79	5	0	3	119	16	0	8(8)

Tabela 5.1: Descrição dos 105 problemas de Hock e Schittkowski. Os valores em parênteses representam quantas das restrições são lineares.

5.2 Implementação

O problema considerado pelo algoritmo em estudo é definido por

$$\begin{aligned} \min \quad & f(x) \\ \text{s. a} \quad & h(x) = 0 \\ & x \in \mathcal{X}, \end{aligned}$$

com $f : \mathbb{R}^n \to \mathbb{R}$ e $h : \mathbb{R}^n \to \mathbb{R}^p$. Supomos que as derivadas das restrições estão disponíveis, mas as derivadas de f, embora também possam estar disponíveis, não são utilizadas. No Capítulo 4, o conjunto $\mathcal{X}$ foi definido como um politopo limitado, descrito por restrições lineares de desigualdade. Na implementação aqui apresentada, consideramos apenas restrições de caixa

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid l \leq x \leq u\},$$

com $l, u \in \mathbb{R}^n$. As restrições lineares são tratadas da mesma maneira que as restrições gerais, pois o algoritmo utilizado na fase de restauração não trabalha explicitamente

com tais tipos de restrições, apenas caixas.

A restrições (lineares e não lineares) da forma $g_i(x) \leq 0$, $i = 1, \ldots, m'$, são convertidas em igualdades através da adição da variável de folga s_i e das novas restrições:

$$g_i(x) - s_i = 0 \quad \text{e} \quad s_i \leq 0,$$

para todo $i = 1, \ldots, m'$. Se g está de acordo com as hipóteses necessárias para a convergência, descritas no Capítulo 4, então tal adaptação não traz mudanças teóricas. Implicações de caráter prático aparecem, devido ao aumento do número de variáveis de n para $n + m'$.

O Algoritmo 5.1 que mostramos aqui é o Algoritmo 4.1, apresentado no Capítulo 4, com todas as definições e condições inseridas dentro do próprio algoritmo.

Como já mencionamos, a vantagem de descrevermos o algoritmo em alto nível é que há liberdade na escolha da técnica utilizada em cada passo. Podemos verificar na literatura exemplos bem sucedidos de aplicações da Restauração Inexata em problemas de dois níveis [Cas04, ACC$^+$09], problemas de engenharia [FG11] e problemas de complementaridade [FF10]. O próprio Algoritmo 5.1 proposto é um exemplo (em alto nível) de aplicação a problemas nos quais a restauração é consideravelmente mais fácil do que a otimização.

Dada a grande quantidade de parâmetros, muitos dos quais não temos uma noção intuitiva para sugestão de valor, realizamos nesta seção pequenos testes preliminares para verificar o comportamento do Algoritmo 5.1. Tais testes consistem na variação do parâmetro que desejamos estudar, mantendo os demais fixos. Essa estratégia parte da premissa (não necessariamente verdadeira) que se otimizamos cada parâmetro individualmente, otimizaremos o algoritmo por completo.

5.2.1 A fase de restauração

Dado x^k vindo da iteração anterior, para encontrar o ponto restaurado y^k, no passo 2 do algoritmo, resolvemos o seguinte problema de viabilidade:

$$\begin{aligned} \min_{y} \quad & \|y - x^k\|_2^2 \\ \text{s. a} \quad & h(y) = 0 \\ & y \in \mathcal{X}\,. \end{aligned} \tag{5.9}$$

O problema (5.9) é um problema de programação não linear com todas as derivadas disponíveis, dado que, por hipótese, as derivadas das restrições estão disponíveis. Logo, pode ser resolvido com um algoritmo clássico de programação não linear. Utilizamos o algoritmo baseado em Lagrangianos Aumentados denominado Algencan [ABMS07, TAN], para o qual x^k é fornecido como ponto inicial. Observe que y^k é calculado sem o uso de avaliações da função objetivo.

Todos os parâmetros originais de Algencan foram mantidos. O critério de parada por otimalidade foi fixado em 10^{-1}, pois não estamos interessados em minimizadores de (5.9), apenas em pontos viáveis. De fato, realizamos testes preliminares

Algoritmo 5.1: Restauração Inexata sem derivadas (Algoritmo 4.1)

Dados: $r \in [0,1)$, $\beta > 0$, $\bar{\mu} \geq \gamma > 0$ e $1 < \alpha_l \leq \alpha_u$

Entrada: $x^0 \in \mathcal{X}$ e $\theta_0 \in (0,1)$

1. $k \leftarrow 0$

2. **Fase de restauração**

Encontre $y^k \in \mathcal{X}$ que satisfaça as condições

$$\|h(y^k)\|_2 \leq r\|h(x^k)\|_2 \tag{5.2}$$

$$\|y^k - x^k\|_2 \leq \beta\|h(x^k)\|_2, \tag{5.3}$$

3. **se**

$$\Phi(y^k, \theta_k) - \Phi(x^k, \theta_k) \leq \frac{1}{2}(1-r)\left(\|h(y^k)\|_2 - \|h(x^k)\|_2\right) \tag{5.4}$$

então

| Faça $\theta_{k+1} = \theta_k$ e vá para o passo 4.

senão

$$\theta_{k+1} = \left[(1+r)\left(\|h(x^k)\|_2 - \|h(y^k)\|_2\right)\right] / \left[2\left(f(y^k) - f(x^k) + \|h(x^k)\|_2 - \|h(y^k)\|_2\right)\right] \tag{5.5}$$

4. **Fase de otimização**

Inicialize $\mu \in [\gamma, \bar{\mu}]$

3.1. Encontre uma solução aproximada $d \in \mathbb{R}^n$ do subproblema:

$$\begin{aligned} \min_{d} \quad & f(y^k + d) + \mu\|d\|_2^2 \\ \text{s. a} \quad & \nabla h(y^k)^T d = 0 \\ & y^k + d \in \mathcal{X}\,. \end{aligned} \tag{5.6}$$

4.2. **se** as seguintes condições são satisfeitas

$$f(y^k + d) \leq f(y^k) - \gamma\|d\|_2^2 \tag{5.7}$$

$$\Phi(y^k + d, \theta_{k+1}) \leq \Phi(x^k, \theta_{k+1}) + \frac{1}{2}(1-r)\left(\|h(y^k)\|_2 - \|h(x^k)\|_2\right) \tag{5.8}$$

então

| Faça $d^k = d$ e $\mu_k = \mu$.

senão

| Escolha $\mu \in [\alpha_l\mu, \alpha_u\mu]$ e volte para o passo 3.1.

5. $x^{k+1} = x^k + d^k$

Faça $k \leftarrow k+1$ e volte para o passo 1.

nesta fase, para comparar pontos advindos da resolução do problema de projeção dado por (5.9), com pontos advindos de um problema de viabilidade, ambos resolvidos com Algencan. Na Figura 5.1, verificamos que a primeira alternativa conduz a melhores

resultados. Embora o *data profile* indique que as duas alternativas resolvem aproximadamente o mesmo número de problemas com o mesmo número de avaliações de f, no gráfico de perfil de desempenho verificamos claramente que a projeção é a melhor alternativa. De fato, se mais de 40000 avaliações são permitidas, a estratégia de projeção atinge os 88 problemas resolvidos, contra 76 da estratégia de viabilidade. Além disso, na teoria de Restauração Inexata [Mar01], há resultados indicando que pontos restaurados com um esquema do tipo projeção ortogonal de x^k satisfazem as condições (5.2) e (5.3). Observe que (5.9) é um problema cujo minimizador global é uma "projeção ortogonal" de x^k no conjunto viável.

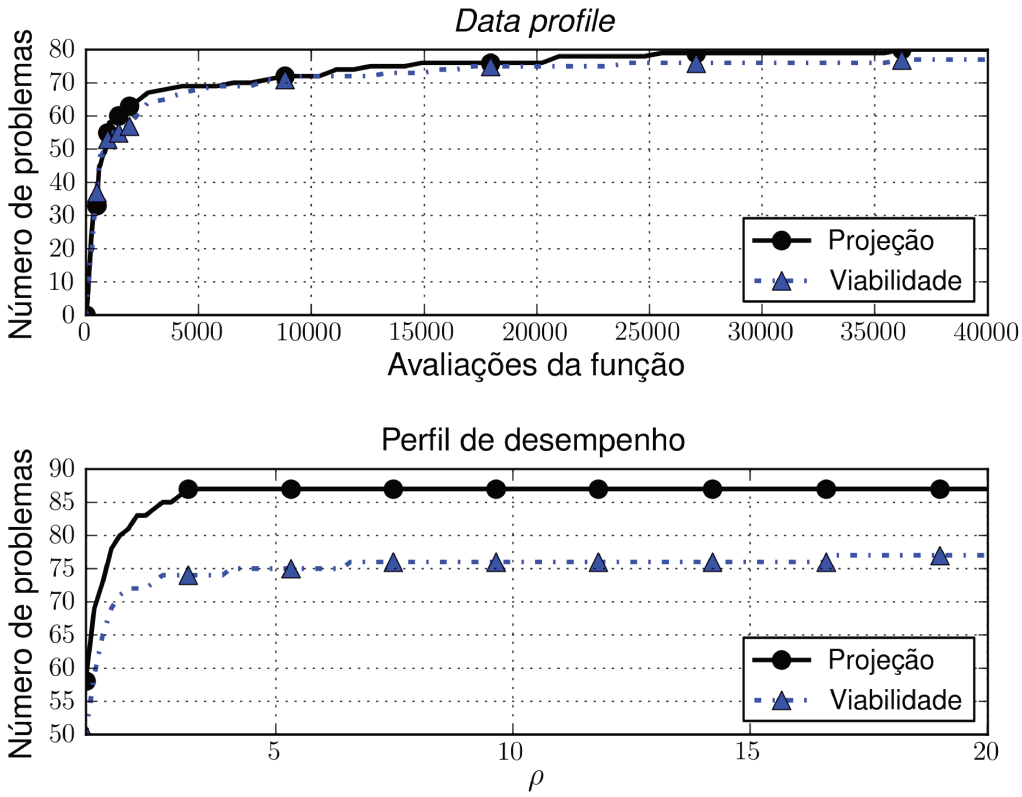


Figura 5.1: Comparação entre a restauração através da "projeção ortogonal" e através de um problema de viabilidade.

O critério de viabilidade de Algencan é escolhido para torná-lo mais exigente conforme x^k se aproxima da viabilidade e/ou da otimalidade. Tal escolha é razoável, dado que se x^k é viável, não podemos encontrar y^k inviável, pois claramente (5.2) não é verificada. Desta forma, definimos $\delta_{-1}^{\text{viab}} = 10^{-2}$ na inicialização e, no início da iteração k definimos

$$\delta_k^{\text{viab}} = \max\left\{\varepsilon_{\text{viab}}/\sqrt{n}, \min\left\{\delta_{k-1}^{\text{viab}}, \|h(x^k)\|_2\right\} \alpha_{\min}^{(k)}\right\}, \tag{5.10}$$

onde $\varepsilon_{\text{viab}}$ é a tolerância máxima para a inviabilidade, utilizada no algoritmo proposto, e $\alpha_{\min}^{(k)}$ é o critério de parada por otimalidade, que será utilizado por GSS no passo 3.1

da iteração k (veja a Seção 4.2). O termo $\varepsilon_{\text{viab}}/\sqrt{n}$, onde n é o número de variáveis do problema, tem por objetivo adequar o critério de parada de Algencan, que utiliza $\|\cdot\|_\infty$ (norma infinito) com o algoritmo proposto, que utiliza a norma Euclidiana. A atualização de $\alpha_{\min}^{(k)}$ é descrita na Subseção 5.2.3.

Na implementação do Algoritmo 5.1, as condições (5.2) e (5.3) não são verificadas. Há duas razões para ignorarmos tal verificação:

- com a definição de δ_k^{viab} dada por (5.10) e graças ao bom desempenho de Algencan, a condição (5.2) é quase sempre verificada com folga;
- ainda que y^k não satisfaça (5.2) ou (5.3), resultados numéricos mostraram que somente em casos extremos e particulares há problemas de convergência.

Em [BM05] os autores utilizaram $r = 0.99$ na implementação de um algoritmo baseado em Restauração Inexata com resultados de convergência super-linear. De acordo com a condição (5.2), utilizar $r = 0.99$ significa aceitar praticamente qualquer ponto que melhore a viabilidade de x^k. Em casos de falha na verificação de (5.2), geralmente por problemas na resolução de (5.9), ainda assim observamos que a continuação do algoritmo é aconselhada, pois permite uma leve mudança no ponto inicial utilizado por Algencan e pode resultar em uma restauração bem sucedida em iterações posteriores.

Observações semelhantes podem ser feitas sobre a condição (5.3). Percebemos que valores de β adequados variam de acordo com o problema a ser resolvido. Além disso, não há uma alternativa para modificar y^k em caso de fracasso. Como mostram os resultados numéricos na Seção 5.3, o funcionamento do Algoritmo 5.1 não é prejudicado com a exclusão da condição (5.3).

A escolha $r = 0.99$ aparenta ir contra nosso real desejo de que y^k seja o mais viável possível. Devemos observar, todavia, que r também é utilizado na condição (5.8) e uma escolha de r próxima de zero força um decréscimo maior na função de mérito do que se r próximo de 1 fosse utilizado. Por isso, tomando $r = 0.99$, mas, na realidade, satisfazendo (5.2) com folga, permitimos que um decréscimo pequeno na função de mérito seja suficiente para satisfazer (5.8). O resultado da escolha $r = 0.99$ é que passos maiores podem ser executados.

Uma última consideração a ser feita nessa fase é a de que Algencan é capaz de lidar explicitamente com as restrições de caixa que compõem o conjunto $\mathcal{X}$. As demais restrições são tratadas de forma igual e penalizadas na função de Lagrangianos Aumentados. Uma alternativa a Algencan encontra-se nos algoritmos que lidam explicitamente com restrições lineares, tais como Minos [MS78] ou o próprio Algencan com modificações [And08].

5.2.2 Atualização de θ_k

A atualização do termo θ_k, utilizado na função de mérito, é implementada exatamente como descrita no passo 3.

Pelo Lema 4.1, sabemos que $\{\theta_k\}$ é decrescente e limitada inferiormente. Isso indica que θ_k estabiliza-se na combinação adequada entre viabilidade e otimalidade. Para evitar que esse "ponto de equilíbrio" seja perdido, inicializamos $\theta_0 = 0.9$. Tal escolha também implica na preferência pelo decréscimo de f em detrimento do decréscimo da inviabilidade nas primeiras iterações, o que parece adequado.

5.2.3 A fase de otimização

Na fase de otimização, precisamos realizar duas tarefas de forma eficiente: resolver os subproblemas (5.6) e escolher valores adequados para o termo regularizador μ em cada iteração.

Resolução dos subproblemas

Para resolver os subproblemas sem derivadas com restrições lineares, utilizamos uma implementação do algoritmo GSS [KLT03], disponível no software HOPSPACK [Pla09]. HOPSPACK implementa um algoritmo baseado em Lagrangianos Aumentados, descrito em [KLT06a, GK10], para resolver problemas sem derivadas com restrições gerais. Em cada iteração do algoritmo, os subproblemas formados pela função penalidade e as restrições lineares do problema original são resolvidos com uma implementação de GSS. Utilizamos essa implementação para resolver (5.6).

Todos os parâmetros pré-definidos na implementação de GSS foram mantidos, com exceção do critério de parada por otimalidade, $\alpha_{\min}^{(k)}$ (veja as seções 1.4 e 4.2), definido, inicialmente, por $\alpha_{\min}^{0} = 0.5$ e, ao final da iteração k, por

$$\alpha_{\min}^{(k+1)} = \max\left\{10^{-16}, \min\left\{\alpha_k, 0.1 \max\left\{\|h(y^k + d^k)\|_2, \|d^k\|_2\right\}\right\}\right\}, \tag{5.11}$$

onde $\alpha_k = 0.5/(1.1)^k$.

A atualização dada por (5.11) é interessante em diversos aspectos. Em primeiro lugar, associa um maior rigor na otimalidade caso $x^{k+1} = y^k + d^k$ esteja próximo da viabilidade e caso o tamanho do passo d^k esteja próximo de zero. Pelos resultados dos teoremas 4.3 e 4.4 no Capítulo 4, os dois critérios estão associados à otimalidade dos pontos limites. Em outras palavras, se estamos nos aproximando de candidatos a minimizadores, a verificação de otimalidade deve ser feita com mais rigor.

Em segundo lugar, caso seja descoberto que não estamos mais nos aproximando de minimizadores, a atualização (5.11) permite que $\alpha_{\min}^{(k)}$ volte a crescer, poupando avaliações da função objetivo. Por fim, necessitamos que, independentemente do que aconteça, $\alpha_{\min}^{(k)}$ tenda a zero, por isso utilizamos o termo α_k, que converge lenta e constantemente a zero. Um teste simples foi realizado, com o objetivo de verificar qual taxa de decréscimo seria mais apropriada para α_k: 1.1 ou 1.5. Na Figura 5.2 observamos que 1.1 é a melhor escolha.

O parâmetro de decréscimo suficiente foi fixado em $\gamma = 2^{-20}$ e com isso pedimos pouca redução da função objetivo, pela condição (5.7). Além disso, como devemos ter $\mu \geq \gamma$, o fato de μ admitir valores próximos de zero mostrou-se uma

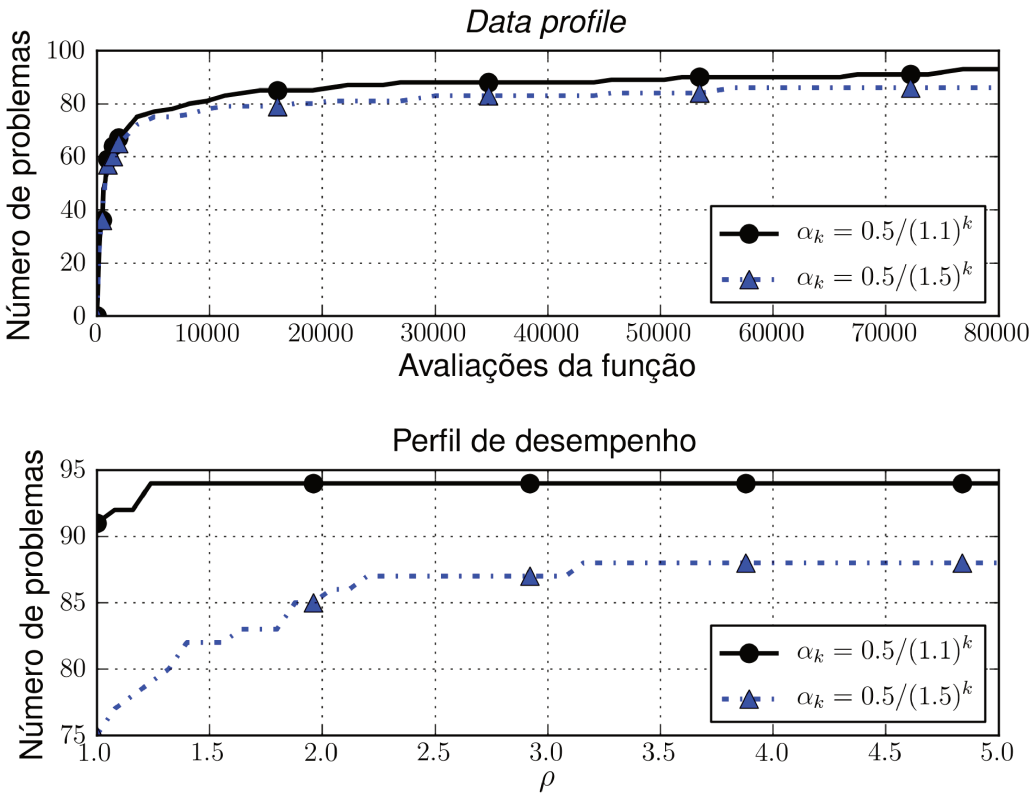


Figura 5.2: Comparação entre duas taxas de decréscimo para α_k. O gráfico de perfil de desempenho mostra que um decréscimo mais lento resulta em maior eficiência e maior robustez.

excelente atitude quando GSS encontra-se longe de uma solução do subproblema. Em conjunto com a escolha de $r = 0.99$, aumentamos as chances de que passos grandes sejam dados.

O termo regularizador μ

Com o uso de GSS na resolução do subproblema (5.6) temos uma maneira eficiente de encontrar uma direção d no passo 3.1. Porém, de nada adiantaria se esse passo tivesse que ser resolvido um grande número de vezes em cada iteração. O termo μ tem esse papel. Em toda iteração, iniciamos μ com um valor no intervalo $[\gamma, \bar{\mu}]$, por necessidades teóricas. Da mesma forma, em caso da não verificação de alguma das condições (5.7) ou (5.8), devemos aumentar μ respeitando o intervalo $[\alpha_l\mu, \alpha_u\mu]$, onde μ, neste caso, é o último valor utilizado, com o qual uma das condições não foi satisfeita.

Com base no Lema 4.2, sabemos que valores suficientemente grandes de μ levam a direções d que satisfazem as condições (5.7) e (5.8). Infelizmente, também sabemos que valores grandes de μ resultam em tamanhos de passo pequenos e, consequentemente, uma convergência lenta. Por outro lado, se desejamos passos grandes

podemos fazer μ próximo de zero. O custo dessa escolha é que o subproblema (5.6) pode ser resolvido muitas vezes, para valores crescentes de μ, até que as referidas condições sejam satisfeitas.

Apresentamos a seguir uma escolha e atualização de μ que respeita os limitantes impostos, para os valores $\bar{\mu} = 1.01 \cdot 10^{40}\gamma$, $\alpha_l = 10$ e $\alpha_u = 10^{10}$. Tal escolha tenta aproveitar toda a informação obtida com a resolução do subproblema (5.6), para fornecer atualizações de μ que se adequem às características do problema. Para construir a forma de atualização, nos baseamos na demonstração do Lema 4.2 que pode ser encontrada em [BFMS11, Lema 3.2] e cuja ideia reproduzimos aqui.

Dado que GSS é um algoritmo baseado em direções viáveis e com decréscimo suficiente, utilizando $d = 0$ (que satisfaz (5.7)) como ponto inicial para a resolução de (5.6), temos a garantia de que, se outra direção for encontrada, ela também satisfaz (5.7). Logo, devemos nos preocupar apenas com a condição (5.8).

Na demonstração do Lema (4.2) encontramos a seguinte desigualdade

$$\Phi(y^k + d, \theta_{k+1}) - \Phi(x^k, \theta_{k+1}) \leq$$

$$\frac{1}{2}(1-r)\left(\|h(y^k)\|_2 - \|h(x^k)\|_2\right) - \theta_{k+1}\mu\|d\|_2^2 + (1-\theta_{k+1})\left(\|h(y^k+d)\|_2 - \|h(y^k)\|_2\right).$$

Desta forma, dados x^k, θ_{k+1}, y^k, d e μ, pela desigualdade anterior, para que (5.8) seja satisfeita devemos ter

$$-\theta_{k+1}\mu\|d\|_2^2 + (1-\theta_{k+1})\left(\|h(y^k+d)\|_2 - \|h(y^k)\|_2\right) \leq 0. \tag{5.12}$$

Observe que tal condição só será verificada após a obtenção de d no passo 3.1. Porém, se nos fosse dada uma direção $d \neq 0$ qualquer, então, de acordo com (5.12), tomando μ da forma

$$\mu \geq \left(\frac{1-\theta_{k+1}}{\theta_{k+1}}\right)\frac{\|h(y^k+d)\|_2 - \|h(y^k)\|_2}{\|d\|_2^2}, \tag{5.13}$$

a condição (5.8) seria satisfeita. Como não temos a informação de d *a priori*, podemos utilizar (5.13) para fornecer informações para as próximas resoluções do subproblema, seja em caso de não verificação de uma das condições, seja para uma outra iteração do Algoritmo 5.1.

Em toda iteração do Algoritmo 5.1, é necessário inicializarmos o valor de μ antes da primeira resolução do subproblema (5.6). Infelizmente, d só está disponível após a resolução do subproblema, de forma que não podemos utilizar (5.13) diretamente. Ainda assim, utilizando a desigualdade (5.13) como base, descrevemos a seguir um esquema de atualização e inicialização de μ. Para facilitar o entendimento da iteração na qual cada valor é calculado, adicionamos, apenas aqui, um índice a μ, de forma que μ_k indica o valor que está sendo utilizado na iteração k. De forma análoga, indicamos por d^k a direção d encontrada com o uso de μ_k na função objetivo do subproblema. Tais modificações não alteram a descrição do Algoritmo 5.1.

Na primeira iteração ($k = 0$), não temos informação alguma advinda de iterações anteriores para sugerir um possível valor para μ_0. Por esse motivo, antes

da primeira resolução do subproblema (5.6), baseados em (5.13), inicializamos μ_0 da seguinte forma:

$$\mu_0 = 1.01 \min\{\max\{\gamma, \mu_0'\}, 10^{40}\gamma\}, \tag{5.14}$$

onde

$$\mu_0' = \begin{cases} \left(\dfrac{1-\theta_1}{\theta_1}\right) \dfrac{\|h(x^0)\|_2 - \|h(y^0)\|_2}{\|x^0 - y^0\|_2^2} & \text{, se } y^0 \neq x^0 \\ \gamma & \text{, se } y^0 = x^0 \end{cases}.$$

Assim, adicionamos a μ_0 um pouco de informação a respeito da função objetivo (com θ_1) e das restrições. No caso em que $y^0 = x^0$, permitimos que passos de tamanho arbitrário sejam dados.

Para toda iteração $k \in \mathbb{N}$, incluindo $k = 0$, logo após a primeira resolução do subproblema (5.6), já possuímos mais informações para atualizar μ_k, pois d^k foi calculada. Desta forma, após o passo 3.1 e antes de decidirmos se o subproblema será resolvido novamente ou se a direção d^k será aceita, calculamos os seguintes valores:

$$\mu_k'' = \begin{cases} \left(\dfrac{1-\theta_{k+1}}{\theta_{k+1}}\right) \dfrac{\|h(y^k + d^k)\|_2 - \|h(y^k)\|_2}{\|d^k\|_2^2} & \text{, se } d^k \neq 0 \\ \mu_k & \text{, se } d^k = 0 \end{cases} \tag{5.15}$$

e

$$\mu_k' = 1.01 \min\{\max\{\gamma, \mu_k''\}, 10^{10}\mu_k, 10^{40}\gamma\},$$

onde μ_k é o termo regularizador que acabou de ser usado na função objetivo do subproblema e d^k é a solução aproximada associada a ele. Com o valor de μ_k' calculado, se uma das condições (5.7) ou (5.8) não foi satisfeita para μ_k e d^k, aumentamos o valor de μ_k da seguinte forma:

$$\mu_k = \max\{\mu_k', 10\mu_k\} \tag{5.16}$$

e voltamos para o passo 3.1 para encontrar um novo d^k. Caso as duas condições sejam satisfeitas, guardamos μ_k' para a inicialização de μ_{k+1} na iteração seguinte.

Assim, nas iterações $k \geq 1$, antes da primeira resolução do subproblema (5.6), fazemos $\mu_k = \mu_{k-1}'$, onde μ_{k-1}' já foi calculado na iteração anterior. Em seguida, como descrito no parágrafo anterior, atualizamos μ_k de acordo com (5.15)–(5.16). Note que o fato de μ_{k-1}' não estar disponível na iteração $k = 0$ é o motivo pelo qual utilizamos a fórmula (5.14).

As escolhas dadas pelas equações (5.14)–(5.16) estão de acordo com os limitantes estabelecidos para μ no Algoritmo 5.1 e também garantem que μ cresça sempre que uma das condições (5.7) ou (5.8) não é satisfeita. Em particular, a equação (5.16) é a responsável por esse crescimento, enquanto a equação (5.15) é a responsável por incluir informações sobre as funções envolvidas. Todo o esquema de atualização pode parecer complexo, mas mostrou-se melhor do que a inicialização e atualização por valores fixos, como mostra a Figura 5.3. Na comparação, usamos o esquema dinâmico, discutido até o momento, e um esquema estático, no qual em toda iteração inicializamos μ com o valor mínimo γ e, caso necessário, aumentamos μ por um fator de 10.

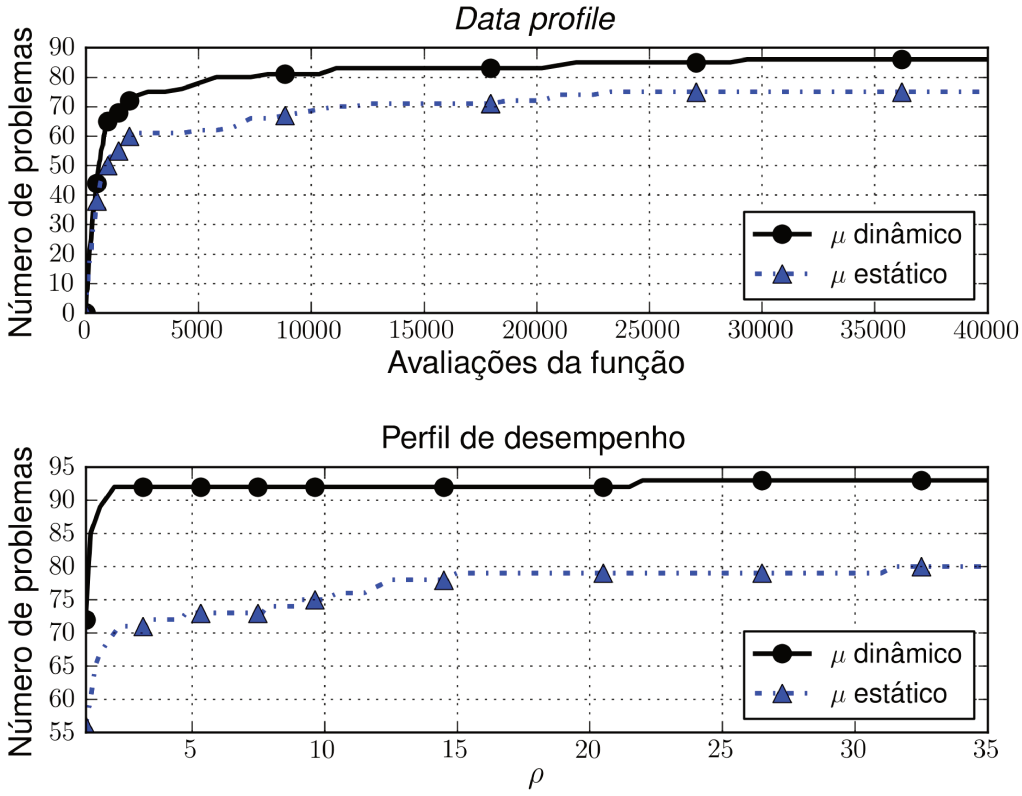


Figura 5.3: Comparação entre a atualização dinâmica de μ, baseada na desigualdade (5.13), e a estática.

Escolha do ponto inicial

O ponto inicial, passado a GSS para resolver o subproblema (5.6), também influencia no esforço computacional aplicado na fase de otimização. Já sabemos que $d = 0$ é uma escolha adequada, pois é um ponto viável de (5.6), está de acordo com a Hipótese R4 (decréscimo suficiente) e satisfaz as condições (5.7) e (5.8). Por isso, em toda iteração, utilizamos $d = 0$ como ponto inicial na primeira resolução de (5.6).

Se, após encontrar uma solução aproximada, digamos $\bar{d}$, do subproblema, a condição (5.7) ou a condição (5.8) não são satisfeitas, devemos resolvê-lo novamente, com um novo valor de μ definindo a nova função objetivo. Nesse caso, se

$$\mu_{\text{novo}} \leq \frac{f(y^k) - f(y^k + \bar{d})}{\|\bar{d}\|_2^2}, \tag{5.17}$$

então temos garantia de que a Hipótese R4 continuará a ser satisfeita, caso $\bar{d}$ seja fornecido como novo ponto inicial para GSS. Utilizando tal procedimento, temos a vantagem de que $\bar{d}$ é viável e, provavelmente, está próximo de ser solução aproximada do novo subproblema. Por fim, se o próprio $\bar{d}$ é declarado novamente solução aproximada,

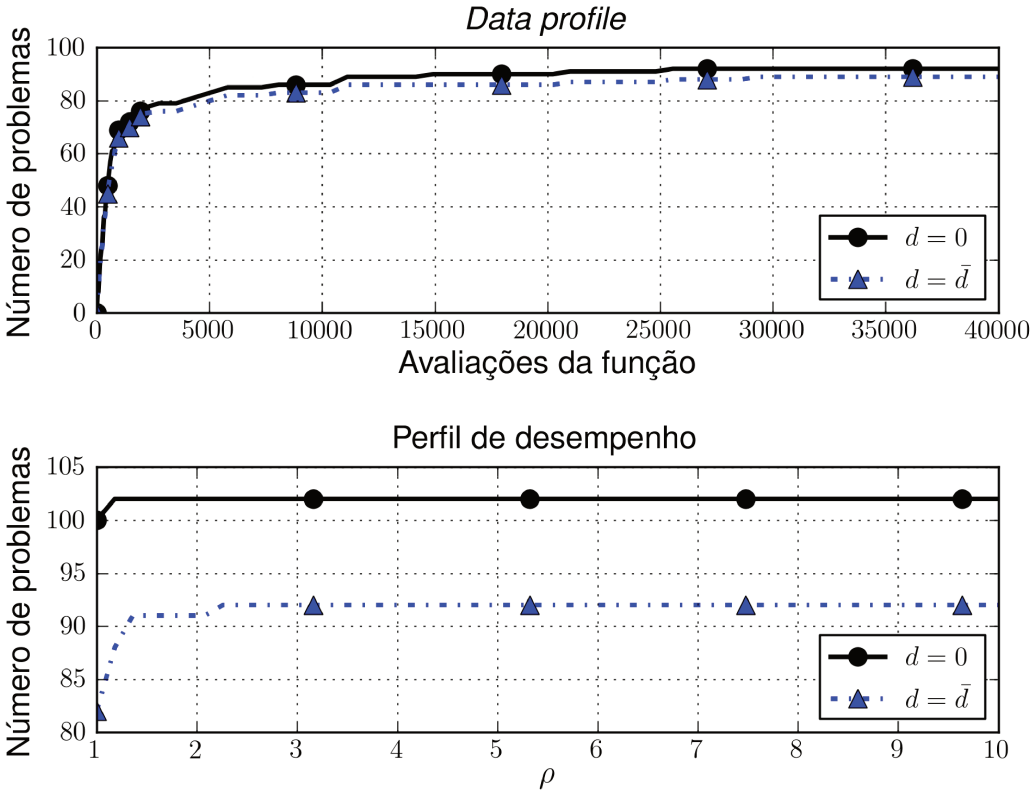


Figura 5.4: Comparação entre o uso da solução do subproblema anterior, denotada $\bar{d}$, e $d = 0$ como pontos iniciais para o subproblema (5.6).

pelas atualizações de μ definidas por (5.15)–(5.16) temos que as condições (5.7) e (5.8) são automaticamente satisfeitas.

Por outro lado, observamos que (5.17) é verificada poucas vezes na prática, de forma que se usamos $\bar{d}$ como ponto inicial podemos não satisfazer a Hipótese R4. Assim, comparamos a inicialização que usa $d = 0$ em todos os subproblemas com o reaproveitamento $d = \bar{d}$ discutido no parágrafo anterior. Podemos observar na Figura 5.4 que a escolha $d = 0$ é a mais eficiente e a mais robusta. Embora tenhamos limitado o *data profile* da figura, para mostrar que ambos resolvem aproximadamente o mesmo número de problemas gastando até 40000 avaliações da função, se um número ilimitado de avaliações é permitido, a escolha $d = 0$ resolve 11 problemas a mais do que a escolha $d = \bar{d}$. O fato de $d = 0$ ter sido a abordagem mais eficiente em aproximadamente 100 problemas indica que há diversos problemas fáceis que também são resolvidos de forma mais rápida.

5.2.4 Critérios de parada

De acordo com a teoria do Capítulo 4, sabemos que, se o algoritmo de Restauração Inexata proposto está próximo de uma solução, então temos que $\|h(x^k)\|_2$ e $\|d^k\|_2$ estão próximos de 0. Além disso, precisamos que $\alpha_{\min}^{(k)}$ se aproxime de 0 para garantirmos algum tipo de estacionariedade da solução. Por esse motivo, o Algoritmo 5.1 declara que houve sucesso na convergência se, após a fase de otimização, um ponto $y^k + d^k$ foi encontrado tal que

$$\|h(y^k + d^k)\|_2 \leq \varepsilon_{\text{viab}}, \quad y^k + d^k \in \mathcal{X}\,, \quad \|d^k\|_2 \leq \varepsilon_{\text{otim}} \quad \text{e} \quad \alpha_{\min}^{(k)} \leq \varepsilon_{\text{otim}}. \tag{5.18}$$

Associamos $\|d^k\|_2$ e $\alpha_{\min}^{(k)}$ ao parâmetro $\varepsilon_{\text{otim}}$, pelo fato de ambos estarem associados à solução vinda do subproblema da fase de otimização. De fato, se ambos estão próximos de zero, significa que o minimizador da aproximação do problema não está muito longe de ser viável, dado que y^k foi encontrado na fase de restauração.

5.3 Experimentos numéricos

O Algoritmo 5.1, com os passos descritos na Seção 5.2, foi implementado em linguagem C/C$_{++}$ como uma extensão do software HOPSPACK [Pla09]. Isso foi necessário porque HOPSPACK, escrito na linguagem C$_{++}$, implementa o algoritmo GSS, o qual é utilizado para resolver os subproblemas sem derivadas com restrições lineares (5.6). Na fase de restauração, utilizamos Algencan, implementado em Fortran 77, e a comunicação com a linguagem C/C$_{++}$ foi feita através de uma interface disponível com o próprio algoritmo [TAN]. Utilizamos nos testes um computador com processador Intel Core i7 2.67GHz, 8GB de memória RAM e sistema operacional Linux (Ubuntu). Para compilar, utilizamos `gfortran-4.2`, `g++-4.2` e a ferramenta `cmake`, necessária para compilarmos HOPSPACK. As bibliotecas BLAS e LAPACK (versão 3.2.1) também foram necessárias.

Comparamos o algoritmo proposto com dois outros algoritmos encontrados na literatura:

1. HOPSPACK [Pla09]: implementação em C$_{++}$ do algoritmo baseado em Lagrangianos Aumentados descrito em [KLT06b, GK10] e discutido na Seção 1.5;

2. NOMAD [LD11]: implementação em C$_{++}$ da classe de algoritmos Mads [ADJ06, AADJLD09], com diversas outras estratégias de otimização, como a barreira progressiva [ADJ09]. Mads foi discutido na Seção 1.2.

A escolha de HOPSPACK e NOMAD é devida ao fato de que são implementações conhecidas e bem desenvolvidas de algoritmos capazes de resolver problemas sem derivadas com restrições. Assim como o Algoritmo 5.1, ambas possuem teoria de convergência a pontos estacionários. Um comentário deve ser feito com respeito a NOMAD, cuja teoria não inclui restrições de igualdade.

Todos os algoritmos foram utilizados com seus parâmetros originais. O critério para considerar um ponto viável foi fixado em 10^{-8}, exceto para NOMAD, cuja modificação do parâmetro original piorou muito o funcionamento do algoritmo (de acordo com o manual, o valor padrão é 0). Como HOPSPACK caminha por pontos inviáveis, permitimos parâmetros de penalidade arbitrariamente altos, de forma que seja possível atingir a viabilidade sem modificarmos os critérios de parada por otimalidade. Para uma compatibilidade com os demais, modificamos o critério de parada por otimalidade de NOMAD para $\Delta_k^p \leq 10^{-3}$, onde Δ_k^p, como explicado na Seção 1.2, representa o grau de refinamento da malha. Para adequar o Algoritmo 5.1 aos critérios utilizados, definimos $\varepsilon_{\text{viab}} = 10^{-8}$ e $\varepsilon_{\text{otim}} = 10^{-3}$ na condição de parada (5.18) e em todos os momentos em que são necessários.

Para efeito das comparações, aplicamos os mesmos problemas descritos na Seção 5.1, assim como os critérios para considerar um problema como resolvido, dados por (5.1), e os gráficos de perfil de desempenho e *data profile*. Para a comparação do valor da função objetivo, presente na condição (5.1), utilizamos $\varepsilon_{\text{comp}} = 10^{-1}$, o que significa que não estamos interessados nas melhores soluções possíveis. O limite de 1 hora de tempo de CPU foi estabelecido para cada problema.

As tabelas 5.2 e 5.3 apresentam os resultados encontrados com o Algoritmo 5.1, HOPSPACK e NOMAD. Na primeira coluna, Prob. representa o número de cada problema, de acordo com [HS81], cujas características encontram-se na Tabela 5.1. Nas colunas seguintes temos que, para cada algoritmo, f representa o valor da função objetivo na solução encontrada, $\|h\|_2$ representa a inviabilidade da solução e #AF representa o número de avaliações da função objetivo utilizadas para encontrar a solução. Temos ainda que $^\bullet$ marca os problemas que não foram considerados resolvidos pelo respectivo algoritmo, ou seja, para os quais uma das condições de (5.1) não foi satisfeita. As entradas da tabela que possuem '$-$' representam os casos em que o tempo máximo de CPU foi utilizado e não obtivemos informação sobre o último ponto calculado.

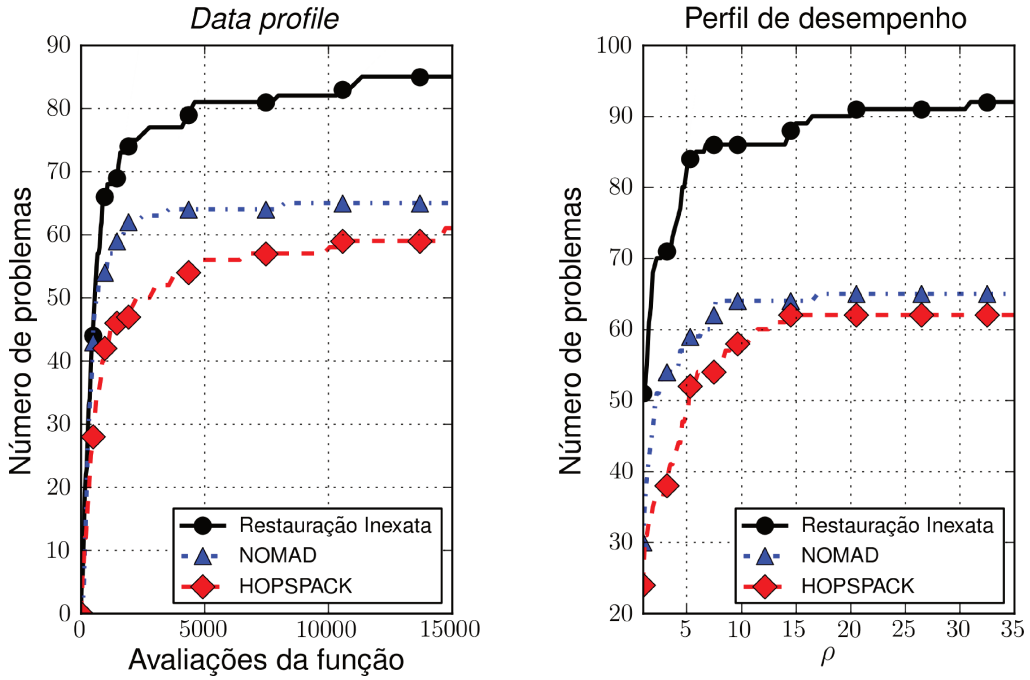
De acordo com as tabelas 5.2 e 5.3, temos que o Algoritmo 5.1 foi capaz de encontrar pontos viáveis em todos os problemas. Esse comportamento deve-se, em grande parte, ao bom desempenho de Algencan na fase de restauração. Em contrapartida, grande parte dos problemas que não foram resolvidos por HOPSPACK e NOMAD está associada à inviabilidade acima da tolerância. Para HOPSPACK, isso se deve à dificuldade intrínseca, que algoritmos baseados em funções de penalidade têm, quando o conjunto viável é dado por funções altamente não lineares. Para NOMAD, isso se deve principalmente às restrições de igualdade, por ser um algoritmo baseado em direções de busca. No quesito qualidade da solução, apenas em 10 problemas o Algoritmo 5.1 não encontrou o menor valor (de acordo com (5.1)) entre os algoritmos comparados. Observe, nos problemas 48–51, por exemplo, que o algoritmo proposto é claramente menos eficiente que HOPSPACK em problemas com apenas restrições lineares.

	Restauração Inexata			HOPSPACK			NOMAD		
Prob.	f	$\|h\|_2$	#AF	f	$\|h\|_2$	#AF	f	$\|h\|_2$	#AF
6	5.57E−10	4E−09	366	• 4.84E+00	5E−03	151	• 3.33E+00	7E−07	83
7	−1.73E+00	1E−09	153	• 6.93E−01	0E+00	325	• −1.72E+00	2E−07	73
8	−1.00E+00	2E−09	4	• −1.00E+00	9E−03	187	• −1.00E+00	1E−06	117
9	−5.00E−01	4E−15	117	−5.00E−01	4E−15	26	−5.00E−01	0E+00	186
10	−1.00E+00	7E−09	260	• −8.65E−01	0E+00	357	−1.00E+00	0E+00	205
11	−8.50E+00	8E−09	126	−8.48E+00	0E+00	523	−8.50E+00	0E+00	321
12	−3.00E+01	1E−09	587	−3.00E+01	0E+00	342	−3.00E+01	0E+00	117
13	1.09E+00	6E−09	52326	9.97E−01	5E−09	1183	1.00E+00	0E+00	110
14	1.39E+00	3E−09	20	• 1.39E+00	4E−06	202	1.39E+00	0E+00	151
15	3.06E+00	7E−14	82	3.06E+00	0E+00	451	3.06E+00	0E+00	154
16	2.58E−01	7E−10	546	2.50E−01	0E+00	600	• 3.98E+00	0E+00	208
17	1.00E+00	8E−10	118	1.00E+00	0E+00	612	1.00E+00	3E−17	132
18	5.00E+00	7E−11	51294	• 1.07E+01	0E+00	263	5.00E+00	0E+00	311
19	−6.96E+03	3E−09	138	• −7.47E+03	3E−01	1370	−6.96E+03	0E+00	225
20	4.02E+01	8E−10	88	3.82E+01	0E+00	393	3.82E+01	0E+00	190
21	−1.00E+02	7E−15	153	−1.00E+02	0E+00	32	−1.00E+02	0E+00	231
22	1.00E+00	9E−09	35	1.00E+00	0E+00	276	1.00E+00	0E+00	153
23	2.00E+00	7E−09	37	2.00E+00	0E+00	466	2.00E+00	0E+00	194
24	−1.00E+00	1E−09	132	−1.00E+00	4E−16	27	−1.00E+00	0E+00	119
26	1.58E−07	9E−09	11112	• 2.12E+01	0E+00	585	• 3.60E−01	0E+00	300
27	4.00E+00	7E−09	4135	• 4.00E+00	3E−06	1358	4.00E+00	0E+00	252
28	2.05E−23	9E−16	510	7.70E−08	3E−15	264	1.11E−06	0E+00	508
29	−2.26E+01	4E−09	540	−2.25E+01	0E+00	327	−2.26E+01	0E+00	302
30	1.00E+00	4E−09	790	1.00E+00	0E+00	55	1.00E+00	0E+00	79
31	6.00E+00	8E−10	526	6.00E+00	0E+00	921	6.00E+00	0E+00	277
32	1.00E+00	1E−09	86	1.00E+00	2E−16	51	• 1.99E+00	6E−17	599
33	• −4.00E+00	5E−09	54	−4.59E+00	0E+00	381	−4.59E+00	0E+00	408
34	−8.34E−01	2E−09	191	• −2.28E−01	0E+00	582	• −1.12E−01	0E+00	463
35	1.11E−01	7E−10	289	1.11E−01	0E+00	340	1.11E−01	0E+00	221
36	−3.30E+03	1E−12	273	−3.30E+03	0E+00	60	−3.30E+03	0E+00	457
37	−3.46E+03	1E−09	404	−3.46E+03	5E−14	102	−3.46E+03	0E+00	638
39	−1.00E+00	9E−10	435	−1.00E+00	0E+00	830	• −1.01E+00	6E−03	997
40	−2.50E−01	2E−09	127	• −2.51E−01	2E−03	897	• −2.50E−01	1E−06	147
41	1.93E+00	1E−09	430	1.93E+00	2E−15	292	1.93E+00	0E+00	600
42	1.39E+01	3E−09	382	1.40E+01	0E+00	779	1.40E+01	0E+00	276
43	−4.40E+01	2E−10	1485	−4.40E+01	0E+00	1134	−4.40E+01	0E+00	225
44	• −1.30E+01	2E−09	277	• −1.30E+01	0E+00	57	−1.50E+01	0E+00	249
46	1.42E−06	1E−09	1485	• 3.34E+00	2E−16	777	• 3.34E+00	2E−16	246
47	1.15E−08	9E−10	289	• 1.25E+01	0E+00	901	• 1.25E+01	0E+00	343
48	1.07E−24	2E−15	861	1.12E−06	2E−15	497	• 2.46E+00	0E+00	317
49	1.34E−07	2E−14	20308	1.43E−04	7E−15	1002	• 2.66E+02	0E+00	408
50	9.25E−27	6E−14	620	5.29E−07	6E−14	290	• 1.74E+04	0E+00	369
51	1.94E−27	9E−16	507	1.25E−06	9E−16	142	• 6.41E−01	0E+00	384
52	5.33E+00	3E−09	307	• 5.33E+00	8E−06	311	5.34E+00	0E+00	449
53	4.09E+00	3E−09	308	4.09E+00	1E−14	216	4.37E+00	0E+00	284
54	• −1.54E−01	5E−10	447	• -	-	-	−9.08E−01	0E+00	8074
55	6.67E+00	3E−09	18	• 6.00E+00	1E+00	0	6.71E+00	2E−16	301
56	• −2.23E−04	2E−09	14452	−1.00E+00	9E−16	2075	−1.00E+00	9E−16	452
57	3.06E−02	1E−09	287	3.06E−02	0E+00	74	2.85E−02	0E+00	193
58	3.19E+00	1E−09	102	3.19E+00	0E+00	817	3.19E+00	0E+00	197
59	−7.80E+00	4E−09	927	• −6.75E+00	0E+00	340	−7.80E+00	0E+00	335
60	3.26E−02	8E−10	596	• 5.47E−02	2E−03	465	• 4.53E+00	9E−07	139

Tabela 5.2: Resultados numéricos encontrados pelo algoritmo de Restauração Inexata sem derivadas (Algoritmo 5.1) e pelos algoritmos HOPSPACK e NOMAD em 105 problemas com restrições.

	Restauração Inexata			HOPSPACK			NOMAD		
Prob.	f	$\|h\|_2$	#AF	f	$\|h\|_2$	#AF	f	$\|h\|_2$	#AF
61	−1.44E+02	3E−09	182	−1.43E+02	0E+00	621	−1.43E+02	0E+00	203
62	−2.63E+04	2E−16	848	−2.63E+04	1E−16	233	−2.63E+04	2E−15	418
63	9.62E+02	4E−09	171	• 9.63E+02	5E−04	317	• 9.66E+02	6E−06	128
64	6.30E+03	3E−09	1076	6.30E+03	0E+00	6680	6.30E+03	0E+00	741
65	9.54E−01	6E−09	1859	1.01E+00	0E+00	379	9.54E−01	0E+00	322
66	5.18E−01	9E−09	382	5.33E−01	0E+00	566	5.32E−01	0E+00	511
68	−9.20E−01	9E−09	10424	• −8.44E−01	2E−04	1316	• −3.08E−01	4E−17	346
69	−9.57E+02	2E−09	4130	• −9.57E+02	1E−04	2471	• −8.46E+02	2E−17	486
70	• 2.69E−01	1E−09	5563	7.74E−03	0E+00	3766	7.50E−03	0E+00	1134
71	1.70E+01	8E−09	4379	• 1.70E+01	5E−07	1939	• 1.82E+01	6E−08	107
72	7.28E+02	3E−11	2714	7.28E+02	0E+00	18188	7.33E+02	1E−13	1604
73	2.99E+01	1E−09	211	3.02E+01	0E+00	223	3.00E+01	0E+00	464
74	5.13E+03	6E−10	392	• 5.14E+03	1E+00	46145	• 5.17E+03	8E−07	47347
75	5.17E+03	1E−09	139	• 5.23E+03	2E−01	22678	• 5.13E+03	3E−02	9883
76	−4.68E+00	6E−10	505	−4.68E+00	0E+00	403	−4.68E+00	0E+00	472
77	2.42E−01	2E−09	790	• 4.68E+00	2E−04	1904	• 2.40E+02	9E−08	358
78	−2.92E+00	8E−09	566	• −2.89E+00	4E−03	869	• −2.15E+00	1E−05	213
79	7.88E−02	6E−09	362	• 2.42E−01	1E−03	1054	• 8.86E+01	1E−06	203
80	5.39E−02	2E−09	658	• 1.00E+00	5E−03	557	• 2.09E−01	5E−06	210
81	5.39E−02	6E−09	770	• 1.00E+00	5E−03	557	• 3.32E−01	3E−06	196
83	−3.07E+04	2E−09	450	−3.07E+04	0E+00	1729	−3.07E+04	0E+00	1306
84	−5.28E+01	4E−10	338	−5.28E+01	0E+00	2842	−5.23E+01	0E+00	1734
86	−3.23E+01	3E−10	437	−3.23E+01	0E+00	485	−3.23E+01	0E+00	534
87	8.85E+03	1E−09	860	• 9.33E+03	7E−01	16244	9.14E+03	4E−09	498
88	1.36E+00	4E−09	271	1.36E+00	0E+00	1341	1.36E+00	0E+00	280
89	1.36E+00	8E−09	605	1.36E+00	0E+00	2208	1.36E+00	0E+00	423
90	1.38E+00	3E−10	833	1.36E+00	0E+00	2885	1.36E+00	0E+00	712
91	1.36E+00	8E−09	1561	1.36E+00	0E+00	4470	1.37E+00	0E+00	847
92	1.37E+00	5E−12	1052	1.36E+00	0E+00	4672	1.36E+00	0E+00	1068
93	1.35E+02	4E−09	766682	1.37E+02	0E+00	129	1.35E+02	0E+00	527
95	1.56E−02	2E−09	741	1.71E−02	0E+00	156	1.56E−02	0E+00	148
96	1.56E−02	5E−10	668	1.71E−02	0E+00	156	1.56E−02	0E+00	148
97	• 4.07E+00	5E−10	5638	• 4.12E+00	0E+00	145	3.14E+00	0E+00	395
98	3.14E+00	8E−10	1540	• 4.12E+00	0E+00	145	3.14E+00	0E+00	362
99	−8.31E+08	2E−09	4	• −7.46E+08	3E+03	729	• −8.16E+08	2E+00	311
100	• 7.94E+02	9E−09	8479059	6.84E+02	0E+00	873	6.81E+02	0E+00	1336
101	1.81E+03	1E−09	267	1.82E+03	0E+00	14614	• 2.54E+03	0E+00	1595
102	• 1.12E+03	2E−09	536617	9.20E+02	0E+00	15222	• 1.42E+03	0E+00	1165
103	• 1.55E+03	3E−12	22	5.44E+02	0E+00	14583	• 1.68E+03	0E+00	2047
104	3.95E+00	8E−10	20905	3.95E+00	0E+00	9844	4.08E+00	0E+00	681
105	1.14E+03	1E−09	25324	1.14E+03	0E+00	10580	1.14E+03	0E+00	1787
106	7.05E+03	9E−09	68430	• 1.14E+04	0E+00	30424	• 6.84E+03	3E−01	861
107	5.06E+03	4E−09	265	• 5.06E+03	3E−03	7232	• 5.23E+03	2E−05	795
108	• −5.00E−01	1E−09	139823	• −5.00E−01	0E+00	99	−8.44E−01	0E+00	2322
109	5.39E+03	1E−09	195471	• 5.50E+03	2E−02	57551	• 5.21E+03	1E+01	244247
111	−4.28E+01	2E−09	2470	• -	-	-	• −4.33E+01	4E−08	3255
112	−4.78E+01	3E−09	10907	−4.78E+01	8E−16	730	• −4.14E+01	3E−06	524
113	2.43E+01	4E−10	4469	2.54E+01	0E+00	1944	2.61E+01	0E+00	1062
114	−1.77E+03	2E−10	152970	• -	-	-	• −1.06E+03	2E−10	2233
116	9.76E+01	2E−09	46328	• 5.00E+01	8E−02	9131	• 9.41E+01	8E−03	2387
117	3.23E+01	5E−10	7951	• 5.40E+01	0E+00	7190	• 1.71E+02	0E+00	70018
118	6.65E+02	1E−09	1987	6.65E+02	2E−14	3762	6.83E+02	0E+00	3370
119	2.45E+02	2E−09	1356	2.45E+02	5E−15	944	• 7.91E+02	9E−06	1236

Tabela 5.3: Resultados numéricos encontrados pelo algoritmo de Restauração Inexata sem derivadas (Algoritmo 5.1) e pelos algoritmos HOPSPACK e NOMAD em 105 problemas com restrições.



(a) (b)

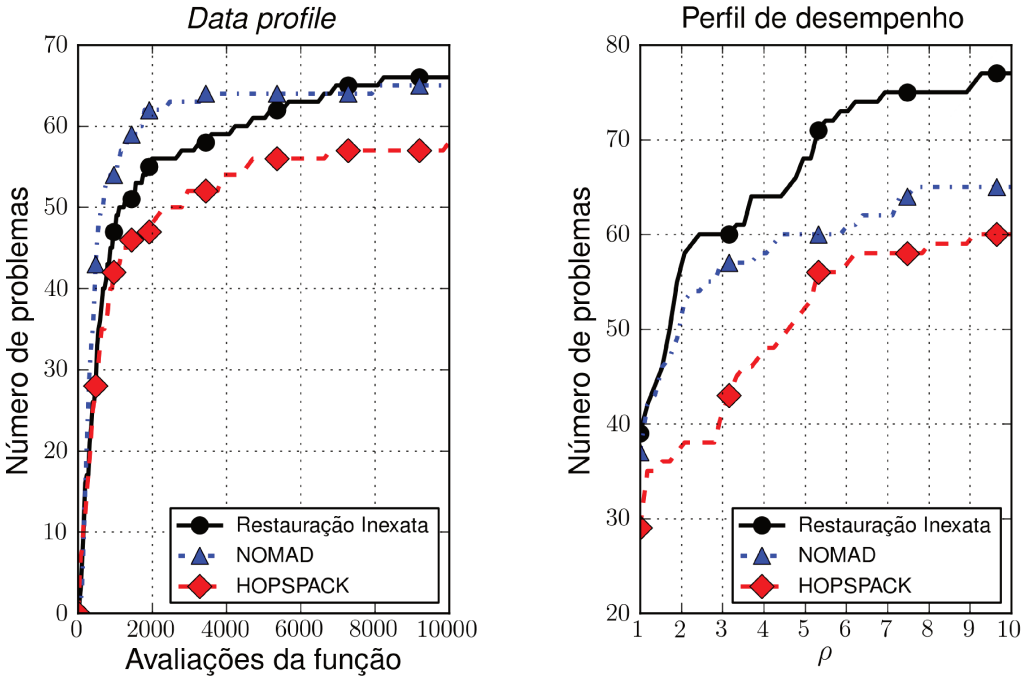
Figura 5.5: Comparação entre o algoritmo de Restauração Inexata sem derivadas (Algoritmo 5.1), NOMAD e HOPSPACK em 105 problemas com restrições.

Na Figura 5.5, apresentamos os resultados relativos ao número de avaliações da função gasto por cada algoritmo. O *data profile* dado pela Figura 5.5(a) mostra que, fixadas 5000 avaliações da função objetivo, por exemplo, o Algoritmo 5.1 resolve 15 problemas a mais do que NOMAD e 25 problemas a mais do que HOPSPACK. Essa diferença aumenta quando permitimos mais avaliações da função. No gráfico de perfil de desempenho, ilustrado na Figura 5.5(b), podemos verificar que o Algoritmo 5.1 é o mais eficiente e o mais robusto dos algoritmos comparados. É importante ressaltarmos que todos os bons resultados apresentados estão relacionados com o uso de algoritmos eficientes nas fases de restauração e otimização. Além disso, utilizamos uma informação adicional importante, ignorada pelos outros dois algoritmos: as derivadas das restrições.

Uma observação interessante pode ser feita com respeito à viabilidade encontrada por HOPSPACK e NOMAD, nos problemas para os quais a inviabilidade está dentro da tolerância de 10^{-8}. Podemos ver claramente que, nesses casos, a inviabilidade é consideravelmente menor do que a encontrada pelo Algoritmo 5.1. No caso de HOPSPACK, o fenômeno é explicado pelo tratamento explícito dado às restrições lineares. Conforme a Tabela 5.1, há diversos problemas compostos apenas por tais tipos de restrições. No caso de NOMAD, é possível que, embora pontos inviáveis sejam aceitos ao longo das iterações, devido à barreira progressiva [ADJ09], exista um crité-

rio de viabilidade mais rigoroso internamente. Embora a alta precisão na viabilidade não altere a comparação feita nas tabelas 5.2 e 5.3, pode influenciar nas comparações de eficiência mostradas na Figura 5.5. É possível que o esforço utilizado para atingir uma alta precisão na viabilidade seja refletido diretamente no número de avaliações da função. Consequentemente, o algoritmo torna-se menos eficiente.

Para verificarmos essa afirmação, modificamos o critério de viabilidade do Algoritmo 5.1 para $\varepsilon_{\text{viab}} = 10^{-16}$. Desta forma, somos capazes de aproximar-nos da precisão naturalmente atingida por HOPSPACK e NOMAD. Os resultados são mostrados na Figura 5.6. Não houve mudanças expressivas na quantidade de problemas resolvidos pelo Algoritmo 5.1 (robustez), por isso destacamos o que ocorre nos intervalos iniciais, que estão relacionados com a eficiência.



(a) (b)

Figura 5.6: Comparação entre o Algoritmo 5.1, com critério de viabilidade $\varepsilon_{\text{viab}} = 10^{-16}$, e os algoritmos NOMAD e HOPSPACK.

Como esperávamos, a comparação de eficiência mostrou-se mais justa. O gráfico de perfil de desempenho, Figura 5.6(b), indica que os três algoritmos resolveram aproximadamente o mesmo número de problemas nos quais foram os mais eficientes (não necessariamente os mesmos problemas). Na Figura 5.6(a), o *data profile* mostra que, com poucas avaliações da função objetivo (≤ 100), HOPSPACK é o algoritmo que resolve o maior número de problemas. Em seguida, NOMAD é aquele que resolve o maior número. Em praticamente todos os problemas que NOMAD foi capaz de resolver

foram gastas até 4000 avaliações da função. Por outro lado, existem aproximadamente 30 problemas nos quais o Algoritmo 5.1 gasta mais de 10000 avaliações. Quando um número arbitrário de avaliações é permitido, o Algoritmo 5.1 resolve 94 problemas, contra 65 resolvidos por NOMAD e 63 por HOPSPACK.

5.3.1 Aplicação

Embora os 105 problemas utilizados para os experimentos numéricos forneçam informações suficientes para o estudo do Algoritmo 5.1, também estamos interessados em problemas com maior apelo para a otimização sem derivadas. Nesse sentido, resolvemos o "Problema do Vestibular", uma variação do Problema do Volume apresentado em [Ped09, DEMP11], que possui ainda um apelo geométrico.

No problema do vestibular, são dados n_{H} hiperplanos e três tipos de pontos no espaço r-dimensional: n_{B} pontos do tipo B, n_{D} pontos do tipo D e n_{O} pontos do tipo O. O objetivo é formar um politopo com os hiperplanos tal que os pontos do tipo tipo B estejam em sua borda, os pontos do tipo D estejam *suficientemente dentro* e que contenha o menor número possível de pontos do tipo O.

A ideia é que um ponto representa r notas obtidas no vestibular para um determinado candidato. Os pontos do tipo B e D são alguns alunos do vestibular anterior que foram aprovados, escolhidos com base no seu desempenho na faculdade. Os pontos do tipo O representam todos os candidatos do vestibular (número relativamente grande). A intuição que acompanha este problema está relacionada com a separação por hiperplanos e é discutida em detalhes no Apêndice A.1.

A Tabela 5.4 apresenta informações sobre a dimensão das instâncias do problema do vestibular utilizadas nos testes. Na primeira coluna temos o nome da instância. As próximas 5 colunas foram explicadas no parágrafo anterior. Nas três últimas colunas, Var. representa o número de variáveis, Des. representa o número de restrições de desigualdade e Ig. representa o número de restrições de igualdade. Os valores entre parênteses indicam quantas das restrições são lineares. Maiores detalhes de cada instância são fornecidos no Apêndice A.1.

A função objetivo é dada pela contagem de pontos do tipo O que estão dentro do politopo. Podemos ver que ela não possui derivadas e seu custo está associado a n_{O}, que pode ser grande dependendo da instância escolhida. Os valores de n_{B}, n_{D} e n_{H} são pequenos e estão associados às restrições. Desta forma, temos que o custo computacional do problema está totalmente ligado às avaliações da função objetivo. Todas as restrições são suaves e possuem as derivadas disponíveis. Apenas na instância SVM3 os dados são realmente baseados no vestibular, para as disciplinas de matemática e português ($r = 2$), encontrados em [CON02]. Nas instâncias SVM2-4 e SVM2-4' apenas mudamos o parâmetro ε_{D} de 0.5 para 2, respectivamente, que indica o quão *suficientemente dentro* do politopo devem estar os pontos do tipo D.

Resolvemos as instâncias com o Algoritmo 5.1 e HOPSPACK. Foram feitas 10 tentativas para cada instância, sendo que todos os pontos iniciais foram os mesmos para os dois algoritmos. Na Tabela 5.5 apresentamos os resultados encontrados. Os

Instância	r	n_{H}	n_{B}	n_{D}	n_{O}	Var.	Des.	Ig.
SVM1-2	2	2	3	1	3	6	12 (8)	3
SVM1-3	2	3	3	1	3	9	18 (12)	3
SVM2-3	2	3	3	1	10^5	9	18 (12)	3
SVM2-4	2	4	3	1	10^5	12	24 (16)	3
SVM2-4'	2	4	3	1	10^5	12	24 (16)	3
SVM3-3	2	3	3	9	15000	9	66 (36)	3
SVM3-4	2	4	3	9	15000	12	88 (48)	3
SVM3-5	2	5	3	9	15000	15	110(60)	3
SVM4-4	2	4	4	4	1000	12	52 (32)	4

Tabela 5.4: Instâncias do problema do vestibular. Os valores entre parênteses representam o número de restrições que são lineares.

valores em negrito representam as soluções com menor valor funcional.

	Restauração Inexata			HOPSPACK		
Instância	f	Viab.	#AF	f	Viab.	#AF
SVM1-2	2	8.2E−10	48	**1**	4.4E−13	2175
SVM1-3	**2**	4.0E−10	262	**2**	0.0	1811
SVM2-3	**29295**	6.1E−10	32	**29295**	1.1E−22	4797
SVM2-4	29186	7.2E−09	1237	**28889**	1.8E−10	3651
SVM2-4'	**47518**	1.5E−09	1226	50119	9.8E−09	4564
SVM3-3	1843	4.2E−11	66	**1709**	9.6E−15	3504
SVM3-4	1821	4.7E−10	292	**1709**	9.6E−15	3504
SVM3-5	2041	5.6E−10	477	**1691**	1.1E−14	5591
SVM4-4	**116**	7.1E−10	191	**116**	7.1E−10	3803

Tabela 5.5: Resultados encontrados para o problema do vestibular.

Os resultados apresentados na Tabela 5.5 indicam que o algoritmo de Restauração Inexata utiliza poucas avaliações da função e não atinge os melhores resultados. Ao observarmos seu funcionamento detalhadamente, verificamos que a fase de otimização não consegue encontrar $d \neq 0$ que reduza o valor da função objetivo. Consequentemente, todos os resultados são oriundos apenas da fase de restauração. Tal dificuldade pode ser causada pelo grande número de restrições lineares de desigualdade que são convertidas em restrições de igualdade, aumentando o número total de variáveis. Sabemos que problemas com mais de 100 variáveis não são considerados de pequeno porte em otimização sem derivadas. Apenas na instância SVM2-4' o algoritmo proposto encontrou um menor valor funcional do que o encontrado por HOPSPACK.

Nas figuras 5.7, 5.8, 5.9, 5.10, 5.11 e 5.12, mostramos as soluções encontradas pelo Algoritmo 5.1 e a melhor solução encontrada, no caso em que esta não foi atingida pelo mesmo. Em todos os desenhos, as setas apontam para o interior do poliedro. Os quadrados (azuis) estão associados aos pontos B, que devem estar na borda do poliedro.

Os pontos maiores (verdes) são os pontos do tipo D, que devem estar suficientemente dentro do poliedro. Por fim, os pontos menores estão associados aos pontos do tipo O, os quais queremos que o menor número possível esteja no interior do poliedro formado pelos hiperplanos.

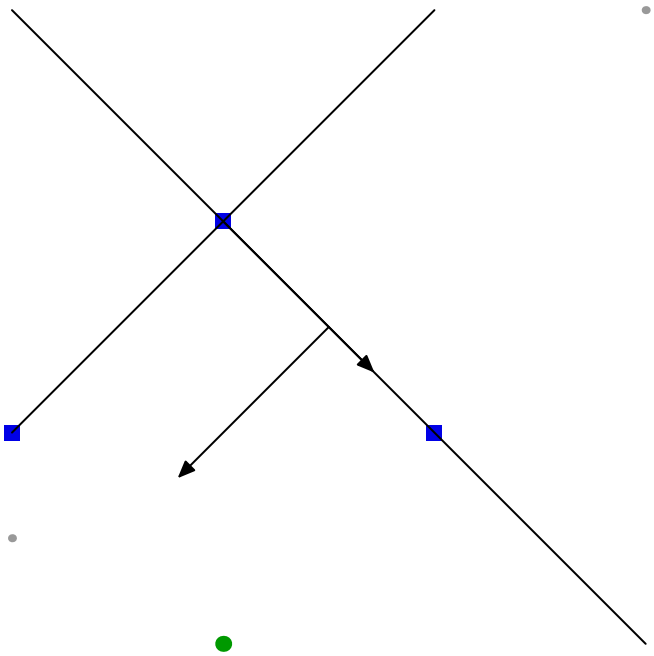


Figura 5.7: Instância SVM1-2, solução encontrada pelo Algoritmo 5.1. Os quadrados representam os pontos B, os pontos circulares grandes representam os pontos D e os menores representam os pontos O. As setas apontam para o interior do poliedro.

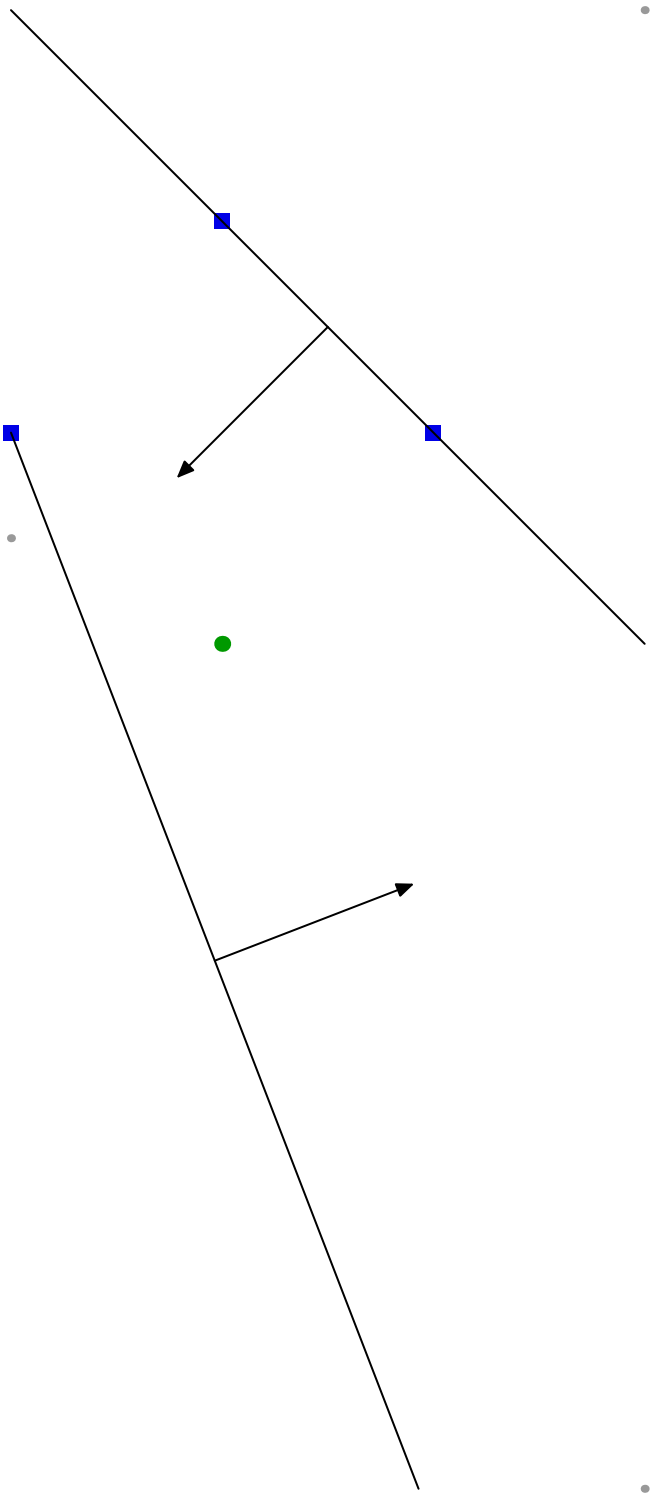


Figura 5.8: Instância SVM1-2, solução ótima encontrada por HOPSPACK. Os quadrados representam os pontos B, os pontos circulares grandes representam os pontos D e os menores representam os pontos O. As setas apontam para o interior do poliedro.

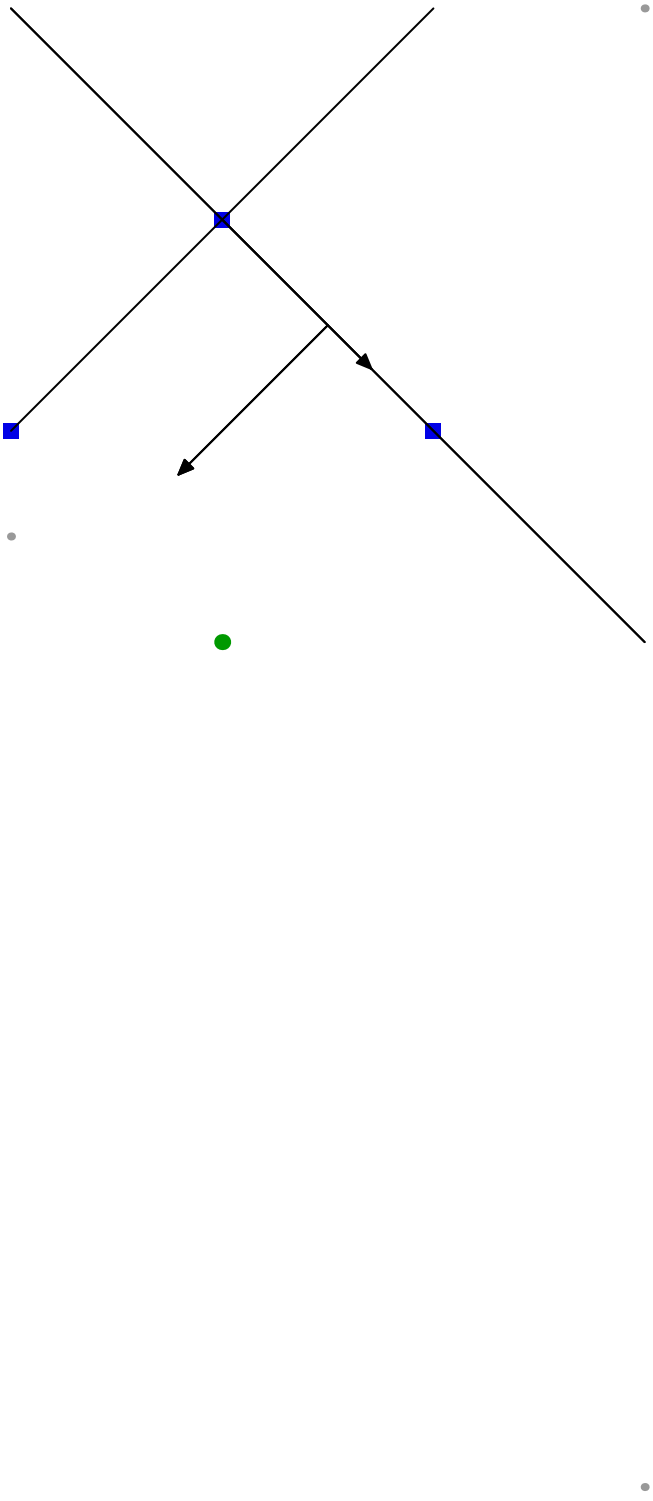


Figura 5.9: Instância SVM1-3, solução (não ótima) encontrada pelo Algoritmo 5.1. A solução ótima tem o aspecto apresentado na Figura 3.10.

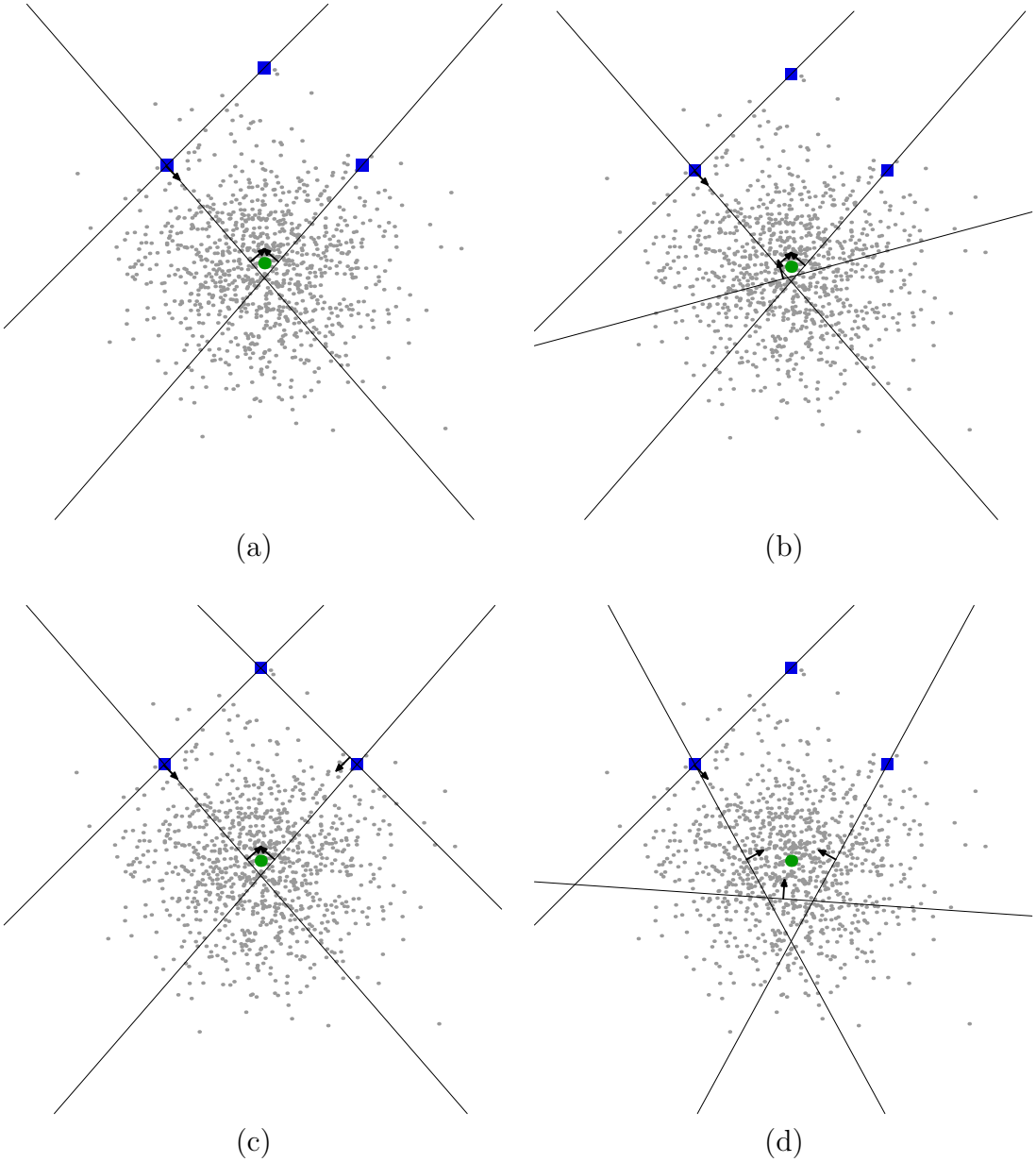


Figura 5.10: Instâncias (a) SVM2-3 com o Algoritmo 5.1, (b) SVM2-4 com o Algoritmo 5.1, (c) SVM2-4 com HOPSPACK (melhor) e (d) SVM2-4' com o Algoritmo 5.1.

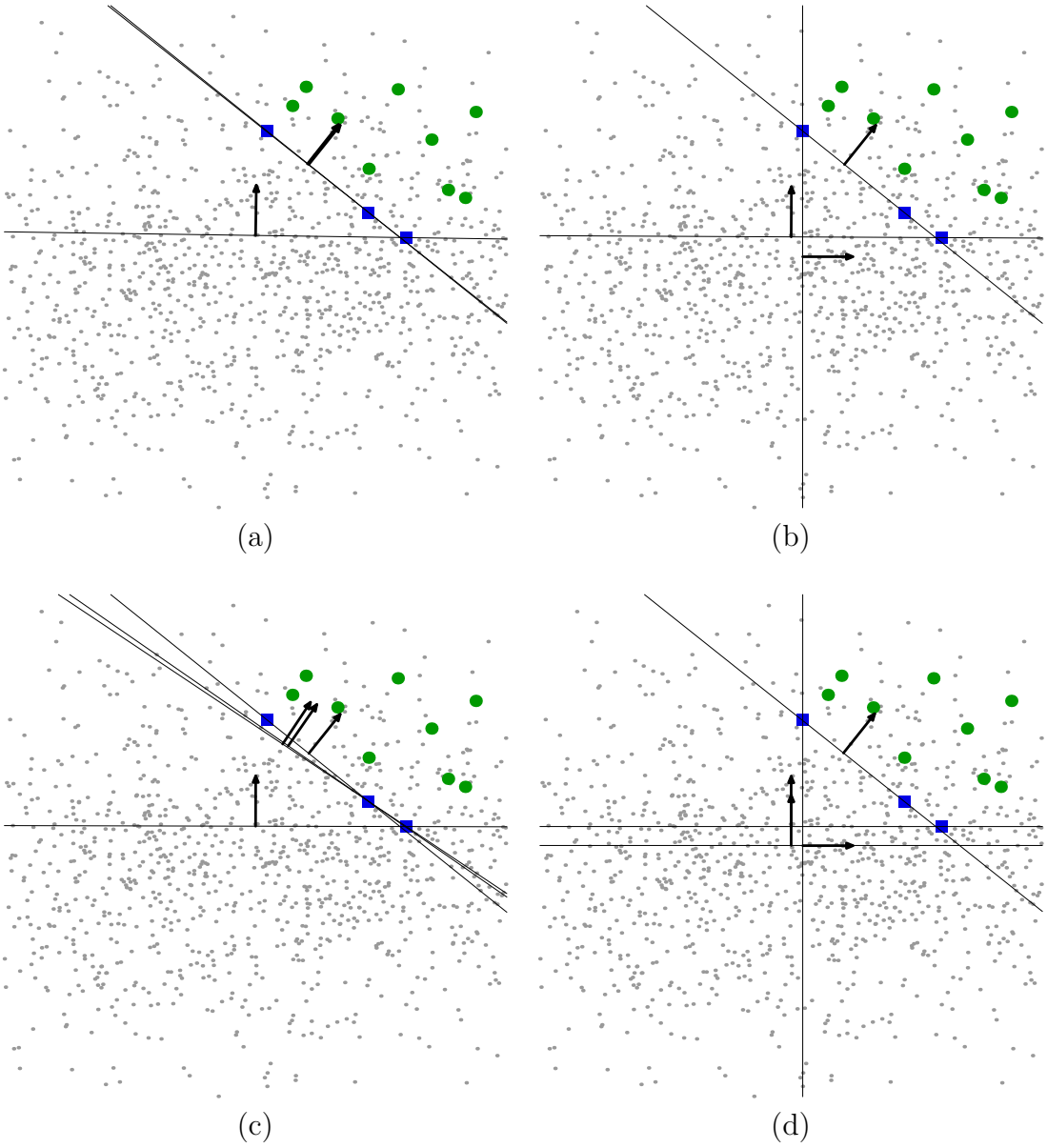


Figura 5.11: Instâncias (a) SVM3-3 com o Algoritmo 5.1, (b) SVM3-3 com HOPSPACK (melhor), (c) SVM3-4 com o Algoritmo 5.1 e (d) SVM3-4 com HOPSPACK (melhor). As coordenadas x estão associadas às notas de matemática e y às notas de português. Os eixos não foram desenhados.

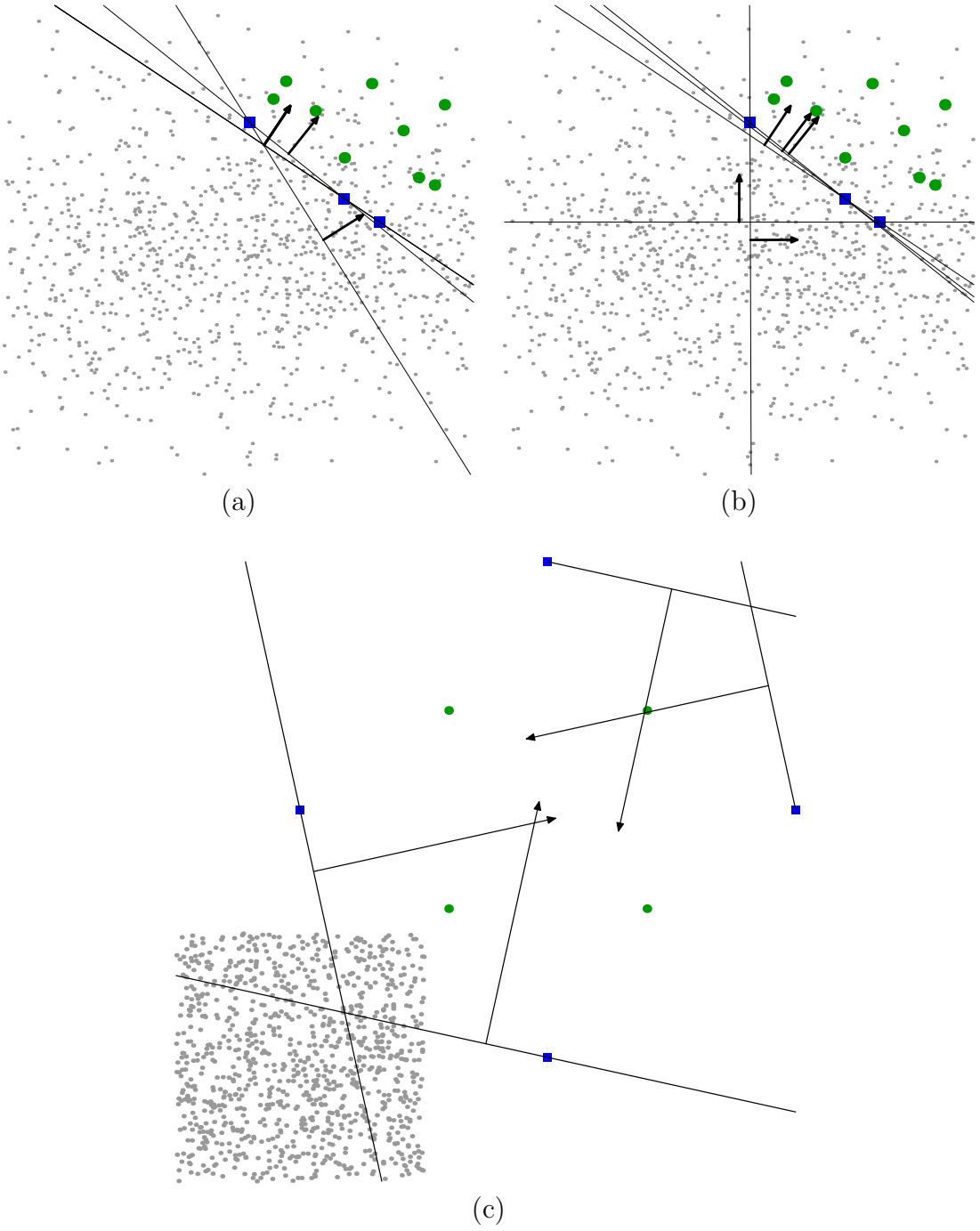


Figura 5.12: Instâncias (a) SVM3-5 com o Algoritmo 5.1, (b) SVM3-5 com HOPSPACK (melhor) e (c) SVM4-4 com o Algoritmo 5.1.

Capítulo 6

Conclusões e trabalhos futuros

Neste trabalho, apresentamos dois algoritmos para problemas sem derivadas com restrições. Ambos têm em comum a habilidade de tratar eficientemente os casos nos quais encontrar um ponto viável exige operações complexas, mas o custo computacional de tais operações é irrelevante ao ser comparado com o custo de uma avaliação da função objetivo.

O princípio de funcionamento dos algoritmos baseou-se fundamentalmente nas ideias de Restauração Inexata [Mar98, MP00, Mar01, FF10], que divide o processo de resolução em duas fases: restauração e otimização. Adaptamos esse princípio para o caso sem derivadas e determinamos que a função objetivo não deve ser usada quando a ocasião é de aprimorar a viabilidade. Desta forma, os capítulos 2 e 3 foram dedicados ao algoritmo denominado *Skinny*, que utiliza a Restauração Inexata em associação com uma estratégia descrita por Paviani e Himmelblau [PH69, Him72] e com a busca direta direcional em conjuntos densos [ADJ06, VC10]. Os capítulos 4 e 5 descreveram a teoria e implementação de uma adaptação para o caso sem derivadas do algoritmo baseado em Restauração Inexata descrito em [FF10].

A principal motivação para o desenvolvimento de *Skinny*, nos capítulos 2 e 3, foi a incapacidade dos métodos baseados em busca direta direcional em trabalhar com conjuntos viáveis magros, em particular com restrições de igualdade.

Primeiramente, na fase de restauração, utilizamos algoritmos eficientes para aprimorar a viabilidade do ponto corrente. É importante usar um algoritmo que se adapte às restrições do problema e evitar que a função objetivo seja avaliada. Mostramos que, no caso em que as restrições têm derivadas, podemos usar algoritmos clássicos, como Algencan [ABMS07, TAN], e, se as derivadas não estão disponíveis, podemos aplicar métodos sem derivadas para um problema de viabilidade [ESV11].

Na fase de minimização, trabalhamos com a função objetivo original sujeita a um relaxamento do conjunto viável, que transforma um conjunto magro em um conjunto denominado gordo. Essa abordagem permitiu que algoritmos eficientes e bem estabelecidos para problemas sem derivadas em conjuntos gordos fossem aplicados. Com a utilização de um conjunto denso de direções de consulta, foi possível demonstrar que o algoritmo proposto, sob hipóteses razoáveis, possui convergência a minimizadores

globais do problema original. Consideramos que a teoria utilizada é mais simples e os resultados obtidos são mais fortes do que os encontrados em [ADJ06, VC10].

Pelo fato de ser especialmente desenvolvido para conjuntos magros, comparamos *Skinny* com algoritmos capazes de lidar, de alguma forma, com tais conjuntos: HOPSPACK [Pla09], baseado em Lagrangianos Aumentados, NOMAD [LD11], baseado em busca direta e conjuntos densos, e DFO [CST97b], baseado em regiões de confiança. Observamos que HOPSPACK foi prejudicado por tratar simultaneamente a viabilidade e a otimalidade. NOMAD, por sua vez, não foi capaz de encontrar com facilidade direções viáveis de consulta em conjuntos magros. Surpreendentemente, DFO resolveu grande parte dos problemas, embora sua utilização em problemas sem derivadas com restrições seja uma estratégia heurística [CST98]. No conjunto de testes, DFO foi o mais eficiente, mas *Skinny* foi o algoritmo mais robusto.

Por não usar função de mérito, mostramos que *Skinny* também foi capaz de resolver problemas que apresentam o fenômeno da voracidade [CMMS10], enquanto HOPSPACK, devido à função de penalidade, encontrou sérias dificuldades. Por fim, mostramos o bom desempenho do algoritmo proposto em diversas instâncias do Problema do Vestibular (descrito no Anexo A.1), de forte apelo geométrico.

Nos capítulos 4 e 5, discutimos a teoria e implementação de um algoritmo baseado em Restauração Inexata para problemas sem derivadas com restrições suaves. O algoritmo proposto seguiu as ideias apresentadas em [FF10]. Todos os resultados de convergência encontrados em [FF10] foram recuperados para o novo algoritmo.

Dado que as restrições, neste caso, têm derivadas, na fase de restauração podemos utilizar algoritmos clássicos de programação não linear para resolver, como mostrado, um problema de projeção ortogonal no conjunto viável. Novamente, foi essencial evitarmos as avaliações da função objetivo nesta fase.

Na fase de otimização, um subproblema formado pela função objetivo original sujeita ao espaço tangente das restrições no ponto restaurado foi resolvido. Qualquer algoritmo pode ser usado para resolver o subproblema, mas utilizamos o algoritmo GSS [KLT06b], por possuir importantes propriedades teóricas. Além disso, GSS foi implementado de uma forma eficiente e essa eficiência foi em grande parte herdada pelo algoritmo de Restauração Inexata. Sob a hipótese de que a avaliação função objetivo é a parte mais custosa do problema, implementar a fase de otimização de forma adequada foi essencial para o bom desempenho do algoritmo. No capítulo 5 discutimos todos os aspectos dessa fase e sugerimos estratégias que aparentaram ser as mais adequadas.

Comparamos uma implementação do algoritmo de Restauração Inexata com os algoritmos HOPSPACK e NOMAD, pois estes são capazes de trabalhar com restrições e têm o suporte de uma teoria de convergência. O algoritmo de Restauração Inexata foi o mais eficiente e o mais robusto em um conjunto com 105 problemas com restrições suaves. Esse resultado ressalta a importância de utilizarmos todas as informações disponíveis para resolver um problema. No presente caso, a informação utilizada, ignorada pelos outros algoritmos, eram as derivadas das restrições.

O mesmo algoritmo encontrou dificuldades para resolver o Problema do Vestibular. Acreditamos que o fato das restrições lineares de desigualdade não serem trata-

das explicitamente foi um dos fatores responsáveis pela baixa qualidade dos resultados.

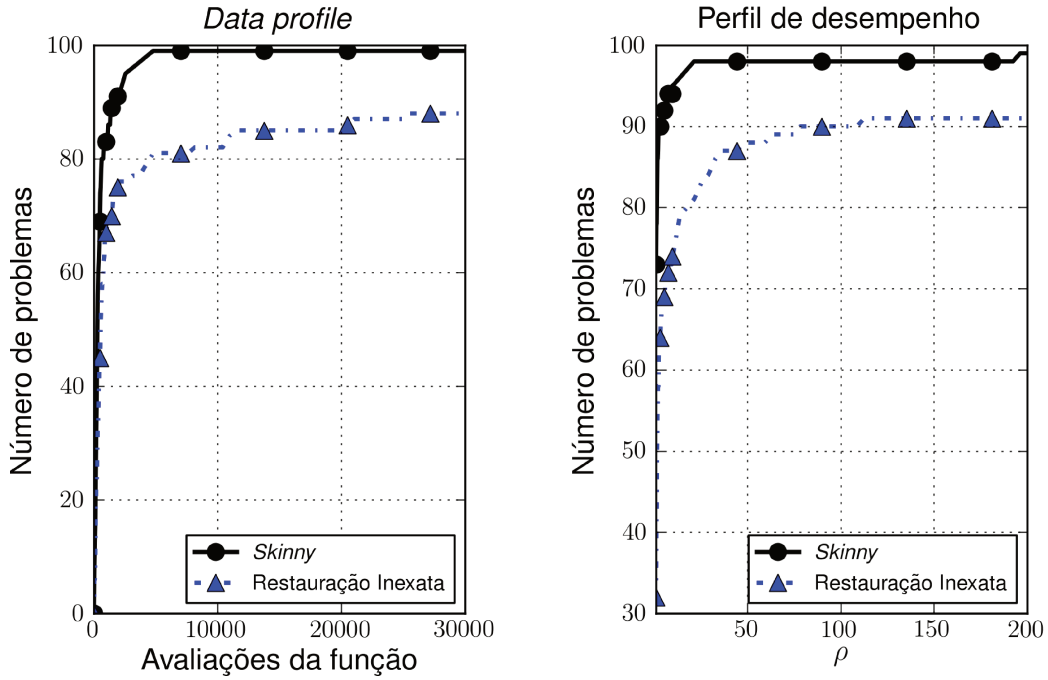
Os dois algoritmos apresentados neste trabalho utilizaram, de alguma forma, as derivadas das restrições. Embora pareça injusto realizarmos a comparação com outros algoritmos que não as utilizam, os bons resultados numéricos mostraram a importância de uma estratégia baseada em duas fases (restauração e otimização). A conclusão mais importante que podemos chegar é que deve-se sempre utilizar o máximo de informações sobre o problema a ser resolvido. Nesse aspecto, os dois algoritmos permitem uma liberdade quase total na escolha dos métodos que serão utilizados tanto na fase de restauração quanto na fase de otimização. Caso a generalidade seja escolhida, o algoritmo está bem definido, mas outros talvez sejam preferíveis. Porém, caso deseje-se utilizar informações específicas do problema, toda a estrutura do algoritmo aceita tal informação, enquanto nos demais algoritmos isso não é possível.

Por fim, vale destacar que cada um dos algoritmos tem uma finalidade específica. De um lado temos o algoritmo de Restauração Inexata sem derivadas, apoiado sobre uma forte teoria de convergência, como os algoritmos de programação não linear. Seu desenvolvimento foi uma extensão para o caso sem derivadas e a implementação foi uma maneira de verificarmos suas aplicações na prática. Do outro lado temos *Skinny*, cuja teoria, baseada em direções densas, já indica que seu desenvolvimento está mais relacionado com a intuição, dado que na prática nunca será possível construir um conjunto denso de pontos. *Skinny* foi desenvolvido para resolver problemas e parte da premissa que utilizaremos tudo o que estiver ao alcance para atingir esse objetivo. De fato, sua descrição em alto nível, dada pelo Algoritmo 2.2, permite que praticamente qualquer estratégia seja utilizada em cada passo. Por esse motivo, como pode ser observado ao longo deste trabalho, é de se esperar que *Skinny* seja mais eficiente que o algoritmo de Restauração Inexata.

Embora *Skinny* tenha sido desenvolvido para lidar especialmente com conjuntos magros, também pode ser aplicado para resolver problemas com conjuntos viáveis gordos. Por esse motivo, utilizamos na comparação os 105 problemas de Hock e Schittkowski, cujas restrições possuem derivadas disponíveis, necessárias para o algoritmo de Restauração Inexata. No *data profile* apresentado na Figura 6.1(a) podemos observar que para todos os problemas que *Skinny* resolveu, foram utilizadas menos do que 10000 avaliações da função objetivo. Por outro lado, há problemas nos quais o algoritmo de Restauração Inexata utiliza mais de 20000 avaliações. Na Figura 6.1(b) o gráfico de perfil de desempenho indica que *Skinny* é realmente o mais eficiente e o mais robusto.

6.1 Sugestões e trabalhos futuros

A área de otimização sem derivadas com restrições é relativamente nova e, portanto, pode ser melhor desenvolvida tanto na teoria quanto na prática. Durante o desenvolvimento do presente trabalho, nos deparamos com diversas possibilidades e temas de estudo, que não puderam ser desenvolvidas devido ao risco de desviarmos de nosso objetivo principal.



(a) (b)

Figura 6.1: Comparação entre o *Skinny* e o algoritmo de Restauração Inexata sem derivadas (Algoritmo 5.1).

Os excelentes resultados apresentados por DFO [CST97b, CST98], utilizando Algencan para encontrar pontos viáveis, trouxe à tona a discussão sobre métodos baseados em região de confiança para problemas sem derivadas. De acordo com a Seção 1.6, sabemos que até o presente momento não há teoria de convergência desses métodos sequer para restrições de caixa. Desconhecemos o motivo, porém observamos que, assim como os resultados teóricos apresentados para Lagrangianos Aumentados sem derivadas [LT02, LT10] e Restauração Inexata (no Capítulo 4), a teoria de regiões de confiança sem derivadas estende os resultados clássicos existentes [NW99] aliados a modelos gerados por interpolação [CSV09b, ST10, Pow11]. Um estudo mais aprofundado de tais métodos e os problemas que são capazes de resolver é o primeiro passo a ser dado. Além disso, percebemos que a restauração influencia positivamente nos resultados. Todo o estudo desses algoritmos está apenas no começo, pois resultados recentes [FMN09] indicaram que hipóteses consideradas essenciais para a convergência não se mostraram necessárias na prática. Logo, existe a possibilidade da teoria ser mais simples.

Embora saibamos que DFO aplicado a problemas restritos seja uma heurística, um tópico interessante é a sua utilização na resolução dos subproblemas sem derivadas com restrições lineares, existentes no algoritmo de Restauração Inexata pro-

posto no Capítulo 4. Não há garantia de que os pontos encontrados por DFO satisfarão a Hipótese R5 (que especifica o significado de "resolução aproximada"), mas, dados os resultados observados na prática, acreditamos que boas soluções podem ser encontradas.

No âmbito computacional, algumas possibilidades foram mencionadas ao longo do trabalho. Em [ESV11] foram apresentados resultados referentes a algoritmos sem derivadas para a resolução de sistemas não lineares. Sabemos que a fase de restauração de *Skinny* pode ser modelada como um problema de viabilidade, definido apenas por um sistema de restrições de igualdade e desigualdade. Logo, no caso em que as derivadas das restrições não estão disponíveis, acreditamos que aplicar o algoritmo proposto em [ESV11] produzirá resultados melhores do que resolver o problema irrestrito de minimização da inviabilidade (3.4), como é feito em *Skinny* atualmente.

No algoritmo de Restauração Inexata, o tratamento explícito das restrições lineares, principalmente de desigualdade, também pode trazer benefícios. Sabemos que o algoritmo Algencan possui uma versão que considera restrições lineares [And08] e o algoritmo GSS naturalmente é capaz de considerá-las. O resultado direto de tal modificação é a redução no número de variáveis de folga que serão adicionadas ao problema.

Uma possibilidade de disponibilização do algoritmo de Restauração Inexata diz respeito ao software HOPSPACK, pois este foi elaborado em C_{++} de forma que possa facilmente ser estendido e novas rotinas possam ser incorporadas. A inclusão do algoritmo de Restauração Inexata sem derivadas como uma classe de HOPSPACK pode ser um estratégia interessante de divulgação.

Apêndice A

Descrição dos problemas utilizados nos testes

Um modo prático para testar o funcionamento de um algoritmo que não utiliza derivadas é aplicá-lo na resolução de um problema que possui derivadas e optar por não utilizá-las. Porém, a justificativa mais forte para utilizar tal algoritmo, em detrimento de um outro algoritmo que se baseia em derivadas, é resolver problemas nos quais a derivada não existe ou é muito difícil de ser calculada. Em um problema no qual a função objetivo é o resultado de uma simulação, experimento ou resolução de um modelo, por exemplo, é praticamente impossível calcularmos o gradiente e a Hessiana.

É claro que o desempenho de um algoritmo que utiliza derivadas, aplicado em um problema no qual elas existam e foram calculadas, será bastante superior ao de um algoritmo que não utiliza tal informação. Por esse motivo, é interessante testarmos o algoritmo em problemas que não podem ser resolvidos de outra maneira senão olhando apenas para a função objetivo, embora testes numéricos com problemas conhecidos da literatura (que possuem derivadas ou não) também sejam de grande valia.

Há quem afirme que, se temos um algoritmo baseado em derivadas e desejamos resolver um problema no qual elas não estejam disponíveis, basta aplicar a técnica das diferenças finitas para fazer as aproximações de primeira e/ou segunda ordem e prosseguir normalmente com o algoritmo. De fato isso pode ser feito e bons resultados podem ser obtidos. Nossa justificativa para não realizar tal procedimento é simples: se a função objetivo calculada em um ponto for o resultado de uma simulação que demore 1 minuto, então calcular o seu gradiente no ponto pode demorar até $2n$ minutos, se forem utilizadas as diferenças centrais. No mesmo tempo, um algoritmo que usa apenas a informação da função objetivo já avaliou a função em $2n$ pontos.

Encontrar problemas que satisfaçam os requisitos acima mencionados e que ainda sejam palpáveis, no sentido de que possamos visualizar sua solução, são um desafio à parte. Em [Ped09], motivados pelas mesmas razões, problemas com forte apelo geométrico são utilizados para testar um algoritmo baseado em Lagrangianos Aumentados sem derivadas.

Descrevemos a seguir os conjuntos de problemas de otimização utilizados

nos experimentos numéricos do presente trabalho. Todos os problemas enquadram-se nas características essenciais que tornam os algoritmos propostos neste trabalho os mais adequados. Essencialmente, todos os problemas possuem restrições suaves que formam, em sua maioria, um domínio magro. Também incluímos problemas que apresentam o fenômeno da voracidade [CMMS10].

A.1 O problema do vestibular

Entre os problemas testados para o algoritmo proposto em [Ped09], está o "Problema da Área". Nesse problema, dado um conjunto fixo de pontos no espaço bidimensional, o objetivo é encontrar a figura de menor área que contenha tais pontos. O autor poderia, em princípio, considerar figuras simples, tais como quadrados, círculos ou elipses, mas essas figuras possuem fórmulas fechadas e a função objetivo associada à sua área provavelmente possui derivadas. Por conseguinte, as figuras consideradas foram formadas pela **intersecção** de figuras simples: o retângulo, o círculo e a elipse. Pode-se verificar que encontrar a área da figura formada pela intersecção de uma elipse e um círculo não é uma tarefa fácil.

Por causa da dificuldade em encontrar fórmulas fechadas para tais figuras, foi utilizado um método que é um caso particular do método de Monte Carlo para o cálculo de integrais múltiplas [DM87]. Para isso, inscrevemos a figura desejada em um retângulo cuja área é conhecida e geramos pontos aleatórios dentro desse retângulo. A estimação da área será dada pela proporção de pontos que estão dentro da figura.

Na Figura A.1 temos um exemplo da aplicação desse método para encontrar a área da intersecção entre uma elipse e um círculo. Foram gerados 40 pontos aleatórios dentro de um retângulo de área 7.5 e apenas 7 deles ficaram dentro da intersecção. Assim, para estimar a área da figura, nos baseamos na proporção de pontos que ficaram dentro dela e na área total do retângulo, ou seja $7.5 \cdot \frac{7}{40} = 0.75$. Obviamente, quanto mais pontos são usados na estimação, mais próximo da verdadeira área será o valor encontrado.

Posteriormente, em [DEMP11] esse problema foi generalizado para um número arbitrário de dimensões. Sabemos que esse problema é um problema sem derivadas, mas desejamos adaptá-lo ao caso restrito, com restrições fáceis, para aplicar os algoritmos desenvolvidos neste trabalho. Com esse objetivo, apresentamos uma extensão do problema da área, denominada Problema do Vestibular, cuja intuição explicamos a seguir.

Um professor, após ministrar um curso de Cálculo para uma turma na universidade, classificou n_{B} de seus alunos como medianos e n_{D} como excelentes. Esses alunos, por sua vez, para entrar na universidade, concorreram com centenas de milhares de outros candidatos na prova do vestibular, que contém questões de todas as áreas do conhecimento. Dessa forma, um candidato, que foi ou não aprovado no vestibular, pode

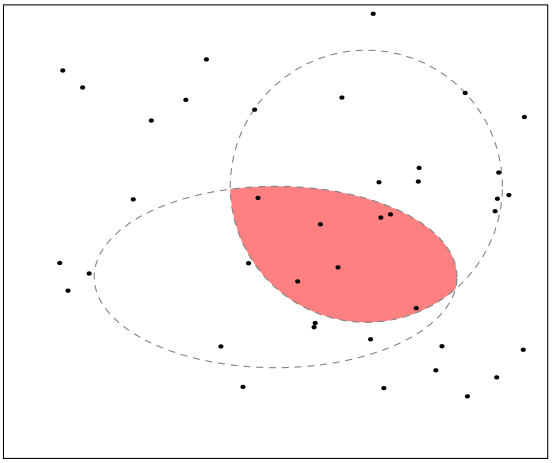


Figura A.1: Usando o método de Monte Carlo com 40 pontos, a área estimada da intersecção entre a elipse e o círculo é 0.75.

ser representado pela r-upla

$$p = (p_1, p_2, \ldots, p_r) \in \mathbb{R}^r$$

representando as notas obtidas em cada uma das r áreas do conhecimento avaliadas pelo vestibular.

Dados n_{O} candidatos que realizaram a prova do vestibular, representados por pontos no espaço r dimensional e denominados pontos do tipo O, queremos encontrar a posição de n_{H} hiperplanos tais que:

(i) os n_{B} alunos selecionados como medianos estejam na borda do politopo formado pelos hiperplanos, ou seja, o ponto associado ao determinado aluno esteja sobre algum dos hiperplanos e "dentro" dos demais (pontos do tipo B);

(ii) os n_{D} alunos selecionados como excelentes estejam *suficientemente dentro* do politopo formado pelos hiperplanos (pontos do tipo D).

O objetivo é minimizar o número de candidatos que prestaram o vestibular (pontos do tipo O) e que se encontram dentro do politopo gerado pela intersecção dos n_{H} hiperplanos.

Como podemos verificar, a complexidade da função objetivo está ligada ao número de candidatos que prestaram o vestibular, número este que pode ser considerado grande (centenas de milhares de candidatos). Logo, não desejamos avaliar a função objetivo quando não for estritamente necessário, ou seja, apenas em pontos viáveis, o que justifica a utilização da função penalidade extrema já mencionada anteriormente. Também é fácil verificar que esta função objetivo não possui derivadas, pois é dada pela contagem de pontos dentro do politopo.

Também podemos supor que a complexidade das restrições do problema é pequena, afinal está associada com os requisitos (i) e (ii) acima descritos e é intuitivo

pensarmos que o número de alunos de um curso de Cálculo (dezenas ou centenas de alunos) é bem inferior ao número de candidatos que prestaram o vestibular para entrar na universidade. Os requisitos do tipo (i) podem ser modelados através de restrições não lineares de igualdade e restrições lineares de desigualdade da seguinte forma

$$(a_1^T p_j - b_1)(a_2^T p_j - b_2) \cdots (a_{n_\mathrm{H}}^T p_j - b_{n_\mathrm{H}}) = 0, \quad j = 1, \ldots, n_\mathrm{B} \tag{A.1}$$

$$a_i^T p_j - b_i \leq 0 \quad i = 1, \ldots, n_\mathrm{H} \quad j = 1, \ldots, n_\mathrm{B}, \tag{A.2}$$

onde $p_j \in \mathbb{R}^r$ representa um aluno e o hiperplano i, $i = 1, \ldots, n_\mathrm{H}$, é representado pelo vetor normal $a_i \in \mathbb{R}^n$ e pelo escalar b_i. Essas restrições são responsáveis pelo conjunto viável não ser um conjunto gordo.

Os requisitos do tipo (ii) são modelados através de restrições lineares e não lineares de desigualdade da seguinte forma:

$$a_i^T p_j - b_i \leq 0 \quad i = 1, \ldots, n_\mathrm{H} \quad j = 1, \ldots, n_\mathrm{D} \tag{A.3}$$

$$\varepsilon_\mathrm{D}^2 \|a_i\|_2^2 \leq (b_i - a_i^T p_j)^2 \quad i = 1, \ldots, n_\mathrm{H} \quad j = 1, \ldots, n_\mathrm{D}, \tag{A.4}$$

onde $\varepsilon_\mathrm{D} > 0$ representa o que queremos dizer com *suficientemente dentro* do politopo. A restrição (A.3) é necessária para que a restrição (A.4) seja continuamente diferenciável. A restrição (A.4) surge da ideia de que a distância de um ponto ao hiperplano pode ser facilmente medida utilizando-se o a direção normalizada dada pelo vetor diretor do hiperplano.

Para evitar que a solução trivial $(a_i, b_i) = (\mathbf{0}, 0)$, $i = 1, \ldots, n_\mathrm{H}$, seja escolhida, adicionamos a seguinte restrição:

$$\|a_j\|_2^2 \geq 1 \quad j = 1, \ldots, n_\mathrm{H}. \tag{A.5}$$

Dessa forma, todas as restrições são continuamente diferenciáveis.

Este problema pode ser relacionado com um conjunto de problemas conhecido como Máquinas de Suporte Vetorial (do inglês SVMs ou *Support Vector Machines*) [SS02]. Em sua forma mais simples, há dois tipos de objetos, representados por $+1$ e -1. São utilizados, então, conjuntos de dados cuja classificação é conhecida para "treinar" a máquina. Depois de treinada, os dados novos são fornecidos e ela classifica-os automaticamente, baseando-se nos dados fornecidos na fase de treino. Pode-se pensar que existe uma função desconhecida de classificação dos objetos, e o treinamento é uma tentativa de aproximar tal função. Uma aplicação bastante conhecida das máquinas de suporte vetorial é a classificação de *e-mails* como *spam* ou não *spam* na caixa de entrada do computador.

Seguindo esse espírito, uma vez resolvido o problema do vestibular, ou seja, depois de encontradas as posições dos hiperplanos, utilizando, para isso, os candidatos a entrar na universidade e os alunos escolhidos pelo professor de Cálculo, podemos utilizá-los para descobrir possíveis bons alunos de Cálculo nos ingressantes de vestibulares que acontecerem posteriormente. Para isso, basta verificarmos quais os pontos (que representam as notas dos alunos no vestibular) estão dentro do politopo previamente

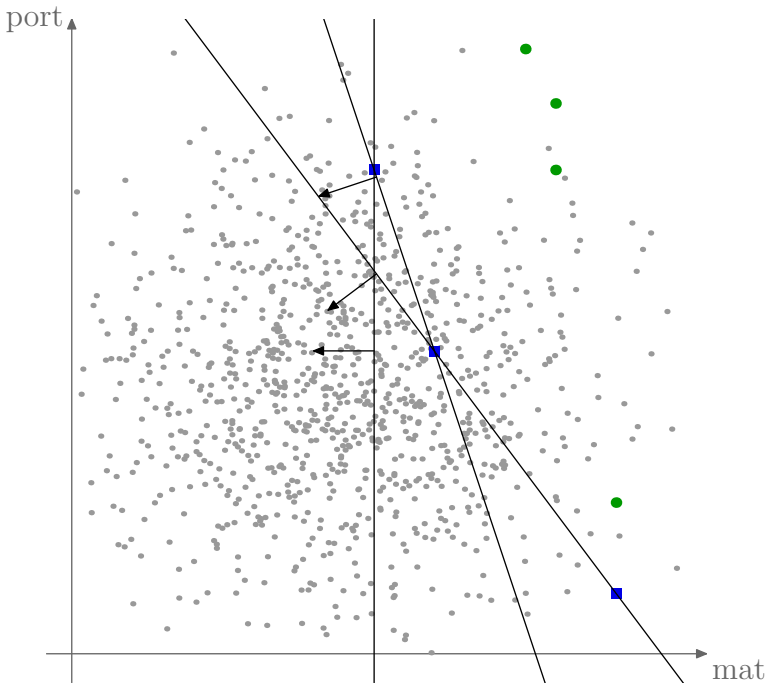


Figura A.2: O problema do vestibular. Neste exemplo, as áreas do conhecimento avaliadas pelo vestibular são apenas as de matemática (eixo x) e português (eixo y). Os alunos classificados pelo professor de Cálculo como medianos são os quadrados (azuis) e os alunos classificados como excelentes são os pontos (verdes).

encontrado. Em outras palavras, com o politopo criado, podemos classificar os candidatos como *possivelmente* bons ou *possivelmente* maus alunos de Cálculo. Tal solução do problema também pode servir de base para esquemas alternativos de avaliação dos alunos no vestibular.

A Figura A.2 mostra a solução de uma instância do problema do vestibular. Três alunos foram classificados como medianos e quatro como excelentes pelo professor de Cálculo e o número de hiperplanos com os quais deseja-se construir o politopo é 3. Todos os pontos representam as notas de matemática e português que os candidatos (incluindo os alunos de Cálculo) obtiveram no vestibular, contabilizando um total de 1000 candidatos.

Olhando este problema apenas como um problema de otimização matemática, as restrições (A.1)–(A.5) são suficientes para modelá-lo adequadamente. De fato, se apenas essas restrições forem utilizadas, a solução da instância descrita no parágrafo anterior está representada na Figura A.3. Porém, se observarmos a Figura A.3, a intuição nos diz que, se um aluno representado pelas notas $(6, 7)$ ficou dentro do politopo, ou seja, tem a possibilidade de ser um bom aluno de Cálculo de acordo com nossas previsões, um aluno representado por $(6, 10)$ também deveria ter essa possibilidade, dado que suas notas são maiores ou iguais a um aluno que ficou dentro do politopo. Pela solução dada pela Figura A.3, o aluno com nota $(6, 10)$ pode ficar fora do politopo.

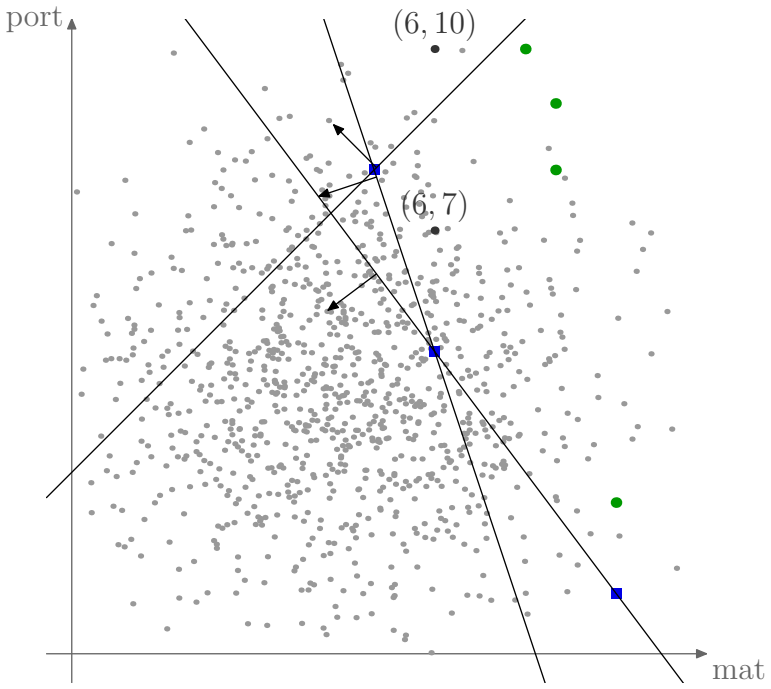


Figura A.3: Sem a restrição de negatividade (A.6) alunos com notas maiores ficam de fora do politopo. Note que o hiperplano não toca os ponto grandes (verdes) por que estes devem estar suficientemente dentro do politopo.

Para tornar o problema do vestibular mais condizente com a realidade e para que seja uma solução também no aspecto "intuitivo", adicionamos a seguinte restrição:

$$a_j \leq 0 \quad j = 1, \ldots, n_{\mathrm{H}}. \tag{A.6}$$

Com isso, o problema do vestibular é modelado como se segue:

$$\begin{array}{ll} \min & \text{O número de pontos do tipo O dentro do politopo} \\ \text{s. a} & \text{(A.1), (A.2), (A.3), (A.4), (A.5) e (A.6).} \end{array} \tag{A.7}$$

Dadas todas as informações necessárias, temos que o problema do vestibular dado por (A.7) tem $n_{\mathrm{H}} + 1$ variáveis, $n_{\mathrm{H}}(n_{\mathrm{B}} + n_{\mathrm{D}})$ restrições lineares de desigualdade, $n_{\mathrm{H}}(n_{\mathrm{D}} + 1)$ restrições não lineares de desigualdade, n_{B} restrições não lineares de igualdade e n_{H} restrições de caixa. A função objetivo não possui derivadas e as restrições são simples e têm derivadas disponíveis.

Todas as características que definem este problema podem ser aproveitadas pelos algoritmos descritos nos capítulos 2 e 4. A existência de restrições de igualdade o torna inadequado para os algoritmos baseados em busca direcional e justifica o processo de engordamento das restrições usado no Capítulo 2. Se considerarmos que n_{H}, n_{B} e n_{D} são números pequenos, o problema possui poucas restrições e que são continuamente diferenciáveis, o que nos permite aplicar qualquer tipo de método para realizar a fase de restauração do algoritmo.

Por fim, ainda supondo n_{B} e n_{D} pequenos quando comparados a n_{O}, temos que toda complexidade computacional dos problemas deste tipo está na avaliação da função objetivo. Por esse motivo, o custo computacional da avaliação das restrições, quando comparado com a função objetivo, é insignificante. Além disso, o fato da avaliação da função objetivo ser custosa, justifica a utilização da função penalidade extrema e de técnicas de restauração, pois não queremos avaliar a função objetivo em pontos inviáveis, que não oferecerão informação para um algoritmo de busca direcional.

A.1.1 Instâncias

Para os testes foram geradas 9 instâncias do problema do vestibular. Seus dados são apresentados na Tabela A.1. A nomenclatura das instâncias segue o seguinte padrão:

SVM<número_instância>-<n_{H}>.

De todas as instâncias criadas, apenas SVM3-3, SVM3-4 e SVM3-5 foram baseadas em dados reais de vestibular e necessitam da restrição (A.6) para que encontremos o resultado desejado. As demais instâncias foram pensadas apenas como problemas de separação de pontos, como a Figura A.3. Com exceção de SVM2-4', em todas as instâncias escolhemos $\varepsilon_{\mathrm{D}} = 0.5$ como o parâmetro de *suficientemente dentro*.

Na Tabela A.1, a primeira coluna identifica a instância. Nas 5 colunas subsequentes, r representa a dimensão do problema, n_{H} representa o número de hiperplanos, n_{B} representa o número de pontos do tipo B, n_{D} representa o número de pontos do tipo B, n_{O} representa o número de pontos do tipo O. Nas três últimas colunas, temos que Var. representa o número total de variáveis do problema, Des. representa o número de restrições de desigualdade e Ig. representa o número de restrições de igualdade. Os valores entre parênteses representam quantas das restrições são lineares.

Instância	r	n_{H}	n_{B}	n_{D}	n_{O}	Var.	Des.	Ig.
SVM1-2	2	2	3	1	3	6	12 (8)	3
SVM1-3	2	3	3	1	3	9	18 (12)	3
SVM2-3	2	3	3	1	10^5	9	18 (12)	3
SVM2-4	2	4	3	1	10^5	12	24 (16)	3
SVM2-4'	2	4	3	1	10^5	12	24 (16)	3
SVM3-3	2	3	3	9	15000	9	66 (36)	3
SVM3-4	2	4	3	9	15000	12	88 (48)	3
SVM3-5	2	5	3	9	15000	15	110(60)	3
SVM4-4	2	4	4	4	1000	12	52 (32)	4

Tabela A.1: Instâncias do problema do vestibular. Os valores entre parênteses representam o número de restrições que são lineares.

As instâncias SVM1-2 e SVM1-3 possuem apenas 3 pontos do tipo O, de forma que é fácil sabermos qual a solução ótima: 1 e 0 respectivamente.

As instâncias SVM2-3, SVM2-4 e SVM2-4' foram criadas para verificar o funcionamento do termo ε_{D}. Em SVM2-4' alteramos este valor de 0.5 para 2. Foram gerados 10^5 pontos aleatórios do tipo O ($n_{\mathrm{O}} = 10^5$), que seguem uma distribuição normal com média 0 e variância 3. São necessários ao menos 2 hiperplanos para obter-se uma solução viável.

A instância SVM4-4 também foi desenvolvida para verificar o comportamento dos algoritmos com os pontos do tipo D. Nesse caso, os pontos do tipo D estão em mesmo número que os do tipo B. Pela localização dos pontos dos tipos B e D, não é possível resolver a instância com menos de 4 hiperplanos. Os 1000 pontos do tipo O foram gerados aleatoriamente de acordo com uma distribuição uniforme no intervalo $[-1.5; -0.5] \times [-1.5; -0.5]$.

As instâncias SVM3-3, SVM3-4 e SVM3-5 foram baseadas em dados do vestibular da UNICAMP, obtidos em [CON02]. Os pontos do tipo B e D foram definidos como as notas de matemática (x) e português (y) dos melhores alunos do vestibular entre 1987 e 2002. Algumas modificações tiveram que ser feitas nas notas para que um resultado razoável fosse obtido. Para gerar 15000 pontos do tipo O, representando os alunos que prestaram o vestibular em 2002, procedemos da seguinte maneira:

1. Escolhemos uma questão da segunda fase de matemática e uma da segunda fase de português associadas com as maiores variâncias.

2. As notas são dadas entre 0 e 5, então foi necessário escalá-las. A distribuição normal permite que escalemos e mesmo assim tenhamos a distribuição normal dos novos valores.

3. Obtivemos as seguintes distribuições, com as quais foram gerados 15000 valores aleatórios:
$$\text{matemática} \sim N(4,94; (4,58)^2)$$
$$\text{português} \sim N(4,62; (1,92)^2),$$
onde a distribuição normal é dada pela média e pela variância (quadrado do desvio padrão).

A.2 Problemas de W. Hock & K. Schittkowski

A coleção de testes para algoritmos de programação não linear de Hock e Schittkowski [HS81, Sch87] é formada por mais de 200 problemas com diversas características, que variam desde problemas simples com poucas variáveis e restrições de caixa a problemas descontínuos e com função objetivo e restrições derivadas de problemas práticos de otimização. Em [HS81] são apresentados 116 problemas e os demais são apresentados em [Sch87].

É uma prática comum na otimização sem derivadas testar o funcionamento de novos métodos em problemas dessa coleção. Para que seja possível uma comparação entre os resultados encontrados pelos métodos propostos neste trabalho, também

resolvemos os problemas da coleção, em particular, os primeiros 116 de [HS81]. Dentre os 116 primeiros problemas de teste, é importante que o foco principal seja dado nos problemas que se encaixam nas características daqueles que nos propusemos a resolver. Não estamos interessados, por exemplo, em resolver problemas apenas com restrições de caixa. O motivo é que tais problemas não vão qualificar o desempenho do nosso algoritmo e sim o desempenho do método que resolverá o subproblema, que nesse caso também contém apenas caixas. Além disso, como o conjunto viável é simples, os métodos de Lagrangianos Aumentados não encontram nenhuma dificuldade em resolvê-los, sendo métodos preferíveis ao nosso. Por outro lado, problemas com restrições de igualdades (magras), com muitas restrições e/ou restrições complexas são extremamente interessantes para o algoritmo proposto. Assim, dos 116 problemas escolhemos 105, descartando aqueles somente com restrições de caixa (9 problemas) e aqueles cuja derivada das restrições não existe ou está errada (2 problemas), que não nos interessam porque utilizamos as derivadas das restrições para encontrar pontos viáveis no conjunto engordado.

Na Tabela A.2 descrevemos as características dos 105 problemas selecionados para os testes. Nas primeiras quatro colunas, Prob. representa o número do problema, de acordo com o descrito em [HS81], Var. representa o número de variáveis, Des. representa o número de restrições de desigualdade e Ig. representa o número de restrições de igualdade. Os valores entre parênteses indicam quantas das restrições são lineares. Nos outros dois conjuntos de quatro colunas o significado é o mesmo.

A edição de [HS81] já está esgotada, mas ainda é possível encontrar a descrição detalhada de cada um dos 116 problemas, incluindo a solução, o ponto inicial e o valor da função objetivo na solução. O manuscrito oficial, porém, não descreve dois problemas que estão na coleção: 46 e 58. Abaixo apresentamos tais problemas.

- **Problema 46**

  - 5 variáveis e 2 restrições de igualdade

$$\begin{aligned} \min \quad & (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6 \\ \text{s. a} \quad & x_1^2 x_4 + \operatorname{sen}(x_4 - x_5) - 1 = 0 \\ & x_2 + x_3^4 x_4^2 - 2 = 0 \end{aligned}$$

  - $x_{\text{ini}} = (0.5\sqrt{2}, 1.75, 0.5, 2, 2)$
  - $x^* = (1, 1, 1, 1, 1)$
  - $f(x^*) = 0$

- **Problema 58**

  - 2 variáveis e 3 restrições de desigualdade

Prob.	Var.	Des.	Ig.	Prob.	Var.	Des.	Ig.	Prob.	Var.	Des.	Ig.
6	2	0	1	43	4	3	0	80	5	0	3
7	2	0	1	44	4	6(6)	0	81	5	0	3
8	2	0	2	46	5	0	2	83	5	6	0
9	2	0	1(1)	47	5	0	3	84	5	6	0
10	2	1	0	48	5	0	2(2)	86	5	10(10)	0
11	2	1	0	49	5	0	2(2)	87	6	0	4
12	2	1	0	50	5	0	3(3)	88	2	1	0
13	2	1	0	51	5	0	3(3)	89	3	1	0
14	2	1	1(1)	52	5	0	3(3)	90	4	1	0
15	2	2	0	53	5	0	3(3)	91	5	1	0
16	2	2	0	54	6	0	1(1)	92	6	1	0
17	2	2	0	55	6	0	6(6)	93	6	2	0
18	2	2	0	56	7	0	4	95	6	4	0
19	2	2	0	57	2	1	0	96	6	4	0
20	2	3	0	58	2	3	0	97	6	4	0
21	2	1(1)	0	59	2	3	0	98	6	4	0
22	2	2(1)	0	60	3	0	1	99	7	0	2
23	2	5(1)	0	61	3	0	2	100	7	4	0
24	2	3(3)	0	62	3	0	1(1)	101	7	6	0
26	3	0	1	63	3	0	2(1)	102	7	6	0
27	3	0	1	64	3	1	0	103	7	6	0
28	3	0	1(1)	65	3	1	0	104	8	6	0
29	3	1	0	66	3	2	0	105	8	1 (1)	0
30	3	1	0	68	4	0	2	106	8	6 (3)	0
31	3	1	0	69	4	0	2	107	9	0	6
32	3	1	1(1)	70	4	1	0	108	9	13	0
33	3	2	0	71	4	1	1	109	9	4 (2)	6
34	3	2	0	72	4	2	0	111	10	0	3
35	3	1(1)	0	73	4	2(1)	1(1)	112	10	0	3(3)
36	3	1(1)	0	74	4	2(2)	3	113	10	8 (3)	0
37	3	2(2)	0	75	4	2(2)	3	114	10	8 (4)	3(1)
39	4	0	2	76	4	3(3)	0	116	13	15 (5)	0
40	4	0	3	77	5	0	2	117	15	5	0
41	4	0	1(1)	78	5	0	3	118	15	29(29)	0
42	4	0	2	79	5	0	3	119	16	0	8(8)

Tabela A.2: Descrição dos 105 problemas de Hock e Schittkowski. Os valores em parênteses representam quantas das restrições são lineares.

$$\begin{aligned} \min \quad & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ \text{s. a} \quad & x_2^2 - x_1 \geq 0 \\ & x_1^2 - x_2 \geq 0 \\ & x_1^2 + x_2^2 - 1 \geq 0 \\ & -2 \leq x_1 \leq 0.5 \end{aligned}$$

- $x_{\text{ini}} = (-2, 1)$
- $x^* = (-0.78615048331, 0.618034533851)$
- $f(x^*) = 3.19033354957$

Uma outra questão relacionada a estes problemas de teste consiste no fato do conjunto viável do subproblema engordado conter pontos para os quais a função objetivo ou alguma das restrições não está bem definida. Esse seria o caso, por exemplo, se quiséssemos resolver o problema de minimizar $\sqrt{x}$ no conjunto $\{x \in \mathbb{R} \mid x \geq 0\}$. Para

qualquer $\gamma > 0$ o conjunto engordado $\{x \in \mathbb{R} \mid x \geq -\gamma\}$ contém pontos nos quais a função objetivo não está bem definida. O funcionamento do algoritmo nesse caso depende da forma como é tratada essa não definição. Na maior parte dos compiladores a raiz quadrada de um número negativo é `NaN` (*Not a Number*), por isso, por mais que tais pontos possam existir, a esperança é que, uma vez que os algoritmos utilizados para resolver os subproblemas se encontram em um ponto para o qual a função está bem definida, eles não desejem ir para pontos de não definição.

Não podemos ignorar esses casos especiais, pois o comportamento de `NaN` também depende do tipo de compilador e de linguagem utilizada. Se o seguinte pedaço de código estiver presente em um algoritmo:

```
se f(a) < f(b) então
   b = a
fim
```

e tivermos que `f(a) = NaN`, na maioria dos compiladores a condição sempre será falsa e nada será feito, como gostaríamos que acontecesse. Porém, se o código estivesse escrito da seguinte forma:

```
se f(a) >= f(b) então
   faça algo
senão /* caso f(a) < f(b) */
   b = a
fim
```

o algoritmo executaria o comando `b = a`, pois a condição também será falsa. Na verdade `NaN` não é igual, maior e nem menor que nenhum número considerado válido. Essa discussão transcende os problemas da coleção de Hock e Schittkowski e pode ser levada à qualquer outro problema.

A.3 Problemas vorazes

O fenômeno da voracidade pode ser descrito como a atração que determinados minimizadores irrestritos exercem sobre algoritmos baseados em penalidades que utilizam funções de mérito combinando a função objetivo e a inviabilidade das restrições. Tais minimizadores geralmente são atingidos rapidamente nas iterações iniciais e seu poder de atração é tão grande que os algoritmos têm que aumentar excessivamente o parâmetro de penalidade, tornando os subproblemas seguintes mal condicionados [CMMS10].

A voracidade em Lagrangianos Aumentados foi discutida em [CMMS10], onde também foram descritos 6 problemas que podem apresentar esse fenômeno. Retiramos outro de [HS81], por observarmos fenômeno semelhante, totalizando 7 problemas. Em [CMMS10, BCMM11] são apresentadas estratégias para contornar tal efeito. Apresentamos aqui a descrição dos problemas utilizados e os pontos iniciais.

- **Problema 1**
  - Problema retirado de [CMMS10]. Este problema possui apenas restrições de caixa, mas elas não devem ser tratadas de forma diferenciada e sim penalizadas
  - 100 variáveis e 100 restrições lineares de desigualdade.

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{n} x_i^3 \\ \text{s. a} & -x_i \leq 0, \quad i = 1, \ldots, n \end{array}$$

  - $n = 100$
  - $x_{\text{ini}} = (-7, \ldots, -7)$
- **Problema 2**
  - Problema retirado de [CMMS10], mas encontrado em [HS81].
  - 7 variáveis e 4 restrições não lineares de igualdade

$$\begin{array}{ll} \min & -x_1 x_2 x_3 \\ \text{s. a} & x_1 - 4.2\,\text{sen}(x_4)^2 = 0 \\ & x_2 - 4.2\,\text{sen}(x_5)^2 = 0 \\ & x_3 - 4.2\,\text{sen}(x_6)^2 = 0 \\ & x_1 + 2x_2 + 2x_3 - 7.2\,\text{sen}(x_7)^2 = 0 \end{array}$$

  - $x_{\text{ini}} = (1, 2, 3, 4, 5, 6, 7)$
- **Problema 3**
  - Problema retirado de [CMMS10].
  - 2 variáveis e 1 restrição não linear de igualdade

$$\begin{array}{ll} \min & -x_1 x_2^3 \\ \text{s. a} & x_1 x_2 - 4\,\text{sen}(x_1)^2 = 0 \end{array}$$

  - $x_{\text{ini}} = (1, 1)$
- **Problema 4**
  - Problema retirado de [CMMS10].
  - 2 variáveis e 1 restrição não linear de igualdade

$$\begin{array}{ll} \min & -x_1 \exp(-x_1 x_2) \\ \text{s. a} & -(x_1 + 1)^3 + 3(x_1 + 1)^2 - 1.5 + x_2 = 0 \end{array}$$

- $x_{\text{ini}} = (1, -1.5)$

- **Problema 5**
  - Problema retirado de [CMMS10].
  - 50 variáveis e 1 restrição não linear de desigualdade

$$\begin{aligned} \min \quad & -\sum_{i=1}^{n}(x_i^8 + x_i) \\ \text{s. a} \quad & \sum_{i=1}^{n} x_i^2 \leq 1 \end{aligned}$$

  - $n = 50$
  - $x_{\text{ini}} = (0.1, \ldots, 0.1)$

- **Problema 6**
  - 100 variáveis e 1 restrição não linear de desigualdade

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \varphi(x_i) \\ \text{s. a} \quad & \sum_{i=1}^{n} x_i^2 \leq 1 \end{aligned}$$

  - $n = 100$
  - $\varphi(t) = \begin{cases} \log(\cos(t)), & \text{se } \cos(t) > 0 \\ -10^{30}, & \text{caso contrário} \end{cases}$
  - $x_{\text{ini}} = (1/n, \ldots, 1/n)$

- **Problema 7**
  - Problema retirado de [HS81].
  - 4 variáveis e 3 restrições não lineares de igualdade

$$\begin{aligned} \min \quad & -x_1 x_2 x_3 x_4 \\ \text{s. a} \quad & x_1^3 + x_2^2 - 1 = 0 \\ & x_1^2 x_4 - x_3 = 0 \\ & x_4^2 - x_2 = 0 \end{aligned}$$

  - $x_{\text{ini}} = (0.8, 0.8, 0.8, 0.8)$

Referências Bibliográficas

[AADJLD09] M. A. Abramson, C. Audet, J. E. Dennis Jr. e S. Le Digabel. OrthoMads: A deterministic Mads instance with orthogonal directions. *SIAM Journal on Optimization*, 20(2):948–966, 2009.

[ABMS07] R. Andreani, E. G. Birgin, J. M. Martínez e M. L. Schuverdt. On Augmented Lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18:1286–1309, 2007.

[ACC+09] R. Andreani, S. L. C. Castro, J. L. Chela, A. Friedlander e S. A. Santos. An inexact-restoration method for nonlinear bilevel programming problems. *Computational Optimization and Applications*, 43(3), 2009.

[ADJ03] C. Audet e J. E. Dennis Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[ADJ06] C. Audet e J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[ADJ09] C. Audet e J. E. Dennis Jr. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472, 2009.

[AHM11] R. Andreani, G. Haeser e J. M. Martínez. On sequential optimality conditions for smooth constrained optimization. *Optimization*, 60(5):627–641, 2011.

[AHSS11a] R. Andreani, G. Haeser, M. L. Schuverdt e P. J. S. Silva. A relaxed constant positive linear dependence constraint qualification and applications. *Mathematical Programming*, 2011. DOI: 10.1007/s10107-011-0456-0.

[AHSS11b] R. Andreani, G. Haeser, M. L. Schuverdt e P. J. S. Silva. Two new constraint qualifications and applications. Disponível em Optimization Online, 2011.

[AMS05] R. Andreani, J. M. Martínez e M. L. Schuverdt. On the relation between Constant Positive Linear Dependence condition and Quasinormality Constraint Qualification. *Journal of Optimization Theory and Applications*, 125(2):473–483, 2005.

[And08] M. Andretta. *Tópicos em otimização com restrições lineares*. Tese de doutorado, Instituto de Matemática e Estatística - Universidade de São Paulo, 2008.

[Ano72] Anonymous. A new algorithm for optimization. *Mathematical Programming*, 3(1):124–128, 1972.

[BAEP11] M. A. Belen Arouxet, N. Echebest e E. A. Pillota. Active-set strategy in Powell's method for optimization without derivatives. *Computational & Applied Mathematics*, 30(1):171–196, 2011.

[BB05] F. V. Berghen e H. Bersini. CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. *Journal of Computational and Applied Mathematics*, 181(1):157–175, 2005.

[BCMM11] E. G. Birgin, E. V. Castelani, A. L. M. Martinez e J. M. Martínez. Outer trust-region method for constrained optimization. *Journal of Optimization Theory and Applications*, 150:142–155, 2011.

[BD93] S. Banerjia e R. A. Dwyer. Generating random points in a ball. *Communication in Statistics – Simulation and computation*, 22(4):1205–1209, 1993.

[Ber99] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 1999.

[BFMS11] L. F. Bueno, A. Friedlander, J. M. Martínez e F. N. C. Sobral. Inexact restoration method for derivative-free optimization with smooth constraints. Disponível em Optimization Online, 2011.

[BG08] R. H. Bielschowsky e F. A. M. Gomes. Dynamic control of infeasibility in equality constrained optimization. *SIAM Journal on Optimization*, 19(3):1299–1325, 2008.

[BM58] G. E. P Box e M. E. Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[BM05] E. G. Birgin e J. M. Martínez. Local convergence of an inexact-restoration method and numerical experiments. *Journal of Optimization Theory and Applications*, 127(2):229–247, 2005.

[Bue11] L. F. Bueno. *Otimização com restrições LOVO, Restauração Inexata e o equilíbrio inverso de Nash*. Tese de doutorado, Instituto de Matemática, Estatística e Computação Científica - Universidade Estadual de Campinas, 2011.

[Cas04] S. L. C. Castro. *Algoritmo de restauração inexata aplicado à resolução de problemas de programação matemática em dois níveis*. Tese de doutorado, Instituto de Matemática, Estatística e Computação Científica – Universidade Estadual de Campinas, 2004.

[CGST96] A. R. Conn, N. I. M. Gould, A. Sartenaer e Ph. L. Toint. Convergence properties of an Augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints. *SIAM Journal on Optimization*, 6(3):674–703, 1996.

[CGT91] A. R. Conn, N. I. M. Gould e Ph. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

[Cla83] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, New York, 1983. Reissued in 1990 by SIAM Publications, Philadelphia, as Vol. 5 in the series Classics in Applied Mathematics.

[CMMS10] E. V. Castelani, A. L. M. Martinez, J. M. Martínez e B. F. Svaiter. Addressing the greediness phenomenon in nonlinear programming by means of proximal Augmented Lagrangians. *Computational Optimization and Applications*, 46:229–245, 2010.

[CON02] CONVEST. Vestibular UNICAMP 15/16 anos, 2002.

[CP01] I. D. Coope e C. J. Price. On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, 11(4):859–869, 2001.

[CST97a] A. R. Conn, K. Scheinberg e Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. Em M. D. Buhmann e A. Iserles, editores, *Approximation theory and optimization: tributes to M.J.D. Powell*, pp. 83–108. Cambridge University Press, 1997.

[CST97b] A. R. Conn, K. Scheinberg e Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79(3):397–414, 1997.

[CST98] A. R. Conn, K. Scheinberg e Ph. L. Toint. A derivative free optimization algorithm in practice. Em *Proceedings of 7th*

AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, 1998.

[CSV09a] A. R. Conn, K. Scheinberg e L. N. Vicente. Global convergence of general derivative-free trust-region algorithms to first- and second-order critical points. *SIAM Journal on Optimization*, 20(1):387–415, 2009.

[CSV09b] A. R. Conn, K. Scheinberg e L. N. Vicente. *Introduction to Derivative-free Optimization*. MPS-SIAM Series on Optimization, 2009.

[DEMP11] M. A. Diniz-Ehrhardt, J. M. Martínez e L. G. Pedroso. Derivative-free methods for nonlinear programming with general lower-level constraints. *Computational & Applied Mathematics*, 30(1):19–52, 2011.

[DEMR08] M. A. Diniz-Ehrhardt, J. M. Martínez e M. Raydan. A derivative-free nonmonotone line-search technique for unconstrained optimization. *Journal of Computational and Applied Mathematics*, 219(2):383–397, 2008.

[DM87] B. P. Demidovich e I. A. Maron. *Computational Mathematics*. Lecture Notes in Mathematics. Mir Publishers, Moscow, 1987.

[DM02] E. D. Dolan e J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[ESV11] N. Echebest, M. L. Schuverdt e R. P. Vignau. Two derivative-free methods for solving underdetermined nonlinear systems of equations. *Computational & Applied Mathematics*, 30(1):217–245, 2011.

[FF10] A. Fischer e A. Friedlander. A new line search inexact restoration approach for nonlinear programming. *Computational Optimization and Applications*, 46(2):333–346, 2010.

[FG11] A. Friedlander e F. A. M. Gomes. Solution of a truss topology bilevel programming problem by means of an inexact restoration method. *Computational & Applied Mathematics*, 30(1), 2011.

[FMN09] G. Fasano, J. L. Morales e J. Nocedal. On the geometry phase in model-based algorithms for derivative-free optimization. *Optimization Methods & Software*, 24(1):145–154, 2009.

[GK10] J. D. Griffin e T. G. Kolda. Nonlinearly constrained optimization using heuristic penalty methods and asynchronous parallel generating set search. *Applied Mathematics Research Express*, 2010(1):36–62, 2010.

[GKV03] C. C. Gonzaga, E. W. Karas e M. Vanti. A globally convergent filter method for nonlinear programming. *SIAM Journal on Optimization*, 14(3):646–669, 2003.

[GT10] N. I. M. Gould e Ph. L Toint. Nonlinear programming without a penalty function or a filter. *Mathematical Programming*, 122(1):155–196, 2010.

[GT12] N. I. M. Gould e Ph. L Toint. Erratum to: Nonlinear programming without a penalty function or a filter. *Mathematical Programming*, 131(1):403–404, 2012.

[GTT11] S. Gratton, Ph. L. Toint e A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optimization Methods and Software*, 36(4–5):873–894, 2011.

[Him72] D. M. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, New York, 1972.

[HJ61] R. Hooke e T. A. Jeeves. "Direct Search" solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, 1961.

[HS81] W. Hock e K. Schittkowski. *Test Examples for Nonlinear Programming Codes*, volume 187 de *Lecture Notes in Economics and Mathematical Systems*. Springer, 1981. Available at `http://www.ai7.uni-bayreuth.de/`.

[Jah96] J. Jahn. *Introduction to the Theory of Nonlinear Optimization*. Springer-Verlag, Berlin, 1996.

[KGR10] E. W. Karas, C. C. Gonzaga e A. A. Ribeiro. Local convergence of filter methods for equality constrained non-linear programming. *Optimization*, 59(8):1153–1171, 2010.

[KLT03] T. G. Kolda, R. M. Lewis e V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[KLT06a] T. G. Kolda, R. M. Lewis e V. Torczon. A generating set direct search augmented Lagrangian algorithm for optimization with a combination of general and linear constraints. SANDIA Report SAND2006-5315, 2006.

[KLT06b] T. G. Kolda, R. M. Lewis e V. Torczon. Stationarity results for generating set search for linearly constrained optimization. *SIAM Journal on Optimization*, 17(4):943–968, 2006.

[LD11] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions On Mathematical Software*, 37(4):44:1–44:15, 2011.

[Lim70] E. L. Lima. *Elementos de topologia geral*. Instituto de Matemática Pura e Aplicada. Coleção elementos de matemática. Ao Livro Técnico, Rio de Janeiro, RJ, 1970.

[Lim83] E. L. Lima. *Espaços Métricos*. Projeto Euclides. Instituto de Matemática Pura e Aplicada, Rio de Janeiro, RJ, 1983.

[LLS10] G. Liuzzi, S. Lucidi e M. Sciandrone. Sequential penalty derivative-free methods for nonlinear constrained optimization. *SIAM Journal on Optimization*, 20(5):2614–2635, 2010.

[LS02] S. Lucidi e M. Sciandrone. On the global convergence of derivative-free methods for unconstrained optimization. *SIAM Journal on Optimization*, 13(1):97–116, 2002.

[LST02] S. Lucidi, M. Sciandrone e P. Tseng. Objective-derivative-free methods for constrained optimization. *Mathematical Programming*, 92(1):37–59, 2002.

[LT99] R. M. Lewis e V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, 9(4):1082–1099, 1999.

[LT00] R. M. Lewis e V. Torczon. Pattern search algorithms for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941, 2000.

[LT02] R. M. Lewis e V. Torczon. A globally convergent Augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4):1075–1089, 2002.

[LT10] R. M. Lewis e V. Torczon. A direct search approach to nonlinear programming problems using an Augmented Lagrangian method with explicit treatment of linear constraints. Technical report WM-CS-2010-01, 2010.

[Mar98] J. M. Martínez. Two-phase model algorithm with global convergence for nonlinear programming. *Journal of Optimization Theory and Applications*, 96(2):397–436, 1998.

[Mar01] J. M. Martínez. Inexact-restoration method with Lagrangian tangent decrease and new merit function for nonlinear programming. *Journal of Optimization Theory and Applications*, 111(1):39–58, 2001. Versão atualizada em `www.ime.usp.br/∼martinez`.

[Mar06] J. M. Martínez. Otimização prática usando o Lagrangiano Aumentado. Departamento de Matemática Aplicada, Universidade Estadual de Campinas, Brasil, 2006. Versão atualizada em `www.ime.usp.br/∼martinez`.

[MF67] O. L. Mangasarian e S. Fromovitz. The Fritz-John necessary optimality conditions in presence of equality and inequality constraints. *Journal of Mathematical Analisys and Applications*, 17:37–47, 1967.

[MP00] J. M. Martínez e E. A. Pillota. Inexact restoration algorithms for constrained optimization. *Journal of Optimization Theory and Applications*, 104(1):135–163, 2000.

[MS78] B. A. Murtagh e M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14(1):41–72, 1978.

[MS95] J. M. Martínez e S. A. Santos. *Métodos Computacionais de Otimização*. Publicações do 20º Colóquio Brasileiro de Matemática, 1995. (Atualizado em 1998).

[MS03] J. M. Martínez e B. F. Svaiter. A practical optimality condition without constraint qualifications for nonlinear programming. *Journal of Optimization Theory and Applications*, 118:117–133, 2003.

[MS11] J. M. Martínez e F. N. C. Sobral. Constrained derivative-free optimization on thin domains. Disponível em Optimization Online, 2011.

[Mul59] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.

[MW09] J. J. Moré e S. M Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[NM65] J. Nelder e R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[NW99] J. Nocedal e S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[Ped09] L. G. Pedroso. *Programação não linear sem derivadas*. Tese de doutorado, Instituto de Matemática, Estatística e Computação Científica – Universidade Estadual de Campinas, 2009.

[PH69] D. A. Paviani e D. M. Himmelblau. Constrained nonlinear optimization by heuristic programming. *Operations Research*, 17(5):872–882, 1969.

[Pla09] T. D. Plantenga. *HOPSPACK 2.0 User Manual*, 2009. SANDIA Report SAND2009-6265.

[Pow70] M. J. D. Powell. A hybrid method for nonlinear equations. Em P. Rabinowitz, editor, *Numerical methods for nonlinear algebraic equations*, pp. 87–114. Gordon and Breach, London, 1970.

[Pow94] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. Em S. Gomez e J. Hennart, editores, *Advances in Optimization and Numerical Analysis*, Mathematics and Its Applications, pp. 51–67. Kluwer Academic (Dordrecht), 1994.

[Pow02] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[Pow06] M. J. D. Powell. The NEWUOA software for unconstrained minimization without derivatives. Em G. Di Pillo e M. Roma, editores, *Large-Scale Nonlinear Optimization*, volume 83 de *Nonconvex Optimization and its Applications*, pp. 255–297. Springer US, 2006.

[Pow09] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report DAMTP 2009/NA06, Cambridge, 2009.

[Pow11] M. J. D. Powell. On the convergence of trust region algorithms for unconstrained minimization without derivatives. Technical report DAMTP 2011/NA01, Cambridge, 2011.

[Sch79] L. Schrage. A more portable Fortran random number generator. *ACM Transactions on Mathematical Software*, 5(2):132–139, 1979.

[Sch87] K. Schittkowski. *More Test Examples for Nonlinear Programming*, volume 282 de *Lecture Notes in Economics and Mathematical Systems*. Springer, 1987.

[SS02] B. Schölkopf e A. Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization and Beyond*. The MIT Press, Cambridge, Massachusetts, London, England, 2002.

[ST10] K. Scheinberg e Ph. L. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM Journal on Optimization*, 02(6):3512–3532, 2010.

[TAN] Trustable algorithms for nonlinear general optimization. `www.ime.usp.br/~egbirgin/tango` (Acessado em 10/02/2012).

[Tor97] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.

[VC10] L. N. Vicente e A. L. Custódio. Analysis of direct searches for discontinuous functions. *Mathematical Programming*, pp. 1–27, 2010. DOI: 10.1007/s10107-010-0429-8.

[WB06] A. Wächter e L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[Win73] D. Winfield. Function minimization by interpolation in a data table. *IMA Journal of Applied Mathematics*, 12(3):339–347, 1973.