Tópicos em otimização com restrições lineares

Marina Andretta

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Ernesto G. Birgin

Durante o desenvolvimento deste trabalho a autora recebeu auxílio financeiro do CNPq.

São Paulo, agosto de 2008

Tópicos em otimização com restrições lineares

Este exemplar corresponde à redação
final da tese devidamente corrigida
e defendida por Marina Andretta
e aprovada pela Comissão Julgadora.

Banca Examinadora:

Prof. Dr. Ernesto G. Birgin (orientador) - IME-USP.

Prof. Dr. Carlos Humes Jr - IME-USP.

Prof. Dr. José Mario Martínez - IMECC-UNICAMP.

Prof. Dr. Nelson Maculan Filho - COPPE-UFRJ.

Prof. Dr. Marcos Raydan - UCV e USB, Venezuela.

Para minha mãe e meu pai.

Agradecimentos

Ao professor Ernesto, pelas discussões, conversas, idéias e paciência nestes anos de trabalho. Ao professor José Mario Martínez por idéias e sugestões feitas ao longo do desenvolvimento deste trabalho.

Ao grupo de otimização contínua, ao pessoal da "salinha", aos professores que tive, por todo apoio, dicas e suporte técnico.

A toda minha famíla, em especial à Míriam, Maira, Dina, Euler, Neto e Miguilim, por tudo. E, claro, à minha mãe e meu pai, que sempre me deram todo o apoio e aos quais dedico este trabalho.

A todos os meus amigos, do IME e de outros lugares, pelo apoio, conversas, bagunças, festas, pizzas e muito mais. Não vou citar nomes para não ser injusta com ninguém, mas vocês sabem quem são!

Resumo

Métodos do tipo Lagrangiano Aumentado são muito utilizados para minimização de funções sujeitas a restrições gerais. Nestes métodos, podemos separar o conjunto de restrições em dois grupos: restrições *fáceis* e restrições *difíceis*. Dizemos que uma restrição é *fácil* se existe um algoritmo disponível e eficiente para resolver problemas restritos a este tipo de restrição. Caso contrário, dizemos que a restrição é *difícil*. Métodos do tipo Lagrangiano aumentado resolvem, a cada iteração, problemas sujeitos às restrições *fáceis*, penalizando as restrições *difíceis*.

Problemas de minimização com restrições lineares aparecem com freqüência, muitas vezes como resultados da aproximação de problemas com restrições gerais. Este tipo de problema surge também como subproblema de métodos do tipo Lagrangiano aumentado. Assim, uma implementação eficiente para resolver problemas com restrições lineares é relevante para a implementação eficiente de métodos para resolução de problemas de programação não-linear.

Neste trabalho, começamos considerando *fáceis* as restrições de caixa. Introduzimos Betra-esparso, uma versão de Betra [3] para problemas de grande porte. Betra é um método de restrições ativas que utiliza regiões de confiança para minimização em cada face e gradiente espectral projetado para sair das faces. Utilizamos Betra (denso ou esparso) na resolução dos subproblemas que surgem a cada iteração de Algencan (um método de Lagrangiano aumentado). Para decidir qual algoritmo utilizar para resolver cada subproblema, desenvolvemos regras que escolhem um método para resolver o subproblema de acordo com suas características.

Em seguida, introduzimos dois algoritmos de restrições ativas desenvolvidos para resolver problemas com restrições lineares (Betralin e Genlin). Estes algoritmos utilizam, a cada iteração, o método do Gradiente Espectral Projetado Parcial quando decidem mudar o conjunto de restrições ativas. O método do Gradiente Espectral Projetado Parcial foi desenvolvido especialmente para este propósito. Neste método, as projeções são computadas apenas em um subconjunto das restrições, com o intuito de torná-las mais eficientes.

Por fim, tendo introduzido um método para minimização com restrições lineares, consideramos como *fáceis* as restrições lineares. Incorporamos Betralin e Genlin ao arcabouço de Lagrangianos aumentados e verificamos experimentalmente a eficiência e eficácia destes métodos que trabalham explicitamente com restrições lineares e penalizam as demais.

Palavras-chave: minimização com restrições lineares, métodos de restrições ativas, gradiente espectral projetado, métodos de Lagrangiano aumentado, programação não-linear.

Abstract

Augmented Lagrangian methods are widely used to solve general nonlinear programming problems. In these methods, one can split the set of constraints in two groups: the set of *easy* and *hard* constraints. A constraint is called *easy* if there is an efficient method available to solve problems subject to that kind of constraint. Otherwise, the constraints are called *hard*. Augmented Lagrangian methods solve, at each iteration, problems subject to the set of *easy* constraints while penalizing the set of *hard* constraints.

Linearly constrained problems appear frequently, sometimes as a result of a linear approximation of a problem, sometimes as an augmented Lagrangian subproblem. Therefore, an efficient method to solve linearly constrained problems is important for the implementation of efficient methods to solve nonlinear programming problems.

In this thesis, we begin by considering box constraints as the set of *easy* constraints. We introduce a version of Betra to solve large scale problems. Betra is an active-set method that uses a trust-region strategy to work within the faces and spectral projected gradient to leave the faces. To solve each iteration's subproblem of Algencan (an augmented Lagrangian method) we use either the dense or the sparse version of Betra. We develope rules to decide which box-constrained inner solver should be used at each augmented Lagrangian iteration that considers the main characteristics of the problem to be solved.

Then, we introduce two active-set methods to solve linearly constrained problems (Betralin and Genlin). These methods use Partial Spectral Projected Gradient method to change the active set of constraints. The Partial Spectral Projected Gradient method was developed specially for this purpose. It computes projections onto a subset of the linear constraints, aiming to make the projections more efficient.

At last, having introduced a linearly-constrained solver, we consider the set of linear constraints as the set of *easy* constraints. We use Betralin and Genlin in the framework of augmented Lagrangian methods and verify, using numerical experiments, the efficiency and robustness of those methods that work with linear constraints and penalize the nonlinear constraints.

Keywords: linearly-constrained methods, active-set methods, spectral projected gradient, augmented Lagrangian methods, nonlinear programming.

Sumário

Introdução

Estamos interessados em resolver problemas de programação não-linear definidos da seguinte forma:

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & h(x) = 0 \\ & g(x) \leq 0 \\ & x \in \Omega, \end{array} \tag{1}$$

onde $h : I\!R^n \to I\!R^m, g : I\!R^n \to I\!R^p, f : I\!R^n \to I\!R$ são suaves e $\Omega \in I\!R^n$ é um conjunto de restrições *fáceis*. Quando dizemos que Ω é um conjunto de restrições *fáceis*, queremos dizer que existem métodos eficientes para resolver problemas restritos ao conjunto Ω.

Uma maneira de resolver o problema (1) é utilizar métodos do tipo Lagrangiano aumentado. Nestes métodos, penalizamos as restrições h_i e g_j e, a cada iteração, resolvemos aproximadamente um subproblema de minimização de uma função de Lagrangiano aumentado (neste trabalho, PHR [30, 43, 48]) restrita ao conjunto Ω.

Há várias vantagens no uso de métodos de Lagrangiano aumentado [7]. Algumas delas são:

1. Progresso na análise e implementação de métodos de otimização eficientes para resolver problemas simples produzem um efeito positivo quase imediato na eficiência e robustez dos métodos de Lagrangiano aumentado correspondentes.

2. Minimização global dos subproblemas implica em convergência a minimizadores globais do método de Lagrangiano aumentado.

3. Em vários problemas práticos, a Hessiana do Lagrangiano é uma matriz estruturalmente densa (ou seja, qualquer entrada pode ser diferente de zero para Hessianas calculadas em diferentes pontos) mas geralmente esparsa (dado um ponto no domínio, a Hessiana do Lagrangiano neste ponto é esparsa). Métodos Newtonianos geralmente têm dificuldades em problemas deste tipo, tanto em termos de memória como em termos de tempo computacional, pois o padrão de esparsidade muda a cada iteração. Esta dificuldade é praticamente irrelevante para métodos do tipo Lagrangiano aumentado que utilizam subalgoritmos que necessitam de pouca memória.

4. Independentemente da densidade da Hessiana do Lagrangiano, a estrutura do sistema KKT pode ser muito ruim para fatorações esparsas. Este é um problema grave para

métodos Newtonianos, mas não para boas implementações de métodos de Lagrangiano aumentado baseados na formulação PHR.

5. Se o problema de programação não-linear possui muitas restrições de desigualdade, o uso de variáveis de folga feito por métodos de pontos interiores (também usado em [2, 18]) pode não ser conveniente. Há várias maneiras de reduzir os efeitos causados por muitas variáveis de folga, mas eles podem não ser tão efetivos como não usar variáveis de folga. A desvantagem de não usar variáveis de folga é a ausência de segundas derivadas contínuas da função de Lagrangiano aumentado. Em muitos casos, isto não se mostra um inconveniente na prática (veja [4]).

6. Problemas de grande porte têm a desvantagem óbvia em termos de necessidade de armazenamento. Métodos do tipo Lagrangiano aumentado apresentam uma solução: dados do problema podem ser computados apenas quando necessários, usados quando preciso para os subproblemas, sem nenhum armazenamento. Isto não é possível quando se usam métodos matriciais, independentemente da estratégia de esparsidade adotada.

7. Se, na solução do problema, não valem condições de qualificação de restrição fortes, o desempenho de métodos Newtonianos podem ser fortemente afetados. Métodos de Lagrangiano aumentado não são sensíveis a este tipo de desvantagem.

Algencan é uma implementação de um método de Lagrangiano aumentado com $\Omega = \{x \in I\!R^n \mid \ell \leq x \leq u\}$, que utiliza Gencan [8] para resolver os subproblemas de minimização com restrições de caixa. Em [3] foi introduzido um método (chamado Betra) para minimização em caixas, mais adequado para problemas de pequeno porte. Estamos interessados em desenvolver uma versão para grande porte de Betra e, então, desenvolver uma implementação de Algencan que utilize Betra, na versão densa ou esparsa, ou Gencan como subalgoritmo, o que for mais apropriado para o problema a ser resolvido. Desejamos também criar regras para decidir qual algoritmo interno deve ser utilizado a cada iteração, baseadas no número de variáveis, grau de esparsidade da matriz Hessiana do Lagrangiano aumentado, etc.

Estamos interessados, também, em desenvolver um método de Lagrangiano aumentado, baseado em Algencan, que considere $\Omega = \{x \in I\!R^n \mid Ax = b,\ Cx \leq d,\ \ell \leq x \leq u\}$, ou seja, trabalhe explicitamente com restrições lineares e de caixa e penalize as demais restrições. Isso porque restrições lineares possuem uma estrutura conhecida e desejamos verificar se penalizá-las ou não torna o método de Lagrangiano aumentado mais eficiente e robusto.

Para o desenvolvimento deste novo método, que deixa as restrições lineares no nível inferior, é necessário desenvolver um método eficiente para minimização com restrições lineares. Os princípios reitores do método para minimização com restrições lineares são os seguintes:

1. Metodologia de pontos admissíveis: no caso de restrições lineares, consideramos que a facilidade relativa de preservar a admissibilidade dos iterandos de um método sobrepassa qualquer vantagem imaginável de sacrificar tal propriedade. Este é um princípio norteador do desenvolvimento do método aqui preconizado.

2. Uso de estratégia de restrições ativas: a região admissível do problema se divide naturalmente no que chamamos faces. Uma face da região admissível se caracteriza por

um conjunto dado de restrições satisfeito estritamente e o conjunto complementar é satisfeito de maneira não-estrita. A região viável é a união disjunta de suas faces. Os algoritmos de restrições ativas, conceitualmente, visitam as diferentes faces da região e as exploram, até que decidem abandoná-las.

3. Uso de estratégias de gradiente espectral projetado para o abandono de faces: dentro de cada face, o algoritmo funciona como um minimizador sem restrições. Pode continuar funcionando desta maneira até que uma de duas possibilidades aconteça. Os iterandos atingem uma face de dimensão menor (na fronteira) ou chegam a um ponto tão bom, dentro da face, que não vale a pena continuar sua exploração. Neste caso, se um ponto KKT do problema original ainda não foi atingido, dizemos que a face corrente deve ser abandonada. O abandono da face é um problema delicado porque, essencialmente, comporta a resolução de um novo problema de otimização, que pode ser relativamente custoso.

O algoritmo interno é o encarregado de minimizar dentro de uma face dada. Uma face se caracteriza por um conjunto de restrições de igualdade, portanto o algoritmo interno minimiza a função, restrita a uma variedade afim. Dentre as estratégias existentes para este objetivo, somos partidários daquelas baseadas no espaço nulo determinado pelas restrições. Isto significa que há de se determinar uma base do subespaço paralelo à variedade afim sob consideração e reduzir o problema a um problema irrestrito, cujas variáveis são os coeficientes na base desse espaço nulo. Reduzido a esta situação, o problema pode ser de grande porte ou não, dependendo da dimensão da variedade afim. Isso motiva o uso de diferentes algoritmos internos, que deverão ser escolhidos e analisados.

Quando uma face está esgotada, ela deve ser abandonada. A maneira mais clássica de abandonar uma face é a introduzida por Rosen em seu algoritmo clássico de gradiente projetado:

(a) Considera-se que o esgotamento de uma face é dado pela anulação do gradiente reduzido a ela.

(b) Verifica-se a condição KKT, ou seja, calculam-se os multiplicadores de Lagrange. Se todos eles tem o sinal "correto" o ponto é KKT.

(c) Abandona-se uma restrição para a qual o multiplicador de Lagrange tem o sinal incorreto.

Este procedimento ainda tem vigência quando algumas condições são satisfeitas. O número de restrições ativas deve ser pequeno e as restrições devem ser linearmente independentes. Em geral, deve-se apelar a outros procedimentos mais eficazes. Com o procedimento de Rosen, apenas uma restrição é abandonada em cada iteração deste tipo, o que pode fazer o processo de resolver o problema muito caro. Uma alternativa é definir um sistema de coordenadas local pelo qual o problema do abandono fica reduzido a um problema local de minimização em caixa. Desta maneira, podem ser abandonadas muitas restrições ao mesmo tempo. Entretanto, a existência de outras restrições muito próximas às abandonadas (ou eventualmente coincidentes, no caso degenerado) motiva a introdução de outros procedimentos mais abrangentes. Uma possibilidade é usar o gradiente espectral projetado [11, 12] no conjunto total de restrições ou em um subconjunto delas, aquelas satisfeitas com menos folga.

Este trabalho está organizado da seguinte maneira: No Capítulo 1, apresentamos novas versões de Algencan, que utilizam Betra denso ou esparso como subalgoritmo. Introduzimos regras para decidir qual algoritmo interno será utilizado em cada iteração de Lagrangiano aumentado. Apresentamos detalhes da implementação de cada um dos métodos, resultados numéricos comparando estes métodos entre si e com outros conhecidos para resolução de problemas de programação não-linear. No Capítulo 2, apresentamos duas versões de um novo método de restrições ativas para minimização com restrições lineares. Explicamos o embasamento teórico, mostramos a teoria de convergência do método, apresentamos resultados numéricos comparando o desempenho das duas versões entre si e com métodos conhecidos que resolvem problemas com restrições lineares. No Capítulo 3, apresentamos duas novas versões de Algencan, que utilizam os métodos do Capítulo 2 como subalgoritmos. Ou seja, trabalham explicitamente com restrições lineares e de caixa e penalizam as demais. Por fim, apresentamos as conclusões finais deste trabalho e idéias para trabalhos futuros.

Capítulo 1

Lagrangiano aumentado com restrições de caixa no nível inferior

Muitos problemas práticos podem ser escritos na forma

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & x \in \Omega_1 \cap \Omega_2, \end{array} \tag{1.1}$$

com Ω_2 tal que subproblemas da forma

$$\begin{array}{ll} \text{Minimizar} & F(x) \\ \text{sujeita a} & x \in \Omega_2 \end{array} \tag{1.2}$$

são muito mais simples de resolver do que o problema (1.1). Ou seja, existem algoritmos eficientes para resolver (1.2) que não podem ser aplicados a (1.1).

Problemas da forma (1.1) podem ser resolvidos usando métodos de Lagrangiano aumentado. Estes métodos penalizam as restrições Ω_1 e, a cada iteração, resolvem um subproblema do tipo (1.2). Considere o problema de programação não-linear

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & h_1(x) = 0 \\ & g_1(x) \leq 0 \\ & h_2(x) = 0 \\ & g_2(x) \leq 0, \end{array} \tag{1.3}$$

onde $h_1 : \mathbb{R}^n \to \mathbb{R}^{m_1}, g_1 : \mathbb{R}^n \to \mathbb{R}^{p_1}, h_2 : \mathbb{R}^n \to \mathbb{R}^{m_2}, g_2 : \mathbb{R}^n \to \mathbb{R}^{p_2}, f : \mathbb{R}^n \to \mathbb{R}$ são suaves. Defina $\Omega_1 = \{x \in \mathbb{R}^n \mid h_1(x) = 0 \text{ e } g_1(x) \leq 0\}$ e $\Omega_2 = \{x \in \mathbb{R}^n \mid h_2(x) = 0 \text{ e } g_2(x) \leq 0\}$. A mais popular e, provavelmente, mais eficiente função de Lagrangiano aumentado (veja [4]) é dada pela fórmula PHR (Powell-Hestenes-Rockafellar [30, 43, 48]):

$$L_\rho(x, \lambda, \mu) = f(x) + \frac{\rho}{2}\left\{ \sum_{i=1}^{m_1} \left[h_{1i}(x) + \frac{\lambda_i}{\rho} \right]^2 + \sum_{i=1}^{p_1} \left[\max\left(0, g_{1i}(x) + \frac{\mu_i}{\rho}\right) \right]^2 \right\}$$

para todo $x \in \mathbb{R}^n, \lambda \in \mathbb{R}^{m_1}, \mu \in \mathbb{R}^{p_1}_+$, $\rho > 0$.

Métodos de Lagrangiano aumentado que usam a formulação PHR para resolver o problema (1.3) são baseados na minimização (aproximada) iterativa de L_ρ com respeito a Ω_2, seguida da atualização do parâmetro de penalidade ρ e dos multiplicadores de Lagrange λ e μ. Em [1], foi introduzido um método de Lagrangiano aumentado usando a formulação PHR que não usa variáveis de folga e pode ter restrições arbitrárias no conjunto de restrições do nível inferior Ω_2. Convergência baseada nas condições CPLD e a limitação dos parâmetros de penalidade foram provados em [1], dadas hipóteses razoáveis sobre o problema.

A forma geral do método de Lagrangiano aumentado baseado na formulação PHR é a seguinte:

Algoritmo 1.1 (Método de Lagrangiano aumentado usando formulação PHR)

Sejam $\lambda_{\min} < \lambda_{\max}$, $\mu_{\max} > 0$, $\gamma > 1$, $0 < \tau < 1$. *Seja* $\{\varepsilon_k\}$ *uma seqüência de números não negativos tal que* $\lim_{k\to\infty} \varepsilon_k = 0$. *Sejam* $\lambda_i^1 \in [\lambda_{\min}, \lambda_{\max}]$, $i = 1, \ldots, m_1$, $\mu_i^1 \in [0, \mu_{\max}]$, $i = 1, \ldots, p_1$ *e* $\rho_1 > 0$. *Faça* $k \leftarrow 1$.

Passo 1. *Resolução do subproblema*

Calcule $x_k \in I\!R^n$ uma solução aproximada de

$$\text{Minimizar } L_{\rho_k}(x, \lambda^k, \mu^k) \text{ sujeita a } x \in \Omega_2. \tag{1.4}$$

Passo 2. Defina

$$V_i^k = \max\left\{g_{1i}(x_k), -\frac{\mu_i^k}{\rho_k}\right\},\ i = 1, \ldots, p_1.$$

Se $k = 1$ ou $\max\{\|h_1(x_k)\|_\infty, \|V_i^k\|_\infty\} \leq \tau \max\{\|h_1(x_{k-1})\|_\infty, \|V_i^{k-1}\|_\infty\}$ então defina $\rho_{k+1} = \rho_k$. Senão, defina $\rho_{k+1} = \gamma\rho_k$.

Passo 3. Calcule $\lambda_i^{k+1} \in [\lambda_{\min}, \lambda_{\max}]$, $i = 1, \ldots, m_1$ e $\mu_i^{k+1} \in [0, \mu_{\max}]$, $i = 1, \ldots, p_1$. Faça $k \leftarrow k + 1$ e volte para o Passo 1.

Em implementações práticas do Algoritmo 1.1, calculamos $\lambda_i^{k+1} = \min\{\max\{\lambda_{\min}, \lambda_i^k + \rho_k h_{1i}(x_k)\}, \lambda_{\max}\}$ e $\mu_i^{k+1} = \min\{\max\{0, \mu_i^k + \rho_k g_{1i}(x_k)\}, \mu_{\max}\}$. Estas são estimativas de primeira ordem dos multiplicadores de Lagrange com salvaguardas. As salvaguardas definidas por $\lambda_{\min}$, $\lambda_{\max}$ e $\mu_{\max}$ são necessárias para provar resultados de convergência global. Sem estas salvaguardas são necessárias hipóteses fortes sobre o problema para garantir a limitação dos parâmetros de penalidade.

Vamos supor que f, g e h admitem primeira (às vezes, segunda) derivada contínua. Observe que L_ρ possui primeiras derivadas contínuas em relação a x, mas segundas derivadas não estão definidas em pontos tais que $g_1(x) + \mu_i/\rho = 0$.

Diferentes algoritmos de Lagrangiano aumentado diferem apenas no Passo 1. Em cada caso, precisamos de uma definição precisa para a solução aproximada de (1.4). Considere Ω_2 com restrições arbitrárias e o Algoritmo 1.2 como o Algoritmo 1.1 trocando apenas o Passo 1.

Algoritmo 1.2

Considere Ω_2 *com restrições arbitrárias e o Algoritmo 1.2 como o Algoritmo 1.1 trocando o Passo 1.*

Passo 1. Calcule (se possível) $x_k \in I\!R^n$ tal que existam $v^k \in I\!R^{m_2}$ e $u^k \in I\!R^{p_2}$ que satisfaçam

$$\begin{array}{l} \|L_{\rho_k}(x_k, \lambda^k, \mu^k) + \sum_{i=1}^{m_2} v^k \nabla h_{2i}(x_k) + \sum_{i=1}^{p_2} u^k \nabla g_{2i}(x_k)\| \leq \varepsilon_k, \\ u_i^k \geq 0,\ g_{2i}(x_k) \leq \varepsilon_k,\ \forall i = 1, \ldots, p_2, \\ g_{2i}(x_k) < -\varepsilon_k \Rightarrow u_i^k = 0,\ \forall i = 1, \ldots, p_2, \\ \|h_2(x_k)\| \leq \varepsilon_k. \end{array} \tag{1.5}$$

Os principais teoremas de convergência provados em [1] para o Algoritmo 1.2 seguem.

Teorema 1.1. *Seja $\{x_k\}$ a seqüência gerada pelo Algoritmo 1.2. Seja x^* o ponto limite de $\{x_k\}$. Então, se a seqüência de parâmetros de penalidade $\{\rho_k\}$ for limitada, o ponto limite x^* é viável. Caso contrário, vale ao menos uma das seguintes possibilidades:*

(i) x^* é um ponto KKT do problema

$$\begin{array}{ll} \textit{Minimizar} & \frac{1}{2}\left[\sum_{i=1}^{m_1}[h_1(x)]_i^2 + \sum_{i=1}^{p_1}\max\{0, [g_1(x)]_i^2\}\right] \\ \textit{sujeita a} & x \in \Omega_2; \end{array}$$

(ii) x^* não satisfaz a condição de qualificação de restrição CPLD associada a Ω_2.

Teorema 1.2. *Seja $\{x_k\}$ a seqüência gerada pelo Algoritmo 1.2. Suponha que x^* seja um ponto limite viável do problema (1.3) que satisfaz a condição de qualificação de restrição CPLD relacionada ao conjunto de todas as restrições. Então, x^* é um ponto KKT do problema (1.3).*

Foi provado também que, sob condições razoáveis, a seqüência de parâmetros de penalidade $\{\rho_k\}$ é limitada (veja [1]).

Estamos interessados no caso em que Ω_2 é um conjunto de restrições de caixa, ou seja, $\Omega_2 = \{x \in I\!R^n \mid \ell \leq x \leq u\}$. Por continuidade da função Lagrangiano aumentado e compacidade de Ω_2, esta definição garante que existe um minimizador global. Claramente, o Algoritmo 1.2 pode ser aplicado a este problema e os Teoremas 1.1 e 1.2 valem. No entanto, no caso específico de caixas, podemos definir o Algoritmo 1.3 como o Algoritmo 1.1, trocando apenas o Passo 1.

Algoritmo 1.3

Considere $\Omega_2 = \{x \in I\!R^n \mid \ell \leq x \leq u\}$. e o Algoritmo 1.3 como o Algoritmo 1.1 trocando o Passo 1.

Passo 1. Calcule x_k tal que

$$x_k \in \Omega_2 \text{ e } \|P_{\Omega_2}(x_k - \nabla L_{\rho_k}(x_k, \lambda^k, \mu^k)) - x_k\| \leq \varepsilon_k, \tag{1.6}$$

onde $P_{\Omega_2}(\cdot)$ é a projeção ortogonal em Ω_2.

A condição (1.6) implica nas condições (1.5). Os Teoremas 1.1 e 1.2 obviamente se aplicam ao Algoritmo 1.3. Note que, como todos os pontos de Ω_2 satisfazem CPLD, o Teorema 1.1 garante que os pontos limite da seqüência gerada $\{x_k\}$ são pontos KKT de

$$\begin{array}{ll} \text{Minimizar} & \frac{1}{2}\left[\sum_{i=1}^{m_1}[h_1(x)]_i^2 + \sum_{i=1}^{p_1}\max\{0, [g_1(x)]_i^2\}\right] \\ \text{sujeita a} & x \in \Omega_2. \end{array}$$

Algencan é uma implementação do Algoritmo 1.3 que utiliza Gencan [8] para resolver o Passo 1. Esta implementação faz parte do projeto Tango [55]. Em [3] foi introduzido um método (chamado Betra) para minimização em caixas, mais adequado para problemas de pequeno porte. Estamos interessados em desenvolver uma versão para grande porte de Betra e, então, desenvolver uma implementação de Algencan que utilize Betra, na versão densa ou esparsa, ou Gencan como subalgoritmo, o que for mais apropriado para o problema a ser resolvido. Desejamos também criar regras para decidir qual algoritmo interno deve ser utilizado a cada iteração, baseadas no número de variáveis, grau de esparsidade da matriz Hessiana do Lagrangiano aumentado, etc. Como Betra pode ser usado para calcular um ponto que satisfaça a condição (1.6), as novas versões de Algencan têm convergência garantida.

Na Seção 1.1 apresentamos os algoritmos Gencan e Betra, mostramos a versão esparsa de Betra (Betra-esparso) e como é feita a escolha automática do algoritmo interno na nova versão de Algencan. Na Seção 1.2 apresentamos alguns detalhes de implementação da nova versão de Algencan, que pode usar Gencan, Betra, Betra-esparso ou escolha automática do algoritmo interno. Na Seção 1.3 apresentamos e comparamos os resultados numéricos obtidos pelas 4 diferentes versões de Algencan nos problemas das coleções Cuter [15] e La Cumparsita [56]. Comparamos também os resultados obtidos pelas diferentes versões de Algencan com os obtidos por Lancelot B [18, 19, 20], outro método conhecido para resolver o problema (1.3). Na Seção 1.4 apresentamos as conclusões.

1.1 Algoritmo interno: método de restrições ativas para minimização em caixas

O subproblema a ser resolvido a cada iteração do método de Lagrangiano aumentado é

$$\text{Minimizar } \bar{f}(x) \text{ sujeita a } x \in \Omega, \tag{1.7}$$

onde $\Omega = \{x \in I\!R^n \mid \ell \leq x \leq u\}$, $\ell, u \in I\!R^n$, $\ell \leq u$, e $\bar{f} : I\!R^n \to I\!R \subset \mathcal{C}^2$.

Para resolver este problema, tanto Betra como Gencan utilizam um método de restrições ativas. Se fosse possível saber o conjunto de restrições ativas na solução, o ponto x^* solução do problema (1.7) seria "facilmente" calculado (pois o problema se tornaria irrestrito). Métodos de restrições ativas tentam, a cada iteração, descobrir qual o conjunto de restrições ativas na solução. Se detectam que o conjunto W de restrições que supunham ser ativas na solução está incorreto, utilizam algum critério para acrescentar ou remover restrições de W.

Como em [3, 8], dividimos a região viável Ω em faces abertas disjuntas. Para todo $I \subset \{1, 2, \ldots, n, n+1, n+2, \ldots, 2n\}$ definimos

$$F_I = \{x \in \Omega \mid x_i = \ell_i \text{ se } i \in I,\ x_i = u_i \text{ se } n + i \in I, \ell_i < x_i < u_i \text{ caso contrário}\}.$$

O conjunto Ω é a união das faces abertas. Definimos V_I o menor espaço afim que contém F_I e S_I o subespaço linear paralelo a V_I. Definimos o gradiente projetado contínuo da seguinte maneira:

$$g_P(x) = P_\Omega(x - g(x)) - x$$

e, para todo $x \in F_I$, definimos o gradiente interno como

$$g_I(x) = P_{S_I}[g_P(x)].$$

A idéia do método de restrições ativas apresentado em [3, 8] é a seguinte: o conjunto viável é dividido em faces abertas e, a cada iteração k, tem-se um ponto x_k pertencente a uma dessas faces. No início de cada iteração, verifica-se qual a relação entre a norma do gradiente interno e a norma do gradiente projetado contínuo. Quando a norma do gradiente projetado contínuo é "muito maior" que a norma do gradiente interno, decide-se mudar de face e uma iteração de SPG [11, 12] é usada para definir o novo ponto x_{k+1}, pertencente a uma nova face aberta. Quando a norma do gradiente projetado contínuo não é "muito maior", decide-se continuar na mesma face e utiliza-se uma iteração de um algoritmo "irrestrito" para obter um ponto x_{k+1} no fecho da face atual, no qual a função tem um valor menor do que em x_k. Este algoritmo "irrestrito" deve ter o cuidado de manter o ponto x_{k+1} viável.

Algoritmo 1.4 (Método de restrições ativas)

Dados $x_0 \in \Omega$ e $\eta \in (0, 1)$. Faça $k \leftarrow 0$.

Passo 1. *Critério de convergência*

Se $g_P(x_k) = 0$ então pare e devolva x_k como solução

Passo 2. *Decisão de permanecer ou não na mesma face*

Se

$$\|g_I(x_k)\| \geq \eta \|g_P(x_k)\| \tag{1.8}$$

então considere o problema (1.7) restrito às variáveis livres e use uma iteração de um algoritmo "irrestrito" para calcular $x_{k+1} \in \bar{F}_I$. Se o algoritmo "irrestrito" terminar sua execução com x_k declarando *"Ponto estacionário de primeira ordem"* ou *"Ponto estacionário de segunda ordem"*, pare e devolva x_k como solução, declarando o mesmo que o algoritmo "irrestrito". Senão, faça uma iteração do SPG para calcular x_{k+1}.

Passo 3. Faça $k \leftarrow k + 1$ e volte para o Passo 1.

No Passo 2 do Algoritmo 1.4, quando decide-se permanecer na face atual, deve-se utilizar um algoritmo "irrestrito". Em Betra, optou-se por utilizar um método de regiões de confiança com o algoritmo Moré-Sorensen [38] para minimização exata de quadráticas em bolas para resolver cada subproblema. Abaixo temos uma iteração do algoritmo "irrestrito" utilizado em Betra.

Algoritmo 1.5 (Iteração do algoritmo "irrestrito" de Betra)

Seja k a iteração atual. Dados x_k viável, $\Delta_{\min} > 0$, $\Delta_k \geq \Delta_{\min}$ e $\sigma > 0$.

Passo 1. Se $\nabla \bar{f}(x_k) = 0$ então pare e devolva x_k como solução declarando *"Ponto estacionário de primeira ordem"*.

Passo 2. Calcule Δ_{borda} a distância de x_k à borda da face.

Se $\Delta_{\text{borda}} < 2\Delta_{\min}$ então faça uma iteração de SPG restrito a F_I (face atual) para calcular x_{k+1} e pare.

Passo 3. Calcule p_k uma solução aproximada de

$$\begin{array}{ll} \text{Minimizar} & \varphi(w) \equiv \frac{1}{2} w^T \nabla^2 \bar{f}(x_k) w + (\nabla \bar{f}(x_k))^T w \\ \text{sujeita a} & \|w\| \leq \Delta_k. \end{array} \tag{1.9}$$

Passo 4. Se $\varphi(p_k) = 0$ então pare e devolva x_k como solução declarando *"Ponto estacionário de segunda ordem"*.

Passo 5. Se $x_k + p_k$ não pertence à face atual calcule

$$\alpha_{\max} = \max\{\alpha \in [0,1] \mid [x_k, x_k + \alpha p_k] \subset \Omega\}.$$

Se $\bar{f}(x_k + \alpha_{\max} p_k) < \bar{f}(x_k)$ então faça $x_{k+1} = x_k + \alpha_{\max} p_k$ e pare declarando *"Iterando na borda"*. Senão, faça

$$\Delta_k = \Delta_{\min} + 0.9 \left(\frac{\Delta_{\text{borda}}}{1+\sigma} - \Delta_{\min} \right)$$

e volte para o Passo 3.

Passo 6. Calcule $\rho_k = \frac{\bar{f}(x_k + p_k) - \bar{f}(x_k)}{\varphi(p_k)}$.

Passo 7. Se $\rho_k \leq 0.1$ então escolha $\Delta_k = \Delta \in [0.1\|p_k\|, 0.9\|p_k\|]$ e volte para o Passo 2.

Passo 8. Faça $x_{k+1} = x_k + p_k$.

Passo 9. Escolha $\Delta_{k+1} \geq \Delta_{\min}$.

Observações:

- No Passo 1, utiliza-se o critério $\|\nabla \bar{f}(x_k)\|_\infty \leq \varepsilon$.
- Como mencionado anteriormente, no Passo 3, p_k é calculado usando-se o algoritmo Moré-Sorensen para minimização exata de quadrática em bolas (com parâmetro σ).
- No Passo 4, utiliza-se o teste $|\varphi(p_k)| \leq \varepsilon_\varphi$.
- Quando se utiliza o algoritmo Moré-Sorensen para o cálculo de p_k, obtém-se um ponto p_k tal que $\|x_k + p_k\| \leq (1+\sigma)\Delta_k$. Assim, quando Δ_k é recalculado no Passo 5, o próximo ponto $x_k + p_k$ necessariamente pertence ao conjunto viável.
- No Passo 7, calcula-se $\Delta_k = 0.25\|p_k\|$.
- No Passo 9, o cálculo de Δ_{k+1} é feito da seguinte maneira:

$$\Delta = \begin{cases} 0.25\|p_k\|_2, & \text{se } \rho_k \leq 0.25, \\ 2\Delta_k, & \text{se } \rho_k \geq 0.5 \text{ e } |\|p_k\|_2 - \Delta_k| \leq 10^{-5}, \\ \Delta_k, & \text{caso contrário,} \end{cases}$$

$$\Delta_{k+1} = \max\{\Delta_{\min}, \Delta\}.$$

- Define-se Δ_0 como $\Delta_0 = \max\{\Delta_{\min}, 100\max\{1, \|x_0\|\}\}$.

Em GENCAN, optou-se por utilizar um método de Newton truncado com busca linear. Abaixo temos uma Iteração do algoritmo "irrestrito" utilizado em GENCAN.

Algoritmo 1.6 (Iteração do algoritmo "irrestrito" de GENCAN)

Seja k a iteração atual. Dados x_k viável, $\theta \in (0,1)$ e $\gamma \in (0,1)$.

Passo 1. Se $\nabla \bar{f}(x_k) = 0$ então pare e devolva x_k como solução declarando *"Ponto estacionário de primeira ordem"*.

Passo 2. Calcule um vetor não-nulo $p_k \in I\!R^n$ que satisfaça

$$(\nabla \bar{f}(x_k))^T p_k \leq -\theta \|\nabla \bar{f}(x_k)\| \|p_k\|. \tag{1.10}$$

Passo 3. Calcule $\alpha_{\max} = \max\{\alpha \in [0,1] \mid x_k + \alpha p_k \in \Omega\}$.

Se $\alpha_{\max} < 1$ e $\bar{f}(x_k + \alpha_{\max} p_k) < \bar{f}(x_k)$ então faça $x_{k+1} = x_k + \alpha_{\max} p_k$ e pare declarando *"Iterando na borda"*.

Passo 4. Calcule $\alpha_k \leq \alpha_{\max}$ para o qual valha $\bar{f}(x_k + \alpha_k p_k) \leq \bar{f}(x_k) + \gamma \alpha_k (\nabla \bar{f}(x_k))^T p_k$.

Passo 5. Faça $x_{k+1} = x_k + \alpha_k p_k$.

Observações:

- No Passo 1, utiliza-se o critério $\|\nabla f(x_k)\|_\infty \leq \varepsilon$.
- No Passo 2, p_k é calculado utilizando um método de gradientes conjugados para resolver o sistema Newtoniano
$$B_k p_k = -\nabla \bar{f}(x_k),$$
onde B_k é uma aproximação da Hessiana de $\bar{f}$ no ponto x_k. A cada iteração de gradientes conjugados, toma-se o cuidado de satisfazer a condição (1.10). Quando esta não é válida, toma-se o ponto calculado na iteração anterior.
- No Passo 4, utiliza-se "backtracking" com interpolação quadrática uni-dimensional para calcular α_k. A quadrática interpolante é dada por $q(w)$, com $q(0) = \bar{f}(x_k)$, $q(\alpha_k) = \bar{f}(x_k + t_k p_k)$ e $\nabla q(0) = \nabla \bar{f}(x_k)^T p_k$. Definimos $\bar{\alpha}_{\text{novo}}$ como o minimizador de $q(\cdot)$ e utilizamos $\bar{\alpha}_{\text{novo}}$ para calcular α_k.

  Passo 4.1. Faça $\alpha_{\text{tent}} \leftarrow \alpha_{\max}$.

  Passo 4.2. Se
  $$\bar{f}(x_k + \alpha_{\text{tent}} p_k) \leq \bar{f}(x_k) + \gamma \alpha_{\text{tent}} \nabla \bar{f}(x_k)^T p_k$$
  então faça $\alpha_k = \alpha_{\text{tent}}$ e pare.

  Passo 4.3. Calcule
  $$\bar{\alpha}_{\text{novo}} = -\frac{\alpha_{\text{tent}}^2 \nabla \bar{f}(x_k)^T p_k}{2(\bar{f}(x_k + \alpha_{\text{tent}} p_k) - \bar{f}(x_k) - \alpha_{\text{tent}} \nabla \bar{f}(x_k)^T p_k)}.$$
  Se $\bar{\alpha}_{\text{novo}} \geq 0.1\alpha_{\text{tent}}$ e $\bar{\alpha}_{\text{novo}} \leq 0.9\alpha_{\text{tent}}$ então faça $\alpha_{\text{novo}} \leftarrow \bar{\alpha}_{\text{novo}}$. Senão, faça $\alpha_{\text{novo}} \leftarrow 0.5\ \alpha_{\text{tent}}$.
  Faça $\alpha_{\text{tent}} \leftarrow \alpha_{\text{novo}}$ e volte para o Passo 4.2.

Tanto BETRA como GENCAN, depois de calcular x_{k+1}, utilizam a técnica de extrapolação para calcular um ponto com valor de função melhor do que x_{k+1} e, possivelmente, com mais restrições ativas. O algoritmo de extrapolação é como segue:

Algoritmo 1.7 (Extrapolação)

Dados x_k viável, p_k direção de descida, $\mu \in (0, 1]$ tal que $x_{k+1} = x_k + \mu p_k$ e $N > 1$.

Passo 1. *Decisão de extrapolar ou não*

Se $p_k^T \nabla \bar{f}(x_k + p_k) \geq 0.5 p_k^T \nabla \bar{f}(x_k)$ então pare.

Passo 2. Calcule $\mu_{\max} = \max\{t \geq 0 \mid [x_k, x_k + tp_k] \subset \Omega\}$.

Passo 3. *Cálculo do novo escalar que multiplica p_k*

Se $\mu < \mu_{\max}$ e $N\mu > \mu_{\max}$ então faça $\mu_{\text{tent}} \leftarrow \mu_{\max}$. Senão, faça $\mu_{\text{tent}} \leftarrow N\,\mu$.

Passo 4. *Continua extrapolação enquanto obtém decréscimo simples da função*

Se

$$\bar{f}(P_\Omega(x_k + \mu p_k)) \leq \bar{f}(P_\Omega(x_k + \mu_{\text{tent}} p_k))$$

então pare e devolva $P_\Omega(x_k + \mu p_k)$. Senão, faça $\mu \leftarrow \mu_{\text{tent}}$ e volte para o Passo 3.

Para problemas de pequeno porte, BETRA apresenta melhores resultados (como mostrado em [3]). Daí a idéia de incluí-lo como alternativa ao algoritmo interno de ALGENCAN. Para problemas muito esparsos, uma versão esparsa de BETRA deve funcionar melhor do que GENCAN. Por isso implementamos uma versão esparsa de BETRA (BETRA-ESPARSO) e também a incluímos como algoritmo interno de ALGENCAN. Na Seção 1.1.1 apresentamos alguns detalhes de sua implementação.

1.1.1 Versão esparsa de Betra

Para a implementação de BETRA-ESPARSO, armazenamos a matriz Hessiana H de $\bar{f}$ no formato de coordenadas (veja [23], por exemplo). Neste formato, o número de elementos não-nulos de H é dado por *hnnz* e são necessários três vetores (*hrow*, *hcol* e *hval*) de tamanho *hnnz*. O elemento H_{ij} é armazenado em uma posição k, tal que $hrow(k) = i$, $hcol(k) = j$ e $hval(k) = H_{ij}$. Como esta matriz é simétrica, apenas a parte triangular inferior deve ser armazenada.

Note que, no Algoritmo 1.5, a matriz Hessiana H é usada apenas no cálculo do passo p_k e para o cálculo do modelo quadrático φ. Para o caso esparso, o valor de φ é obtido de maneira análoga ao caso denso, trocando-se apenas o produto matriz-vetor pelo produto esparso. Já no caso do cálculo do passo p_k, muitas outras mudanças são necessárias. A seguir apresentamos o algoritmo Moré-Sorensen para minimização de quadráticas em bolas, usado para calcular p_k. No caso de BETRA, a matriz de entrada H é a Hessiana de $\bar{f}$ em um ponto x_k, o vetor g é o gradiente de $\bar{f}$ em x_k, Δ é o raio da região de confiança na iteração k, λ^0 é um escalar que vale 0 na primeira iteração de regiões de confiança e é reaproveitado a cada iteração do algoritmo Moré-Sorensen, $\sigma_1 = \sigma$ e $\sigma_2 = 0$.

Algoritmo 1.8 (Algoritmo Moré-Sorensen)

Dados uma matriz simétrica H, um vetor g, $\Delta > 0$, $\lambda^0 \geq 0$, $\sigma_1, \sigma_2 \in (0,1)$. Faça $k \leftarrow 0$.

Passo 1. *Valores iniciais de λ_L, λ_U, λ_S e k*

Faça $\lambda_S \leftarrow \max_{1\leq i\leq n}\{-H_{ii}\}$, $\lambda_L \leftarrow \max\{0, \lambda_S, \frac{\|g\|}{\Delta} - \|H\|_1\}$, $\lambda_U \leftarrow \frac{\|g\|}{\Delta} + \|H\|_1$.

Passo 2. *Salvaguarda de λ^k*

Faça $\lambda^k = \max\{\lambda^k, \lambda_L\}$ e $\lambda^k = \min\{\lambda^k, \lambda_U\}$.

Se $\lambda^k \leq \lambda_S$ então $\lambda^k = \max\{0.001\lambda_U, \sqrt{\lambda_L\lambda_U}\}$.

Passo 3. *Tentativa de fatoração de $H + \lambda^k I$ (I matriz identidade)*

Use Cholesky para fatorar $H + \lambda^k I = R^T R$.

Se a fatoração não foi possível então compute δ e $\|u\|^2$ (como descrito ao final deste algoritmo). Faça $\lambda_S \leftarrow \max\left\{\lambda_S, \lambda^k + \frac{\delta}{\|u\|^2}\right\}$ e vá para o Passo 6.

Passo 4. Resolva $R^T R p_k = -g$ em p_k.

Passo 5. *Cálculo da aproximação de z*

Se $\|p_k\| < \Delta$ então calcule $\hat{z}$ usando a técnica de Cline, Moler, Stewart e Wilkinson [17] e faça

$$\tau = -p_k^T\hat{z} + \operatorname{sgn}(p_k^T\hat{z})\sqrt{((p_k^T\hat{z})^2 - (\|p_k\|^2 - \Delta^2))}.$$

Passo 6. *Atualização dos valores de λ_L, λ_U, λ_S*

Se $\lambda^k \in (-\lambda_1, \infty)$ e $\phi(\lambda^k) < 0$ então faça $\lambda_U \leftarrow \min\{\lambda_U, \lambda^k\}$ e $\lambda_S \leftarrow \max\{\lambda_S, \lambda^k - \|R\hat{z}\|^2\}$. Senão, faça $\lambda_L \leftarrow \max\{\lambda_L, \lambda_S, \lambda^k\}$.

Passo 7. *Critérios de convergência*

Se $\|p_k\| \leq \Delta$ e $\lambda^k = 0$ então pare e devolva p_k como solução.

Considere as desigualdades

$$| \Delta - \|p_k\| | \leq \sigma_1\Delta \text{ e} \tag{1.11}$$

$$\|R(\tau\hat{z})\|^2 \leq \sigma_1(2-\sigma_1)\max\{\sigma_2, \|Rp_k\|^2 + \lambda^k\Delta^2\}. \tag{1.12}$$

Passo 7.4 *Decisão do critério a ser aceito*

Se **apenas** a desigualdade (1.11) é satisfeita então pare e devolva p_k como solução.

Se **apenas** a desigualdade (1.12) é satisfeita então pare e devolva $p_k + \tau\hat{z}$ como solução.

Se as desigualdades (1.11) e (1.12) são satisfeitas então: se $\|R(\tau\hat{z})\|^2 \leq \lambda^k(\Delta^2 - \|p_k\|^2)$, pare e devolva $p_k + \tau\hat{z}$ como solução; senão, pare e devolva p_k como solução.

Passo 8. *Atualização de λ^k e k*

Se foi possível a fatoração de Cholesky e $g \neq 0$ então resolva $R^T t_k = p_k$ em t_k e faça

$$\lambda^{k+1} = \lambda^k + \left(\frac{\|p_k\|}{\|t_k\|}\right)^2\left(\frac{\|p_k\| - \Delta}{\Delta}\right).$$

Senão, faça $\lambda^{k+1} = \lambda_S$.

Faça $k \leftarrow k + 1$ e volte para o Passo 2.

No Passo 1, temos de calcular $\|H\|_1$, o que é simples de ser feito com a representação esparsa de H.

No Passo 3, quando não é possível completar a fatoração de Cholesky, pode-se calcular $\delta \geq 0$ de forma que

$$H + \lambda^k I + \delta e_l e_l^T$$

seja singular e l a posição da matriz $H + \lambda^k I$ onde a decomposição falhou. Ainda é possível determinar o vetor $u \in I\!R^l$ tal que

$$(H^l + \lambda^k I + \delta e_l e_l^T)u = 0,$$

com H^l a submatriz com as primeiras l linhas e colunas de H e $u_l = 1$. Para isso, basta calcular a solução do subsistema $\bar{R}^T \bar{R}\bar{u} = 0$, onde $\bar{R}$ é a matriz de tamanho $(l-1) \times (l-1)$ obtida pela fatoração de Cholesky incompleta e $\bar{u} \in I\!R^{(l-1)}$.

No Passo 3, é necessário calcular a fatoração de Cholesky de H. Para isso utilizamos a rotina MA27 da coleção HSL [24]. Esta rotina, no entanto, não calcula $H + \lambda I = R^T R$, com R matriz triangular superior, como é necessário para o Algoritmo 1.8, mas sim $P^T(H + \lambda I)P = LDL^T$, com L matriz triangular inferior, D matriz diagonal e P matriz de permutação. Baseando-nos nas subrotinas da MA27 que resolvem os sistemas triangulares, conseguimos extrair a matriz diagonal D e as linhas da matriz triangular L, que serão necessárias nos passos seguintes.

Ainda no Passo 3, precisamos utilizar a fatoração de Cholesky para calcular δ e u. Acrescentamos uma linha ao código da rotina de MA27 que calcula a fatoração de Cholesky para que a posição onde fica armazenado o valor de δ ficasse disponível. Já no caso do cálculo de u, como não dispomos da submatriz resultante da fatoração incompleta, criamos uma matriz esparsa $\bar{H}$ com as primeiras $l-1$ linhas e colunas de H, onde a fatoração é possível, e resolvemos o sistema linear $(\bar{H} + \lambda^k I)\bar{u} = 0$ usando as subrotinas de MA27.

No Passo 4, é preciso resolver um sistema linear $R^T R p_k = -g$ em p_k. Para isso usamos a subrotina de MA27 para resolver o sistema linear, dada a fatoração computada. Tanto no Passo 6 como no Passo 7, precisamos calcular $\|R\hat{z}\|^2$. Como estamos trabalhando com norma euclidiana, temos que $\|R\hat{z}\|^2 = (R\hat{z})^T(R\hat{z}) = \hat{z}^T(H + \lambda I)\hat{z}$. Este produto é calculado com a matriz esparsa H, para que não se tenha de recuperar toda sua fatoração de Cholesky. No Passo 7 ainda é necessário calcular $\|Rp_k\|^2$, o que é feito de maneira análoga.

No Passo 8, é preciso resolver o sistema $R^T t_k = p_k$ em t_k. Lembre-se que a fatoração fornecida pela MA27 é $P^T(H + \lambda I)P = LDL^T$. Ou seja, $H + \lambda I = PLDL^T P^T$. Como MA27 dispõe de uma subrotina para resolver sistemas do tipo $PLP^T x = b$, calculamos a solução de $PLP^T \bar{y} = p_k$. De posse de $\bar{y}$, calculamos $y = P^T \bar{y}$ e, finalmente, $t_k = D^{-1/2} y$.

No Passo 5, calculamos $\hat{z}$ usando a técnica de Cline, Moler, Stewart e Wilkinson apresentada em [17]. Nesta técnica, um vetor e com componentes 1 ou -1 é escolhido de maneira que a solução z do sistema $R^T z = e$ seja grande. A idéia é escolher o sinal das componentes de e

de maneira a obter máximo crescimento local de w durante a resolução do sistema. Depois, resolve-se o sistema $Rv = z$ para v. Definimos então $\hat{z} = \frac{v}{\|v\|}$. O algoritmo para o cálculo de $\hat{z}$ é o seguinte:

Algoritmo 1.9 (Algoritmo de Cline, Moler, Stewart e Wilkinson)

Dada uma matriz triangular superior R que seja fator de Cholesky uma matriz simétrica definida positiva.

Passo 1. Faça $e \leftarrow 1$ e $z \leftarrow 0$.

Passo 2. Para k de 1 a n

Passo 2.1. Se $|z_k| \neq 0$ então faça $e_k \leftarrow \text{sgn}(-z_k)$.

Passo 2.2. Faça $s \leftarrow \min\{1, |R_{kk}|/|e_k - z_k|\}$, $z \leftarrow s\ z$, $e_k \leftarrow s\ e_k$, $w_k \leftarrow e_k - z_k$, $w_{km} \leftarrow -e_k - z_k$, $s \leftarrow |w_k|$, $sm \leftarrow |w_{km}|$.

Passo 2.3. Se $R_{kk} = 0$ então faça $w_k \leftarrow 1$ e $w_{km} \leftarrow 1$. Senão, faça $w_k \leftarrow \frac{w_k}{R_{kk}}$ e $w_{km} \leftarrow \frac{w_{km}}{R_{kk}}$.

Passo 2.4. Se $k < n$ então faça $sm \leftarrow sm + \sum_{i=k+1}^{n} |z_i + w_{km} R_{ki}|$, $z_i \leftarrow z_i + \sum_{i=k+1}^{n} w_k R_{ki}$ e $s \leftarrow s + \sum_{i=k+1}^{n} |z_i|$. Se $s < sm$ então faça $w \leftarrow w_{km} - w_k$, $w_k \leftarrow w_{km}$ e $z_i = z_i + \sum_{i=k+1}^{n} w R_{ki}$.

Passo 2.5. Faça $z_k \leftarrow w_k$.

Passo 3. Resolva $Rv = z$.

Passo 4. Faça $\hat{z} = \frac{v}{\|v\|}$.

Para implementar a versão esparsa do Algoritmo 1.9, tivemos de fazer várias modificações. Primeiramente, como temos $H + \lambda I = PLDL^T P^T$, utilizamos a matriz $D^{1/2} L^T P^T$ no lugar de R. Como não queremos montar toda a matriz, a cada iteração k, extraímos uma linha de $D^{1/2} L^T P^T$ e a representamos por um vetor esparso.

No Passo 3 do Algoritmo 1.9, para resolver $Rv = z$, devemos resolver $D^{1/2} L^T P^T v = z$. Como a subrotina de MA27 que resolve o sistema triangular superior resolve sistemas do tipo $PDL^T P^T x = b$, temos de calcular $y = D^{1/2} z$. Em seguida, calculamos $\bar{y} = Py$ e, por fim, resolvemos o sistema $PDL^T P^T v = \bar{y}$.

Outro ponto importante é que, neste algoritmo, temos somatórios envolvendo até n elementos dentro de um laço de tamanho n. Para o caso esparso, isso não deve ser feito, pois teríamos um algoritmo com iterações da ordem de $O(n^2)$, no lugar de $O(hnnz)$. Para que isto fosse evitado, calculamos o valor inicial s da soma dos valores absolutos dos elementos de z. Cada vez que s ou z deve ser alterado, seu valor é modificado em $O(1)$ passos. Assim, temos um total de n iterações e cada iteração k calcula um somatório do tamanho do número de elementos não-nulos da linha k da fatoração. Ou seja, cada iteração do algoritmo gasta $O(hnnz)$ operações.

1.1.2 Escolha automática do algoritmo interno

Vejamos agora como funciona a escolha automática para o algoritmo interno de Algencan. Segundo justificativas teóricas e resultados empíricos, Betra geralmente é mais adequado para problemas pequenos, Betra-esparso para problemas maiores e muito esparsos e Gencan para o restante. Com base nisso, criamos as seguintes regras para decidir qual algoritmo interno usar em cada iteração de Lagrangiano aumentado para cada problema:

- Se o número de variáveis n for menor do que 150, Betra será utilizado. Esta escolha é feita pelo fato de Betra ser mais adequado para problemas de pequeno porte.

- Se foi usado Gencan como algoritmo interno na iteração anterior de Lagrangiano aumentado e o Gencan parou por chegar a um tamanho de passo muito pequeno ($x_{k+1} \approx x_k$), Betra-esparso será escolhido.

- Se foi usado Gencan como algoritmo interno na iteração anterior de Lagrangiano aumentado e foram necessárias mais de $0.6n$ iterações de gradientes conjugados para calcular o passo p_k, Betra-esparso será utilizado. Cada iteração de gradientes conjudados gasta $O(n^2)$ operações. Ou seja, se são usadas aproximadamente n iterações, temos um método da ordem de $O(n^3)$. Esta é a complexidade da fatoração de Cholesky. Portanto, a idéia por trás deste critério é que, se Gencan está usando um método que gasta $O(n^3)$ operações para calcular o passo p_k, pode ser interessante utilizar Betra-esparso em seu lugar.

- Se Betra-esparso foi utilizado como algoritmo interno na iteração anterior de Lagrangiano aumentado e Betra-esparso parou por chegar a um tamanho de passo muito pequeno ($x_{k+1} \approx x_k$) ou por obter um raio de região de confiança muito pequeno ($\Delta_k < \varepsilon_{\text{mach}} \max(1, \|x_k\|)$), Gencan será usado.

- Se Betra-esparso foi usado como algoritmo interno na iteração anterior de Lagrangiano aumentado e foram necessárias mais de 7 iterações de regiões de confiança para calcular o ponto x_{k+1}, Gencan será utilizado. Se são necessárias muitas iterações de regiões de confiança para obter decréscimo suficiente da função, o modelo quadrático pode não ser uma boa aproximação para a função. Por isso, neste caso, mudamos o algoritmo interno para Gencan.

- Se foi usado Betra-esparso como algoritmo interno na iteração anterior de Lagrangiano aumentado e foram necessárias, em média, pelo menos 10 fatorações de Cholesky por iteração de região de confiança para calcular o passo p_k, Gencan será escolhido. Espera-se que Betra-esparso use, em média, em torno de 3 fatorações de Cholesky por iteração de regiões de confiança. Se são necessárias muito mais fatorações, pode ser interessante trocar o método direto para calcular p_k e passar a utilizar gradientes conjugados.

Há mais um critério para a escolha do algoritmo interno. Se nenhuma das condições anteriores é satisfeita, calculamos um limitante superior *est* para o número de elementos não-nulos da Hessiana de $\bar{f}$. Lembre-se que $\bar{f}$ aqui é uma função Lagrangiano aumentado com ρ_k,

λ_k e μ_k fixos

$$L_{\rho_k}(x, \lambda_k, \mu_k) = f(x) + \frac{\rho_k}{2}\Big\{ \sum_{i=1}^{m} \Big[h_i(x) + \frac{\lambda_{ki}}{\rho_k} \Big]^2 + \sum_{i=1}^{p} \Big[\max\Big(0, g_i(x) + \frac{\mu_{ki}}{\rho_k}\Big) \Big]^2 \Big\}.$$

Em um ponto x_k, apenas as restrições de igualdade h e as restrições de desigualdade g que não são satisfeitas ou as que são satisfeitas com uma "margem pequena" (ou seja, $g_i(x) \geq -\frac{\mu_{ki}}{\rho}$), contribuem para o valor de $L_{\rho_k}(x_k, \lambda_k, \mu_k)$. Seja J o conjunto dos índices das restrições de desigualdade que contribuem para o valor da função Lagrangiano aumentado. Então, a segunda derivada de L_{ρ_k} com relação a x no ponto x_k é dada por

$$\nabla^2_{xx} L_{\rho_k}(x_k, \lambda_k, \mu_k) = \nabla^2 f(x_k) + \sum_{i=1}^{m} [(\rho_k h_i(x_k) + \lambda_{ki}) \nabla^2 h_i(x_k)] + \sum_{i \in J} [(\rho_k g_i(x_k) + \lambda_{ki}) \nabla^2 g_i(x_k)] +$$

$$\sum_{i=1}^{m} [\rho_k \nabla h_i(x_k) (\nabla h_i(x_k))^T] + \sum_{i \in J} [\rho_k \nabla g_i(x_k) (\nabla g_i(x_k))^T]$$

Assim, calculamos a Hessiana da função Lagrangiano (definindo *hlnnz* como seu número de elementos não-nulos na parte triangular) e estimamos o número de elementos não-nulos na parte triangular dos dois últimos termos da soma acima.

Defina nc_i o número de elementos não-nulos do gradiente cada restrição de igualdade e nc_j o número de elementos não-nulos do gradiente cada restrição $j \in J$ que contribui no cálculo da Hessiana do Lagrangiano aumentado. Tome $up_1 = \sum_{i=1}^{m} \frac{nc_i^2 + nc_i}{2} + \sum_j \frac{nc_j^2 + nc_j}{2}$. Defina nc como o número de variáveis que aparece pelo menos uma vez em algum dos gradientes das restrições de igualdade ou das $j \in J$ restrições de desigualdade. Tome $up_2 = \frac{nc^2 + nc}{2}$. Note que $hlnnz + \min\{up_1 + nc, up_2\}$ é um limitante superior para o número de elementos não-nulos na parte triangular da Hessiana do Lagrangiano aumentado. Definimos, então, $est = \min\{hlnnz + \min\{up_1 + nc, up_2\}, \frac{n^2+n}{2}\}$. Se o valor de est for menor ou igual a $20n$, consideramos que a matriz é suficientemente esparsa e Betra-esparso é escolhido como algoritmo interno. Caso contrário, utilizamos Gencan.

1.2 Detalhes de implementação

Utilizamos a implementação Betra, em Fortran 77, com algumas modificações para melhorar seu desempenho. Determinamos um número máximo de 20 iterações para o algoritmo Moré-Sorensen (Passo 3 do Algoritmo 1.5). Quando este número de iterações é atingido e o Algoritmo 1.5 não encontrou a solução para o subproblema de regiões de confiança, utilizamos uma iteração de dogleg para calcular a direção de descida (veja [42], por exemplo). Nesta iteração de dogleg, em vez de resolver o sistema de Newton, aproveitamos o passo $\bar{p}_k$ calculado pelo Algoritmo 1.5. Quando este foi calculado, o passo $\bar{p}_k$ é combinado com a direção de Cauchy para obter uma direção de descida dentro da região de confiança. Quando $\bar{p}_k$ não está disponível, a direção calculada por dogleg será a própria direção de Cauchy.

Outra mudança feita em BETRA está relacionada com a atualização do raio Δ_k da região de confiança. Sempre que, no Passo 5 do Algoritmo 1.5, é necessário diminuir Δ_k para que o ponto $x_k + p_k$ pertença a face atual, armazenamos Δ_{ant}, o valor de Δ_k antes desta mudança de valor. Assim, quando uma nova iteração de regiões de confiança se inicia, verificamos se o ponto x_k está em uma face diferente da face a qual pertencia x_{k-1}. Se a face é diferente, fazemos $\Delta_k = \Delta_{\text{ant}}$. Note que, quando diminuimos o raio da região de confiança para que um ponto pertença a face atual, não estamos levando em consideração o ajuste do modelo quadrático à função. Assim, se na iteração seguinte estamos em uma nova face, não queremos que o esta diminuição do raio afete o cálculo do próximo ponto. Daí a idéia desta mudança.

Outra mudança na atualização de Δ_k foi feita depois da realização de uma extrapolação (Algoritmo 1.7). Neste caso, definimos $\Delta_{k+1} = 2\max(\Delta_k, \mu\|p_k\|)$, onde μ é o tamanho de passo dado na direção p_k na extrapolação. A motivação para esta mudança é a seguinte: se foi possível obter decréscimo da função usando tamanho de passo μ, podemos confiar no modelo quadrático como boa aproximação para a função objetivo e aumentar o raio da região de confiança.

Alguns parâmetros de BETRA foram modificados (em relação aos descritos em [3]). São eles: $\Delta_{\min} = 10^{-8}$, $\sigma = 0.1$ (parâmetro σ_1 de Algoritmo 1.8). A extrapolação passou a ser feita depois da execução dos Passos 2, 5 e 8 do Algoritmo 1.5, usando $N = 2$. Todas estas mudanças trouxeram uma grande melhora na eficiência de BETRA.

Implementamos BETRA-ESPARSO em Fortran 77, como descrito na Seção 1.1.1, utilizando todas as modificações mencionadas acima. Utilizamos a rotina MA27 para calcular a fatoração de Cholesky e resolver os sistemas lineares esparsos necessários. Além disso, fizemos com que a fatoração fosse interrompida assim que fosse detectado que a matriz a ser fatorada é singular ou não é definida positiva[1].

Implementamos a versão de ALGENCAN que utiliza BETRA ou BETRA-ESPARSO como subalgoritmos, usando os parâmetros padrão de ALGENCAN. Implementamos também a versão que faz a escolha automática do algoritmo interno, seguindo os critérios apresentados na Seção 1.1.2. Note que a diferença entre BETRA, BETRA-ESPARSO e GENCAN está no algoritmo "irrestrito" usado no Passo 4 do Algoritmo 1.4. Mantivemos, então, o nome ALGENCAN para as 4 versões. Chamaremos de ALGENCAN-G a versão original de ALGENCAN, ALGENCAN-B a versão que usa BETRA, ALGENCAN-BS a versão que usa BETRA-ESPARSO e ALGENCAN-A a versão que usa a escolha automática do algoritmo interno.

1.3 Resultados numéricos

Para comparar o desempenho das diferentes versões de ALGENCAN, utilizamos duas coleções de problemas. Uma delas é a coleção LA CUMPARSITA [56], que é parte do projeto TANGO [55], do qual ALGENCAN-G faz parte. Escolhemos três problemas desta coleção para testar as 4 diferentes versões de ALGENCAN: Ellipsoid, Hard-spheres e Mountain. Estes problemas, além de serem conhecidos e possuírem características interessantes, nos convêm para a análise

[1] Isso é feito definindo $cntl(1) = -1$ como parâmetro da rotina MA27ID.

comparativa entre os métodos por conhecermos suas estruturas.

A outra coleção de problemas é CUTEr (Constrained and Unconstrained Testing Environment, revisited) [15]. Utilizamos todos os problemas que possuem segunda derivada e pelo menos uma restrição que não seja de caixa e os dividimos em dois grupos: problemas com até 150 variáveis, para compararmos o desempenho de ALGENCAN-B com ALGENCAN-G; e problemas com mais de 150 variáveis, para compararmos o desempenho de ALGENCAN-A e ALGENCAN-G. Temos, então, um total de 739 problemas, sendo 366 do primeiro grupo e 363 do segundo grupo.

Para verificar o desempenho das diferentes versões de ALGENCAN, as comparamos com LANCELOT B [18, 19, 20], um método de Lagrangiano aumentado para resolver (1.3) conhecido na literatura. Utilizamos os mesmos problemas da coleção CUTEr mencionados acima, divididos da mesma maneira.

Os critérios de convergência de todas as versões de ALGENCAN são $\|g_P(x_k)\|_\infty \leq \varepsilon_1$ e $\max\{\|h_1(x_k)\|_\infty, \|\sigma_k\|_\infty\} \leq \varepsilon_2$, onde $g_P(x_k) = P_{\Omega_2}(x_k - \nabla L_{\rho_k}(x_k, \lambda_k, \mu_k)) - x_k$ e $[\sigma(x, \mu, \rho)]_i = \max\{[g_1(x)]_i, \mu_i/\rho\}$, como na versão original. Utilizamos $\varepsilon_1 = \varepsilon_2 = 10^{-8}$. Para os critérios de convergência dos algoritmos internos das diferentes versões de ALGENCAN, utilizamos $\varepsilon = 10^{-8}$ e $\varepsilon_\varphi = 10^{-5}$, ou seja, $\|\nabla \bar{f}(x_k)\|_\infty \leq 10^{-8}$ e $|\varphi(p_k)| \leq 10^{-5}$. Para o algoritmo "irrestrito" de GENCAN (Algoritmo 1.6) usamos ainda $\gamma = 10^{-4}$ e $\theta = 10^{-6}$.

Todos os experimentos foram feitos utilizando o compilador g77 - GNU project Fortran Compiler, com opção de compilação -O3 para otimizar o código, usando um AMD Athlon 64 K8 3200, com 1 Gb de memória RAM, com sistema operacional Linux.

Daqui em diante, diremos que dois valores de função f_1 e f_2 são equivalentes se

$$|f_1 - f_2| \leq \max\{10^{-10}, 10^{-6} \min\{|f_1|, |f_2|\}\} \text{ ou } (f_1 \leq -10^{20} \text{ e } f_2 \leq -10^{20}). \tag{1.13}$$

O tempo gasto pelos métodos para resolver alguns problemas é muito próximo de zero. Assim, sempre que utilizamos métodos implementados por nós ou algum método que pode ser usado como chamada a uma rotina, executamos cada par problema/método nv vezes, com um mesmo ponto inicial, medindo o tempo gasto para terminar as nv execuções. Calculamos nv tal que o tempo total gasto seja de, pelo menos, 1 segundo. Consideramos, daqui em diante, a média de tempo gasto nas execuções do par problema/método como o tempo gasto por cada método para resolver cada problema. Se o método não pode ser executado várias vezes, não confiamos em medições de tempo menores do que 0.01 segundo.

Na Seção 1.3.1 apresentamos os resultados obtidos pelas 4 versões de ALGENCAN nos problemas da coleção LA CUMPARSITA. Na Seção 1.3.2 apresentamos os resultados obtidos na aplicação das diferentes versões de ALGENCAN aos problemas da coleção CUTEr. Apresentamos também a comparação de ALGENCAN-G e ALGENCAN-A com LANCELOT B aplicados aos problemas de CUTEr.

1.3.1 Comparação entre versões do Algencan usando La Cumparsita

Há algumas expectativas em relação ao comportamento das diferentes versões de Algencan. Espera-se que utilizar Betra como algoritmo interno seja a melhor alternativa para problemas com número pequeno de variáveis. Espera-se que utilizar Betra-esparso seja melhor para problemas com Hessiana da função Lagrangiano aumentado (e sua fatoração) muito esparsa. Espera-se que Gencan seja o melhor algoritmo interno para os demais casos. Espera-se ainda que a escolha automática do algoritmo "irrestrito" do Passo 4 do Algoritmo 1.4 obtenha resultados, no pior caso, entre o melhor e o pior que podem ser obtidos escolhendo qualquer outro dos algoritmos internos, podendo obter resultados ainda melhores do que qualquer um dos métodos anteriores usados separadamente.

Como temos informação sobre a estrutura dos problemas da coleção La Cumparsita, utilizaremos aqui três problemas desta coleção para verificar se todas as expectativas descritas acima são confirmadas: Ellipsoid, Hard-spheres e Mountain.

1.3.1.1 Ellipsoid [53]

O problema Ellipsoid consiste em, dado um número np de pontos em $\mathbb{R}^{nd}$, minimizar o volume do elipsóide, centrado na origem, que contém todos os pontos p_i.

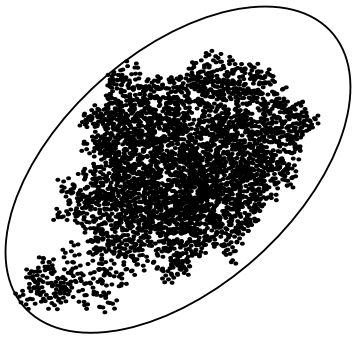


Figura 1.1: Representação do problema Ellipsoid.

O modelo para este problema pode ser escrito como

$$\begin{array}{lll} \text{Minimizar} & -\sum_{i=1}^{nd} log(l_{ii}) & \\ \text{sujeita a} & p_i^T L L^T p_i \leq 1, & i = 1, ..., np, \\ & l_{ii} \geq \varepsilon, & i = 1, ..., nd, \end{array} \tag{1.14}$$

onde $p_i \in \mathbb{R}^{nd}$, L é uma matriz triangular inferior $nd \times nd$ e $\varepsilon > 0$.

O número de variáveis deste problema é $n = nd(nd+1)/2$. O número de restrições é $m = np$, além das nd restrições de caixa. Neste exemplo, os np pontos p_i são gerados usando a distribuição de Cauchy. O valor de ε é 10^{-16}.

Instâncias interessantes do problema Ellipsoid são aquelas com $nd = 2, 3, 4$ e 10, que possuem um número pequeno de variáveis. Por este motivo, geramos instâncias com até 100

variáveis e as utilizamos para verificar o comportamento de Algencan-B e Algencan-G, pois estas são as alternativas mais adequadas para problemas de pequeno porte. Como estamos tratando de problemas de pequeno porte, esperamos que o desempenho de Algencan-B seja melhor do que o desempenho de Algencan-G.

Definimos 6 números diferentes de variáveis n. Para cada n, definimos 10 números diferentes de restrições m. Para cada par (n, m) geramos 10 problemas Ellipsoid com pontos iniciais diferentes e distribuição dos pontos no espaço diferentes, totalizando 600 problemas com menos de 100 variáveis.

Para analisar o desempenho dos dois métodos, consideramos cada conjunto de 10 problemas com mesmo par (n, m). Dentre cada um destes conjuntos, eliminamos aqueles para os quais algum dos métodos não satisfez os critérios de convergência ou ambos obtiveram valores de função diferentes (usando o critério (1.13)). Considerando os problemas restantes, calculamos a média aritmética do tempo e do número de avaliações de função gasto por cada método para resolver o problema. Na Tabela 1.1 temos o número de problemas de cada conjunto para os quais ambos os métodos satisfizeram os critérios de convergência e obtiveram mesmo valor de função (coluna n_{prob}), média de tempo e média do número de avaliações de função (nef) usadas para resolver os n_{prob} problemas de cada conjunto.

Na maioria dos casos, os métodos terminaram satisfazendo os critérios de convergência. Em alguns casos, os métodos param por atingir o número máximo de 50 iterações ou por atingir o limite de tempo de 1000 segundos. Algencan-B pára por atingir o número máximo de iterações (obtendo ponto final viável) em:

- 3 instâncias com $n = 66$, sendo uma com $m = 3000$, uma com $m = 4000$ e uma com $m = 5000$;
- uma instância com $n = 91$ e $m = 3000$.

Algencan-G pára por atingir o número máximo de iterações (obtendo ponto final viável) em:

- uma instância com $n = 45$ e $m = 5000$;
- 5 instâncias com $n = 66$, sendo uma com $m = 3000$, uma com $m = 4000$, uma com $m = 7000$, uma com $m = 8000$ e uma com $m = 10000$;
- 11 instâncias com $n = 91$, sendo uma instância com $m = 2000$, uma com $m = 3000$, uma com $m = 4000$, uma com $m = 8000$, três com $m = 9000$ e 4 com $m = 10000$.

Algencan-G pára por atingir o limite de tempo de 1000 segundos (obtendo ponto viável) em:

- uma instância com $n = 15$ e $m = 8000$;
- uma instância com $n = 28$ e $m = 4000$;
- uma instância com $n = 66$ e $m = 5000$.

ALGENCAN-G pára por atingir o limite de tempo de 1000 segundos (parando em ponto inviável) em:

- uma instância com $n = 28$ e $m = 8000$;
- uma instância com $n = 45$ e $m = 6000$;
- 2 instâncias com $n = 66$, sendo uma com $m = 9000$ e uma com $m = 10000$.

Em todos os casos em que ambos os métodos satisfizeram os critérios de convergência, o valor de função obtido foi equivalente (usando o critério (1.13)).

Em termos de robustez, ALGENCAN-B se mostrou um pouco melhor do que ALGENCAN-G, já que ALGENCAN-B satisfez seus critérios de convergência mais vezes e sempre obteve ponto final viável. Em termos de eficiência, usar ALGENCAN-B se mostrou a melhor alternativa. Em todas as instâncias do problema, ALGENCAN-B chegou mais rapidamente a solução. Isso era esperado, já que estamos lidando aqui com problemas pequenos. Ou seja, ALGENCAN-B é mais robusto e eficiente do que ALGENCAN-G, como era esperado.

1.3.1.2 Hard-spheres [21, 32, 36]

O problema Hard-spheres faz parte de uma família de problemas de empacotamento de esferas. Estes problemas datam do século XVII e já foram tratados por vários matemáticos conhecidos, como Newton e Hilbert. No entanto, várias instâncias deste problema permanecem em aberto. O problema Hard-spheres consiste em maximizar a menor distância entre np pontos distribuídos em uma esfera de raio unitário em $\mathbb{R}^{nd}$. Isto pode ser modelado da seguinte forma:

$$\begin{array}{lll} \text{Minimizar} & z & \\ \text{sujeita a} & < x_i, x_j > \leq z, & i, j = 1, ..., np,\ i \neq j, \\ & \|x_i\|^2 = 1, & i = 1, ..., np, \end{array} \tag{1.15}$$

onde $z \in \mathbb{R}$ e $x_i \in \mathbb{R}^{nd}$.

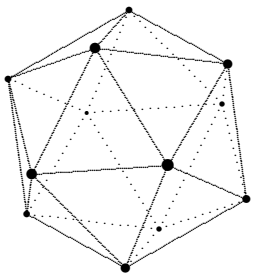


Figura 1.2: Representação do problema Hard-spheres.

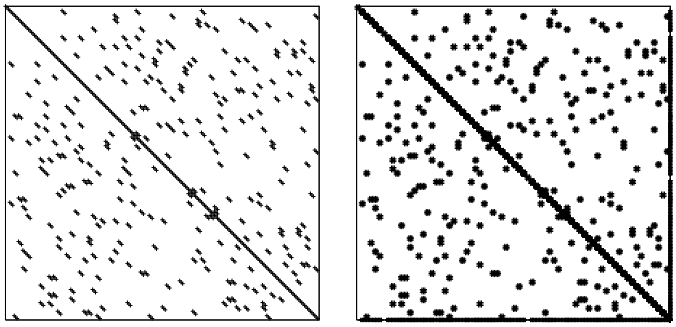
Nesta formulação, se encontramos um valor menor ou igual a 0.5 para a função objetivo, significa que os np pontos pertencem à esfera e a distância entre cada par de pontos é pelo menos 1. Ou seja, temos uma solução para o problema *Kissing* (veja [21]). Este problema tem $n = nd \times np + 1$ variáveis e $m = np + np(np - 1)/2$ restrições. Na implementação

			ALGENCAN-B		ALGENCAN-G	
n	m	n_{prob}	nef	Tempo	nef	Tempo
6	1000	10	185.80	0.0359	317.50	0.0526
6	2000	10	165.70	0.0614	283.90	0.0956
6	3000	10	169.70	0.0935	353.80	0.1676
6	4000	10	175.00	0.1315	339.10	0.2174
6	5000	10	171.80	0.1816	330.30	0.2938
6	6000	10	180.60	0.2541	346.60	0.4093
6	7000	10	187.70	0.3377	386.30	0.5708
6	8000	10	180.60	0.3939	386.20	0.6938
6	9000	10	176.40	0.4478	491.20	0.9412
6	10000	10	190.80	0.5330	425.00	0.9828
15	1000	10	186.60	0.0740	498.10	0.1550
15	2000	10	157.80	0.1240	701.10	0.3597
15	3000	10	184.60	0.2145	676.00	0.5602
15	4000	10	174.20	0.3086	533.20	0.7139
15	5000	10	185.60	0.4226	733.90	1.1531
15	6000	10	176.80	0.5015	679.70	1.4007
15	7000	10	200.70	0.6289	2724.60	5.3825
15	8000	9	228.33	0.7674	512.11	1.4858
15	9000	10	190.80	0.7701	1432.00	3.6530
15	10000	10	208.90	0.9092	1132.60	3.3200
28	1000	10	206.90	0.1605	666.90	0.4104
28	2000	10	167.90	0.2561	1071.50	1.0003
28	3000	10	179.30	0.4048	869.80	1.3134
28	4000	9	163.89	0.5266	1139.67	1.9538
28	5000	10	212.80	0.7693	1125.30	2.6330
28	6000	10	209.20	0.9197	1311.10	3.6020
28	7000	10	193.50	1.0334	1521.00	5.0810
28	8000	9	220.67	1.2479	929.11	3.5611
28	9000	10	235.10	1.4500	1653.20	6.6260
28	10000	10	206.10	1.4880	1438.70	6.2130
45	1000	10	185.30	0.2989	1007.90	0.9976
45	2000	10	189.90	0.5157	852.20	1.5422
45	3000	10	190.60	0.7467	1244.00	2.7180
45	4000	10	188.10	0.9885	991.50	3.2680
45	5000	9	230.22	1.3333	884.89	3.3622
45	6000	9	196.67	1.4733	944.33	4.3333
45	7000	10	195.30	1.6900	1444.90	7.1380
45	8000	10	225.30	2.0660	1417.50	7.6390
45	9000	10	246.50	2.4010	2051.60	11.5440
45	10000	10	224.90	2.5490	4134.90	27.1810
66	1000	10	221.60	0.6292	1324.20	2.8090
66	2000	10	216.70	1.0438	1703.40	4.7710
66	3000	9	194.33	1.4178	1178.78	5.6378
66	4000	9	211.78	1.8678	1164.56	6.4144
66	5000	9	229.44	2.3600	1846.11	11.3500
66	6000	10	212.30	2.6640	1810.10	11.5230
66	7000	9	216.44	3.1111	3463.78	23.4833
66	8000	9	205.33	3.4422	2537.00	19.1944
66	9000	9	218.67	3.9278	3901.11	34.3667
66	10000	8	236.38	4.4800	3306.88	32.4638
91	1000	10	190.10	1.0525	1461.70	5.1470
91	2000	9	180.78	1.6378	1603.11	6.8333
91	3000	9	198.00	2.3467	1514.44	9.2333
91	4000	9	198.89	2.9344	1771.33	11.6811
91	5000	10	194.90	3.4990	1927.40	15.4020
91	6000	10	220.70	4.3330	2724.20	22.5370
91	7000	10	176.70	4.5910	2585.70	24.6860
91	8000	9	205.00	5.4633	1818.11	20.7100
91	9000	7	200.43	6.0186	2070.86	25.0614
91	10000	6	217.33	6.8250	2328.83	28.1800

Tabela 1.1: Comparação entre ALGENCAN-B e ALGENCAN-G para problemas do Ellipsoid.

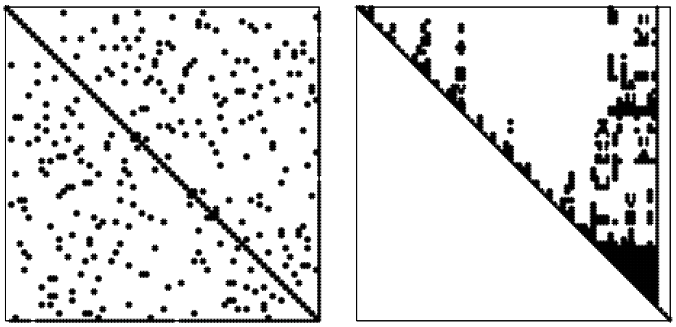
deste problema, os pontos iniciais foram gerados aleatoriamente, com cada ponto no intervalo $[-1, 1]$.

Nas Figuras 1.3 e 1.4 temos a distribuição dos elementos não-nulos na Hessiana da função Lagrangiano aumentado, a distribuição dos elementos não-nulos em cada umas das matrizes que a compõe (neste caso, a matriz resultante da soma das Hessianas das restrições e a matriz resultante da soma das matrizes de posto um $g_c g_c^T$, com g_c gradiente de uma restrição) e a distribuição dos elementos não-nulos na fatoração de Cholesky da Hessiana da função Lagrangiano aumentado no ponto final de uma instância do problema Hard-spheres com $nd = 3$ e $np = 72$, ou seja, 217 variáveis e 2628 restrições. Note que estas matrizes são esparsas, por este motivo esperamos que ALGENCAN-BS apresente melhor desempenho do que ALGENCAN-G.



(a) (b)

Figura 1.3: (a) Soma das Hessianas das restrições e (b) soma de $g_c g_c^T$ (g_c gradiente de uma restrição) no ponto final de uma instância do problema Hard-spheres com 217 variáveis e 2628 restrições.



(a) (b)

Figura 1.4: (a) Hessiana do Lagrangiano aumentado e (b) seu fator de Cholesky no ponto final de uma instância do problema Hard-spheres 217 variáveis e 2628 restrições.

Para verificar o comportamento de ALGENCAN-G, ALGENCAN-BS e ALGENCAN-A, resolvemos 80 instâncias do problema Hard-spheres com mais de 150 variáveis. Como este problema possui muitos pontos estacionários, diferentes pontos iniciais tendem a gerar diferentes soluções. Assim, escolhemos 8 pares (n, m) e, para cada um desses pares, geramos 10

problemas com pontos iniciais diferentes. Calculamos, então, a média aritmética dos valores de função encontrados, dos tempos gastos, do número de iterações de gradientes conjugados e do número de fatorações de Cholesky para todo problema resolvido com sucesso.

Na Tabela 1.2 temos as médias dos tempos, de iterações de gradientes conjugados, de fatorações de Cholesky e o dos valores de função obtidos por ALGENCAN-G, ALGENCAN-BS e ALGENCAN-A para resolver as instâncias. Os três métodos param satisfazendo os critério de convergência na resolução de todas as instâncias.

n	m	Tempo	ncg	nch	f_{med}
151	1275	1.4270	25537.70	-	8.686776E−01
217	2628	4.0160	46304.50	-	9.069195E−01
295	4851	11.7110	75383.10	-	9.382852E−01
385	8256	41.0270	194724.00	-	9.478572E−01
487	13203	108.7850	391735.40	-	9.628894E−01
601	20100	161.8010	349425.40	-	9.731471E−01
727	29403	358.4020	524719.80	-	9.721616E−01
865	41616	675.0760	646271.80	-	9.788911E−01

ALGENCAN-G aplicado a instâncias do problema Hard-spheres.

n	m	Tempo	ncg	nch	f_{med}
151	1275	1.1770	-	1864.80	8.683296E−01
217	2628	2.2730	-	2049.70	9.069188E−01
295	4851	7.0340	-	3809.40	9.315575E−01
385	8256	19.2070	-	6900.30	9.479004E−01
487	13203	32.5400	-	8358.10	9.587558E−01
601	20100	67.8300	-	11413.60	9.664310E−01
727	29403	81.3140	-	11385.30	9.721847E−01
865	41616	202.2460	-	15331.60	9.765676E−01

ALGENCAN-BS aplicado a instâncias do problema Hard-spheres.

n	m	Tempo	ncg	nch	f_{med}
151	1275	1.1943	6735.20	1437.60	8.683110E−01
217	2628	2.5860	15320.00	1473.70	9.069092E−01
295	4851	5.6350	22350.30	2032.30	9.313914E−01
385	8256	13.8750	38228.10	3335.60	9.478588E−01
487	13203	27.2460	55326.80	4422.90	9.586768E−01
601	20100	51.7180	73488.40	5594.70	9.664657E−01
727	29403	92.7770	109940.70	5679.90	9.721929E−01
865	41616	185.5940	151453.90	7864.10	9.765705E−01

ALGENCAN-A aplicado a instâncias do problema Hard-spheres.

Tabela 1.2: Média aritmética do tempo, de iterações de gradientes conjugados, de fatorações de Cholesky e do valor de função encontrado pelas versões de ALGENCAN em instâncias do problema Hard-spheres.

Comparando apenas ALGENCAN-BS e ALGENCAN-G, podemos perceber que ALGENCAN-BS leva menos tempo para resolver as instâncias. Além disso, comparando as médias dos valores de função obtidos por cada instância, ALGENCAN-BS obtém menores valores de função mais vezes do que ALGENCAN-G. Isso nos mostra que, quando a Hessiana do Lagrangiano aumentado e sua fatoração são esparsas, vale a pena usar BETRA-ESPARSO como algoritmo interno, pois ALGENCAN-BS é mais ecifiente e robusto do que ALGENCAN-G.

Analisando também os resultados obtidos pela aplicação do ALGENCAN-A, vemos que esta é a versão mais eficiente e robusta. O tempo médio gasto por este método para resolver as instâncias foi menor do que o tempo gasto por ALGENCAN-G para resolver todas as instâncias do problema, e menor do que o tempo gasto por ALGENCAN-BS para resolver as instâncias de 5 dos 8 problemas. Além disso, ALGENCAN-A obteve melhores valores médios de função mais vezes do que ALGENCAN-G e ALGENCAN-BS.

1.3.1.3 Mountain [31, 37]

Dada uma superfície $S(x, y)$ e pontos inicial p_I e final p_F em $\mathbb{R}^2$, o problema consiste em encontrar um caminho $p_I, p_1, p_2, ..., p_N, p_F$ de p_I a p_F, tal que $\max_{1 \leq k \leq N} S(p_k)$ seja mínimo. Além disso, a distância entre dois pontos consecutivos no caminho deve ser menor ou igual a uma dada tolerância.

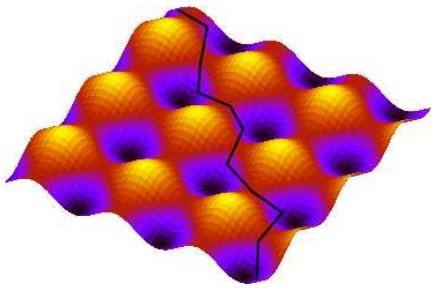


Figura 1.5: Representação do problema Mountain.

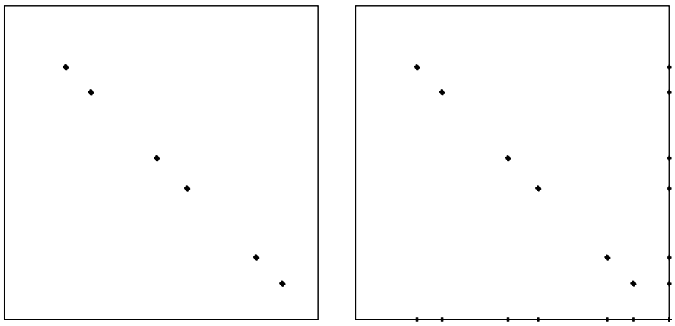
Um modelo para este problema é:

$$\begin{array}{lllll} \text{Minimizar} & z & & & \\ \text{sujeita a} & d(p_I, p_1)^2 & \leq d_{\max}^2, & & \\ & d(p_{k-1}, p_k)^2 & \leq d_{\max}^2, & k = 2, ..., N, & (1.16) \\ & d(p_N, p_F)^2 & \leq d_{\max}^2, & & \\ & S(p_k) & \leq z, & k = 1, ..., N, & \end{array}$$

onde $p_I, p_F \in \mathbb{R}^2$ e $d_{\max}$ são dados, $z \in \mathbb{R}$, $p_k \in \mathbb{R}^2$, $d(.,.)$ é a distância euclidiana, e $S(x, y)$ representa a superfície.

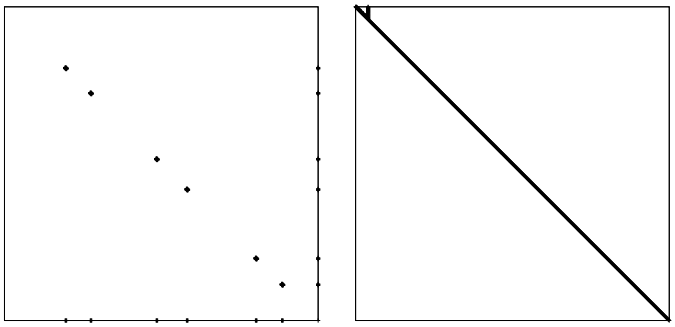
O número de variáveis deste problema é $n = 2N + 1$. O número de restrições é $m = 2N + 1$. Neste exemplo usaremos $p_I = (-10, -10)$, $p_F = (10, 10)$ e $S(x, y) = sen(x) + cos(y)$. Os valores iniciais de $p_1, p_2, ..., p_N$ correspondem a uma perturbação dos pontos igualmente espaçados em $[p_I, p_F]$. O valor inicial de z é $z = \max\{S(p_I), S(p_1), S(p_2), ..., S(p_N), S(p_F)\}$.

Nas Figuras 1.6 e 1.7 temos a distribuição dos elementos não-nulos na Hessiana da função Lagrangiano aumentado, a distribuição dos elementos não-nulos em cada umas das matrizes que a compõe (neste caso, a matriz resultante da soma das Hessianas das restrições e a matriz resultante da soma das matrizes de posto um $g_c g_c^T$, com g_c gradiente de uma restrição) e a distribuição dos elementos não-nulos na fatoração de Cholesky da Hessiana da função Lagrangiano aumentado no ponto final de uma instância do problema Mountain com $N = 300$, ou seja, 601 variáveis e 601 restrições. Note que estas matrizes são muito esparsas e, portanto, esperamos que ALGENCAN-BS apresente melhor desempenho do que ALGENCAN-G.



(a) (b)

Figura 1.6: (a) Soma das Hessianas das restrições e (b) soma de $g_c g_c^T$ (g_c gradiente de uma restrição) no ponto final de uma instância do problema Mountain com 601 variáveis e 601 restrições.



(a) (b)

Figura 1.7: (a) Hessiana do Lagrangiano aumentado e (b) seu fator de Cholesky no ponto final de uma instância do problema Mountain com 601 variáveis e 601 restrições.

Para comparar o desempenho dos métodos, escolhemos 12 pares (n, m) com mais de 150 variáveis e, para cada um destes pares, geramos 10 problemas com pontos iniciais diferentes. O valor de $d_{\max}$ é sempre definido como o dobro da distância entre p_I e p_F dividida pelo número de pontos. Calculamos a média aritmética dos valores de função encontrados, do número de iterações de gradientes conjugados, do número de fatorações de Cholesky e dos tempos gastos por cada método para resolver instâncias com sucesso (o que aconteceu com todos os métodos ao resolver todas as instâncias).

Na Tabela 1.3 temos as médias dos tempos, de iterações de gradientes conjugados, de fatorações de Cholesky e o dos valores de função obtidos por Algencan-G,Algencan-BS e Algencan-A para resolver as instâncias.

Note que, como esperado, Algencan-BS é mais rápido do que Algencan-G, já que tanto a Hessiana do Lagrangiano aumentado como sua fatoração são esparsas. No entanto, quando analisamos o valor de função obtido por cada método, há mais casos em que, utilizando Algencan-G, obtemos valor de função menor do que quando utilizamos Algencan-BS.

Usando Algencan-A, no qual podem ser combinadas iterações internas de Gencan e Betra-esparso, obtemos resultados melhores do que os obtidos por Algencan-G ou Algencan-BS. Comparando Algencan-A com Algencan-BS, temos que Algencan-BS é mais rápido para resolver um número maior de instâncias. No entanto, Algencan-A obtém valores médios de função melhores do que Algencan-BS mais vezes. Quando comparado com Algencan-G, Algencan-A se mostra mais rápido e chega a valores médios de função melhores o mesmo número de vezes que Algencan-G chega a valores médios de função melhor. Ou seja, Algencan-A é menos eficiente, porém mais robusto do que Algencan-BS. E Algencan-A é mais eficiente e tão robusto quanto Algencan-G.

1.3.2 Comparações utilizando a coleção Cuter

Como mencionado anteriormente, selecionamos todos os problemas da coleção Cuter que possuem segunda derivada e pelo menos uma restrição que não seja de caixa e os dividimos em dois grupos: problemas com até 150 variáveis (totalizando 366 problemas) e problemas com mais de 150 variáveis (totalizando 363 problemas).

Nesta seção, utilizaremos os problemas do primeiro grupo para comparar o desempenho de Algencan-B e Algencan-G. Os problemas do segundo grupo serão utilizados para comparar o desempenho de Algencan-A e Algencan-G. Compararemos, em seguida, o desempenho de Algencan-B e Algencan-A com Lancelot B [18, 19, 20].

Note que, como Algencan-BS foi desenvolvido para resolver problemas com matriz Hessiana do Lagrangiano aumentado esparsa e apresenta bons resultados quando temos problemas esparsos, não iremos comparar o desempenho de Algencan-BS na resolução dos problemas da coleção Cuter, pois não temos informação sobre a esparsidade destes problemas.

1.3.2.1 Comparação entre Algencan-B e Algencan-G para problemas pequenos

Nesta seção vamos comparar o desempenho de Algencan-B com Algencan-G na resolução dos 366 problemas da coleção Cuter com até 150 variáveis e pelo menos uma restrição que não seja de caixa.

O iterando final de Algencan-B satisfaz viabilidade com tolerância 10^{-8} em 338 dos 366 problemas. Nos 28 problemas restantes, Algencan-B pára com a seguinte explicação:

- nos problemas model, vanderm4 e a4x12, Algencan-B pára devido ao limite de

n	m	Tempo	ncg	nch	f_{med}
601	601	2.3670	29399.70	-	-4.411715E−03
801	801	5.3650	50950.00	-	-2.486516E−03
1001	1001	11.4200	89525.80	-	-1.529347E−03
1201	1201	18.2660	119516.60	-	-1.078653E−03
1401	1401	33.2540	192917.60	-	-7.802251E−04
1601	1601	55.1850	279684.90	-	-6.202542E−04
1801	1801	110.4960	515302.40	-	-4.918522E−04
2001	2001	139.5170	561518.30	-	-3.440294E−04
2201	2201	231.9960	863518.70	-	-3.299601E−04
2401	2401	273.3080	925614.00	-	-2.668200E−04
2601	2601	348.1010	1072399.70	-	-2.194401E−04
2801	2801	585.7360	1701364.40	-	-1.913293E−04

Algencan-G aplicado a instâncias do problema Mountain.

n	m	Tempo	ncg	nch	f_{med}
601	601	1.2812	-	883.20	-4.411715E−03
801	801	2.2070	-	955.20	-2.348135E−03
1001	1001	4.2650	-	1245.30	-1.273953E−03
1201	1201	6.2600	-	1345.60	-9.929757E−04
1401	1401	8.4020	-	1409.90	-7.690034E−04
1601	1601	11.6610	-	1520.90	-6.140112E−04
1801	1801	22.1240	-	2453.90	-4.918522E−04
2001	2001	30.1300	-	2745.20	-3.716052E−04
2201	2201	36.4280	-	2757.80	-2.948198E−04
2401	2401	39.1920	-	2554.50	-2.773026E−04
2601	2601	48.8170	-	2687.40	-2.306883E−04
2801	2801	58.8790	-	2842.50	-1.963104E−04

Algencan-BS aplicado a instâncias do problema Mountain.

n	m	Tempo	ncg	nch	f_{med}
601	601	1.3780	86.60	939.40	-4.411715E−03
801	801	2.2240	307.60	932.60	-2.348136E−03
1001	1001	4.3250	556.60	1296.20	-1.401650E−03
1201	1201	7.6570	878.40	1701.60	-1.078653E−03
1401	1401	8.3080	616.80	1377.80	-7.690038E−04
1601	1601	11.9260	1811.50	1483.80	-6.171326E−04
1801	1801	18.3120	1117.30	2009.30	-4.924814E−04
2001	2001	26.5860	1793.70	2384.50	-3.716002E−04
2201	2201	33.9100	3841.30	2501.90	-3.123905E−04
2401	2401	37.6710	3427.10	2362.30	-2.458569E−04
2601	2601	55.1010	5844.80	2835.30	-2.306882E−04
2801	2801	54.0350	6313.50	2517.20	-1.938197E−04

Algencan-A aplicado a instâncias do problema Mountain.

Tabela 1.3: Média aritmética do tempo, de iterações de gradientes conjugados, de fatorações de Cholesky e do valor de função encontrado pelas versões de Algencan em instâncias do problema Mountain.

tempo de CPU imposto na execução de cada par problema/método (10 minutos);

- no problema HS75, ALGENCAN-B pára por atingir o número máximo de iterações permitido (50 iterações);

- nos problemas ALSOTAME, HIMMELBD, HS88, HS91, HS93, S365MOD, S365, EIGMINA e KISSING, ALGENCAN-B pára declarando que há falta de progresso na diminuição da inviabilidade;

- nos problemas SNAKE, ARGAUSS, GROWTH, LOOTSMA, YFITNE, LEWISPOL, HS106, NASH, DEGENLPA, DISC2, DISCS, MESH, AVION2, QPNBLEND e QPCBOEI2, ALGENCAN-B pára declarando que o parâmetro de penalização se tornou muito grande.

Considerando os 338 problemas para os quais ALGENCAN-B pára em um ponto viável, ALGENCAN-B pára satisfazendo seu critério de parada relativo a sucesso em 320 problemas. Nos 18 problemas restantes, a explicação é a seguinte:

- no problema CRESC132, ALGENCAN-B pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

- nos problemas CSFI1, HS84, CRESC50, HS87, ELATTAR, HS99, HS116, CONCON, MCONCON, DEGENLPB, HS99EXP, HIMMELBJ, BATCH, QPCBLEND, LAKES, TRIMLOSS e QPNBOEI2, ALGENCAN-B pára por atingir o número máximo de iterações permitido.

Analisemos agora ALGENCAN-G. O iterando final de ALGENCAN-G satisfaz viabilidade com tolerância 10^{-8} em 341 dos 366 problemas. Nos 25 problemas restantes, ALGENCAN-G pára com a seguinte explicação:

- no problema A4X12, ALGENCAN-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

- nos problemas HS99EXP e QPCBOEI2, ALGENCAN-G pára por atingir o número máximo de iterações permitido (50 iterações);

- nos problemas HS88, CSFI1, HS93, TRIGGER, S365MOD, S365, MESH, VANDERM4, EIGMINA e KISSING, ALGENCAN-G pára declarando que há falta de progresso na diminuição da inviabilidade;

- nos problemas HIMMELBD, SNAKE, ARGAUSS, GROWTH, LOOTSMA, YFITNE, LEWISPOL, NASH, DISC2, DISCS, MODEL e MSS1, ALGENCAN-G pára declarando que o parâmetro de penalização se tornou muito grande.

Considerando os 341 problemas para os quais ALGENCAN-G pára em um ponto viável, ALGENCAN-G pára satisfazendo seu critério de parada relativo a sucesso em 316 problemas. Nos 25 problemas restantes, a explicação é a seguinte:

- no problema HS91, ALGENCAN-G pára por atingir o tempo limite de CPU (10 minutos);

- em 24 problemas[2], ALGENCAN-G pára por atingir o número máximo de 50 iterações permitido.

Considerando os 334 problemas nos quais ALGENCAN-B e ALGENCAN-G obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final de ALGENCAN-B é equivalente ao obtido por ALGENCAN-G. No entanto,

- em 11 problemas (HET-Z, HS89, WOMFLET, OET6, CRESC100, HS92, ELATTAR, OET7, ROBOT, HIMMELBJ, GROUPING), ALGENCAN-B obteve valor de função menor do que ALGENCAN-G;
- em 12 problemas (HS70, CRESC132, CRESC50, HS54, HS105, HS116, MAKELA3, HIMMELBK, GOFFIN, KISSING2, TRIMLOSS, QPNBOEI2), ALGENCAN-G obteve valor de função menor do que ALGENCAN-B.

Nos 311 problemas restantes, tanto ALGENCAN-B como ALGENCAN-G obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 311 problemas:

- ALGENCAN-B realizou menos avaliações de função em 190 casos;
- ALGENCAN-G realizou menos avaliações de função em 76 casos;
- ALGENCAN-B e ALGENCAN-G realizaram o mesmo número de avaliações de função em 45 casos;
- ALGENCAN-B foi mais rápido que ALGENCAN-G (com tolerância de 10%) em 105 casos;
- ALGENCAN-G foi mais rápido que ALGENCAN-B (com tolerância de 10%) em 82 casos;
- ALGENCAN-B e ALGENCAN-G gastaram a mesma quantidade de tempo em 124 casos;
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (45 problemas), ALGENCAN-B foi mais rápido que ALGENCAN-G 25 vezes e ALGENCAN-G foi mais rápido que ALGENCAN-B 16 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 4 casos).

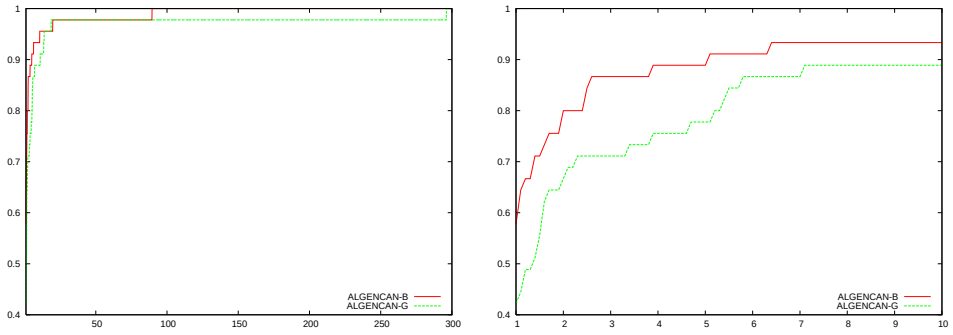
Na Tabela 1.4 temos um resumo da comparação entre ALGENCAN-B e ALGENCAN-G. Na linha da tabela que se refere a problemas viáveis, por exemplo, temos a/b. Isso significa que a problemas são viáveis de um total de b problemas. O mesmo vale para as demais linhas da tabela.

Considerando os 311 problemas para os quais tanto ALGENCAN-B como ALGENCAN-G chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho (veja Apêndice A). Na Figura 1.8, utilizamos o tempo de CPU como medida de desempenho. Na Figura 1.9, usamos o número de avaliações de função como medida de desempenho.

[2] POLAK4, HS75, HS268, HS84, S268, CRESC100, CRESC132, CRESC50, HS87, HS99, HS106, HS116, CONCON, MCONCON, DEGENLPA, DEGENLPB, HIMMELBJ, BATCH, AVION2, QPCBLEND, QPNBLEND, LAKES, HAIFAM e QPNBOEI2.

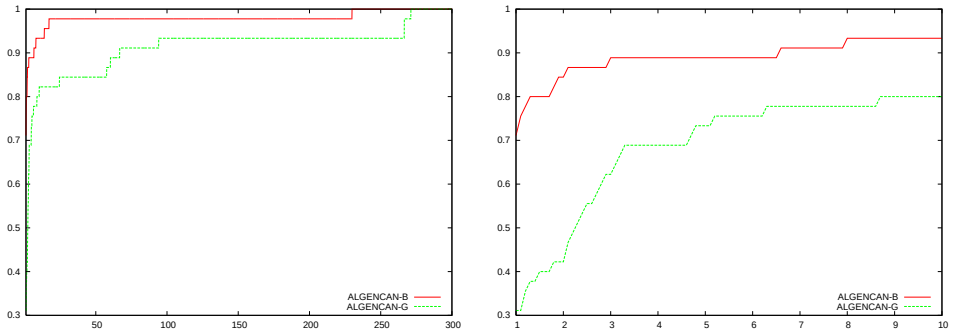
	ALGENCAN-B	ALGENCAN-G
Problemas viáveis	338/366	341/366
Satisfez critérios de convergência	320/338	316/341
Valor de função melhor	11/334	12/334
Menos avaliações de função	190/311	76/311
Tempo menor (10% tolerância)	105/311	82/311

Tabela 1.4: Comparação entre ALGENCAN-B e ALGENCAN-G para problemas selecionados da coleção CUTEr



(a) (b)

Figura 1.8: Curva de perfil de desempenho comparando ALGENCAN-B e ALGENCAN-G usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).



(a) (b)

Figura 1.9: Curva de perfil de desempenho comparando ALGENCAN-B e ALGENCAN-G usando o número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

O fato de Algencan-B parar em ponto final inviável em três problemas a mais do que Algencan-G e obter valor de função pior em um problema a mais do que Algencan-G, pode ser explicado fato de que Gencan, diferentemente de Betra, vem sendo usado há muitos anos para resolver diversos tipo de problemas (veja, por exemplo, [9, 10, 14]), motivo pelo qual seus parâmetros estão bem calibrados. Isto é uma justificativa para a perda de robustez de Algencan-B com relação a Algencan-G. Apesar desta pequena perda de robustez, Algencan-B se mostrou mais eficiente. Por isso o utilizaremos nas comparações com Lancelot B para resolução de problemas pequenos.

1.3.2.2 Comparação entre Algencan-A e Algencan-G para problemas grandes

Vamos agora comparar o desempenho de Algencan-A com Algencan-G na resolução dos 363 problemas da coleção CUTEr com mais de 150 variáveis e pelo menos uma restrição que não seja de caixa.

Em 204 dos 363 problemas, o iterando final de Algencan-A satisfaz viabilidade com tolerância 10^{-8}. Nos 159 problemas restantes, Algencan-A pára com a seguinte explicação:

- em 135 problemas[3], Algencan-A pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas methanol e yatp1sq, Algencan-A pára por atingir o número máximo de iterações permitido (50 iterações). Nestes dois problemas o valor da norma das restrições não está definido (mensagem NaN);
- nos problemas arwhdne, ktmodel, dittert, eigena, woodsne, lukvle10 e lukvli10, Algencan-A pára declarando que há falta de progresso na diminuição da inviabilidade;
- nos problemas agg, yorknet, qpcboei1, qpnboei1, static3, drugdise, mss2, camshape, helsby, mss3, rocket, lukvle2 e lukvli2, Algencan-A pára declarando que o parâmetro de penalização se tornou muito grande;
- nos problemas rotdisc e optcdeg3, Algencan-A pára por erro na fatoração de Cholesky (matriz a ser fatorada é definida positiva, mas, por erros numéricos, não é possível realizar a fatoração).

[3] polygon, arglale, arglble, arglcle, lincont, hadamard, elec, chain, gpp, reading7, twirimd1, ferrisdc, yao, reading8, liswet10, liswet11, liswet12, liswet1, liswet2, liswet7, liswet8, liswet9, eigenb, eigenc2, eigenc, bridgend, flosp2hh, flosp2hl, flosp2hm, flosp2th, flosp2tl, flosp2tm, catenary, catena, ssnlbeam, optmass, twiribg1, lubrifc, lubrif, drcavty1, drcavty2, drcavty3, reading1, bloweya, bloweyb, bloweyc, reading3, sreadin3, robotarm, corkscrw, optcdeg2, optctrl3, optctrl6, bratu2dt, porous1, chemrcta, chemrctb, powell20, reading4, reading5, svanberg, orthrds2, glider, clnlbeam, drugdis, coshfun, reading2, brainpc0, brainpc1, brainpc3, brainpc4, brainpc5, brainpc6, brainpc7, brainpc8, brainpc9, synpop24, orthrdm2, stnqp1, pinene, junkturn, lukvle12, lukvle17, lukvle18, lukvli12, lukvli15, lukvle13, lukvle14, lukvle6, lukvli6, cvxqp1, cvxqp2, cvxqp3, dtoc6, lukvle4, lukvle5, lukvle7, lukvle8, lukvli4, lukvli5, lukvli8, ncvxqp1, ncvxqp2, ncvxqp3, ncvxqp4, ncvxqp5, ncvxqp6, ncvxqp7, ncvxqp8, ncvxqp9, eg3, gasoil, marine, brainpc2, lobsterz, cont6-qq, aug2dc, aug2d, gausselm, a0nndnil, a2nndndl, a2nndnil, a2nndnsl, a2nsdsil, a5nndndl, a5nndnil, a5nndnsl, a5nsdsil, aug3dcqp, aug3dqp, aug3d, allinqp, net4, portsnqp e yatp2sq.

Considerando os 204 problemas para os quais Algencan-A pára em um ponto viável, Algencan-A pára satisfazendo seu critério de parada relativo a sucesso em 189 problemas. Nos 15 problemas restantes, a explicação é a seguinte:

- nos problemas qpnstair, net3, nuffield, liswet5, eigenb2, eigenbco, eigencco, trainh, hvycrash, lukvli17, lukvli1, lukvli9 e cont5-qp, Algencan-A pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

- nos problemas leaknet e sawpath, Algencan-A pára por atingir o número máximo de iterações permitido.

Analisemos agora Algencan-G. O iterando final de Algencan-G satisfaz viabilidade com tolerância 10^{-8} em 185 dos 363 problemas. Nos 178 problemas restantes, Algencan-G pára com a seguinte explicação:

- em 160 problemas[4], Algencan-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

- nos problemas methanol e yatp1sq, Algencan-G pára por atingir o número máximo de iterações permitido. Nestes dois problemas o valor da norma das restrições não está definido (mensagem NaN);

- nos problemas dittert, eigena, woodsne, robotarm, lukvle10, lukvle2, lukvle4, lukvli10 e lukvli2, Algencan-G pára declarando que há falta de progresso em obter viabilidade;

- nos problemas yorknet, arwhdne, sawpath, drugdise, ktmodel, mss2 e mss3, Algencan-G pára declarando que o parâmetro de penalização se tornou muito grande.

Considerando os 185 problemas para os quais Algencan-G pára em um ponto viável, Algencan-G pára satisfazendo seu critério de parada relativo a sucesso em 170 problemas. Nos 15 problemas restantes, a explicação é a seguinte:

[4] polygon, arglale, arglble, arglcle, lincont, qpcboei1, qpnboei1, hadamard, elec, camshape, chain, rotdisc, nuffield, gpp, reading7, twirimd1, helsby, ferrisdc, yao, reading8, liswet10, liswet11, liswet12, liswet1, liswet2, liswet7, liswet8, liswet9, rocket, eigenb, eigenc2, eigenc, bridgend, flosp2hh, flosp2hl, flosp2hm, flosp2th, flosp2tl, flosp2tm, zigzag, catenary, catena, ssnlbeam, optmass, twiribg1, hanging, lubrifc, lubrif, drcavty1, drcavty2, drcavty3, trainf, trainh, bloweya, bloweyb, bloweyc, sreadin3, corkscrw, optcdeg2, optcdeg3, optctrl3, optctrl6, orthregf, bratu2dt, ubh5, spmsqrt, bdvalue, chemrcta, chemrctb, powell20, reading4, reading5, semicn2u, semicon1, semicon2, svanberg, orthrds2, orthregd, glider, dtoc2, clnlbeam, drugdis, coshfun, brainpc0, brainpc1, brainpc3, brainpc4, brainpc5, brainpc6, brainpc7, brainpc8, brainpc9, synpop24, orthrege, orthrdm2, stnqp1, pinene, ubh1, junkturn, lukvle12, lukvle15, lukvle16, lukvle17, lukvle18, lukvli12, lukvli15, lukvli16, lukvli17, dtoc5, lukvle13, lukvle14, lukvli13, bdvalues, cvxqp1, cvxqp2, cvxqp3, dtoc6, lukvle7, lukvle8, lukvli4, lukvli5, lukvli8, ncvxqp1, ncvxqp2, ncvxqp3, ncvxqp4, ncvxqp5, ncvxqp6, ncvxqp7, ncvxqp8, ncvxqp9, eg3, reading9, orthrgdm, gasoil, marine, brainpc2, lobsterz, cont6-qq, aug2dcqp, aug2dc, aug2dqp, aug2d, gausselm, a0nndnil, a2nndndl, a2nndnil, a2nndnsl, a2nsdsil, a5nndndl, a5nndnil, a5nndnsl, a5nsdsil, aug3dcqp, aug3dqp, aug3d, allinqp, net4, portsnqp e yatp2sq.

- nos problemas QPNSTAIR, NET3, LISWET5, EIGENB2, EIGENBCO, EIGENCCO, READING3, READING2, LUKVLE9, LUKVLI1 e CONT5-QP, ALGENCAN-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas LEAKNET, AGG, STATIC3, e HVYCRASH, ALGENCAN-G pára por atingir o número máximo de iterações permitido.

Considerando os 176 problemas nos quais ALGENCAN-A e ALGENCAN-G obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final de ALGENCAN-A é equivalente ao obtido por ALGENCAN-G. No entanto,

- em 6 problemas (STEENBRB, STEENBRF, STEENBRC, STEENBRE, CATMIX, ORTHREGC), ALGENCAN-A obteve valor de função menor do que ALGENCAN-G;
- em 5 problemas (STEENBRD, STEENBRG, LUKVLE11, LUKVLI11, LUKVLI9), ALGENCAN-G obteve valor de função menor do que ALGENCAN-A.

Nos 165 problemas restantes, tanto ALGENCAN-A como ALGENCAN-G obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 165 problemas:

- ALGENCAN-A realizou menos avaliações de função em 55 casos;
- ALGENCAN-G realizou menos avaliações de função em 13 casos;
- ALGENCAN-A e ALGENCAN-G realizaram o mesmo número de avaliações de função em 97 casos;
- ALGENCAN-A foi mais rápido que ALGENCAN-G (com tolerância de 10%) em 39 casos;
- ALGENCAN-G foi mais rápido que ALGENCAN-A (com tolerância de 10%) em 27 casos;
- ALGENCAN-A e ALGENCAN-G gastaram a mesma quantidade de tempo em 99 casos.

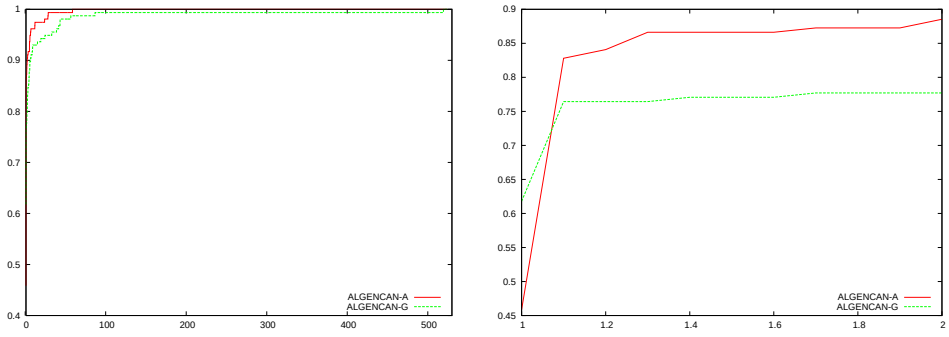
Na Tabela 1.5 temos um resumo da comparação entre ALGENCAN-A e ALGENCAN-G.

	ALGENCAN-A	ALGENCAN-G
Problemas viáveis	204/363	185/363
Satisfez critérios de convergência	189/204	170/185
Valor de função melhor	6/176	5/176
Menos avaliações de função	55/165	13/165
Tempo menor (10% tolerância)	39/165	27/165

Tabela 1.5: Comparação entre ALGENCAN-A e ALGENCAN-G para problemas selecionados da coleção CUTEr

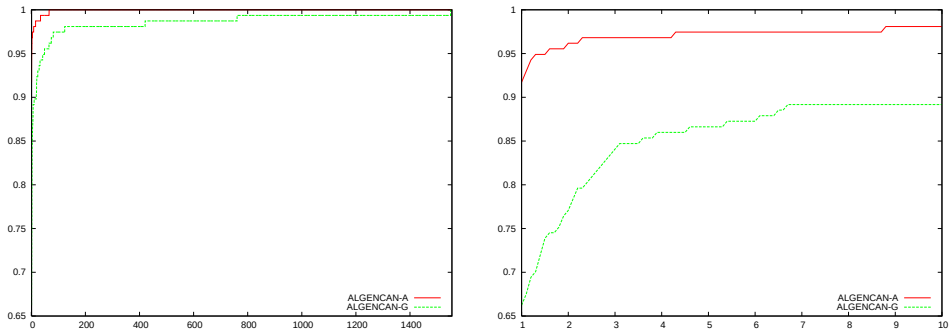
Considerando os 165 problemas para os quais tanto ALGENCAN-A como ALGENCAN-G chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 1.10, utilizamos o tempo de CPU como medida de desempenho. Na Figura 1.11, usamos o número de avaliações de função como medida de desempenho.

Note que ALGENCAN-A resolveu mais problemas e se mostrou mais rápido do que ALGENCAN-G. Portanto, ele será utilizado nas comparações com LANCELOT B para resolução de problemas grandes.



(a) (b)

Figura 1.10: Curva de perfil de desempenho comparando ALGENCAN-A e ALGENCAN-G usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).



(a) (b)

Figura 1.11: Curva de perfil de desempenho comparando ALGENCAN-A e ALGENCAN-G usando o número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

1.3.2.3 Comparação das versões de Algencan com Lancelot B

Nesta seção vamos comparar o desempenho de Algencan-B e Algencan-A com Lancelot B [18, 19, 20] na resolução dos problemas selecionados da coleção Cuter, com segunda derivada e pelo menos uma restrição que não seja de caixa.

Lancelot B é um método de Lagrangiano aumentado que, a cada iteração, utiliza um algoritmo de regiões de confiança com ponto de Cauchy e gradientes conjugados para resolver os subproblemas. Lancelot B adiciona variáveis de folga para trabalhar apenas com restrições de igualdade e de caixa. Usamos os parâmetros padrão de Lancelot B, mudando apenas as tolerâncias para viabilidade e otimalidade[5] para 10^{-8}. O tempo medido foi o utilizado para resolver cada problema uma única vez.

Lancelot B transforma problemas de viabilidade (ou seja, problemas sem função objetivo) em problemas de minimização da soma dos quadrados das inviabilidades. Assim, Lancelot B considera o problema de encontrar x tal que $h(x) = 0$ como

$$\text{Minimizar} \quad \|h(x)\|_2^2, \tag{1.17}$$

Como a solução x^* de (1.17) com tolerância 10^{-8} para otimalidade não implica em obter um ponto viável com $\|h(x^*)\|_\infty \leq 10^{-8}$, eliminamos os 89 problemas com esta característica. Foram eliminados 42 dos 366 problemas do grupo com até 150 variáveis[6] e 47 dos 363 problemas do grupo com mais de 150 variáveis[7].

Primeiramente, vamos comparar o desempenho de Algencan-B e Lancelot B no conjunto de problemas pequenos (até 150 variáveis), eliminando os problemas sem função objetivo mencionados acima. Note que, neste novo conjunto de 324 problemas, Algencan-B pára em pontos viáveis com tolerância 10^{-8} em 301 problemas.

O iterando final de Lancelot B satisfaz viabilidade com tolerância 10^{-8} em 284 dos 324 problemas restantes . Nos demais 40 problemas, a explicação é a seguinte:

- em 31 problemas[8], Lancelot B pára por atingir o número máximo de iterações;
- nos problemas lootsma, polak4, hs93, lewispol, hs99, s365, nash, model e grouping, Lancelot B pára declarando que o problema não aparenta ter uma solução viável.

[5] Acrescentamos as linhas `primal-accuracy-required 1.0d-08` e `dual-accuracy-required 1.0d-08` ao arquivo RUNLANB.SPC.

[6] booth, cluster, cubene, gottfr, himmelba, himmelbc, himmelbd, hypcir, powellbs, powellsq, rsnbrne, sinvalne, argauss, growth, hatfldf, himmelbe, pfit1, pfit2, pfit3, pfit4, recipe, yfitne, zangwil3, aircrfta, heart6, trigger, heart8, coolhans, nystrom5, res, hatfldg, hydcar6, methanb8, methanl8, chnrsbne, deconvne, hydcar20, chandheq, vanderm1, vanderm2, vanderm3, vanderm4.

[7] arglale, arglble, arglcle, argtrig, brownale, qr3dbd, arwhdne, chandheu, integreq, qr3d, msqrta, msqrtb, cbratu3d, eigena, eigenau, eigenb, eigenc, flosp2hh, flosp2hl, flosp2hm, flosp2th, flosp2tl, flosp2tm, cbratu2d, bratu3d, drcavty1, drcavty2, drcavty3, woodsne, bratu2d, bratu2dt, porous1, porous2, spmsqrt, artif, bdvalue, broydn3d, broydnbd, chemrcta, chemrctb, semicn2u, semicon1, semicon2, channel, bdvalues, yatp1sq, yatp2sq.

[8] snake, oet5, cresc100, cresc132, cresc4, cresc50, hs87, hs101, hs102, hs103, s365mod, hs106, hs109, truspyr2, hs116, concon, dembo7, errinbar, tenbars4, launch, hs99exp, discs, mesh, himmelbj, avion2, feedloc, lakes, haifam, reading6, a4x12 e qpnboei2.

Em 230 destes 284 problemas, LANCELOT B pára satisfazendo seu critério de parada relativo a sucesso. Nos 54 problemas restantes, a explicação é a seguinte:

- em 53 problemas[9], LANCELOT B pára declarando que o passo é muito pequeno e não gera mudança na função objetivo;
- no problema HS67, LANCELOT B pára declarando que o problema não aparenta ter uma solução viável.

Considerando os 274 problemas nos quais ALGENCAN-B e LANCELOT B obtiveram ponto final viável, observamos que, na maior parte dos casos, os valores de função no ponto final são equivalentes. No entanto,

- em 15 problemas (HET-Z, BT2, KIWCRESC, SPIRAL, FLETCHER, HS44NEW, HS90, HS85, HALDMADS, HS54, HS97, HS98, OET7, KISSING2, EIGMINB), ALGENCAN-B obteve valor de função menor do que LANCELOT B;
- em 11 problemas (HS13, HS59, ALLINITC, HS70, PENTAGON, TENBARS1, TENBARS2, HIMMELBK, GOFFIN, QPCBLEND, TRIMLOSS), LANCELOT B obteve valor de função menor do que ALGENCAN-B.

Nos 248 problemas restantes, tanto ALGENCAN-B como LANCELOT B obtiveram pontos viáveis com valores de função equivalentes. Como não dispomos do número de avaliações de função utilizados por LANCELOT B, analisaremos apenas o tempo gasto. Considerando estes 248 problemas:

- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo (116 problemas), ALGENCAN-B foi mais rápido que LANCELOT B (com tolerância de 10%) 88 vezes e LANCELOT B foi mais rápido que ALGENCAN-B (com tolerância de 10%) 23 vezes (ambos gastaram a mesma quantidade de tempo em 5 casos).
- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (46 problemas), ALGENCAN-B foi mais rápido que LANCELOT B 27 vezes e LANCELOT B foi mais rápido que ALGENCAN-B 17 vezes (ambos gastaram a mesma quantidade de tempo em 2 casos).

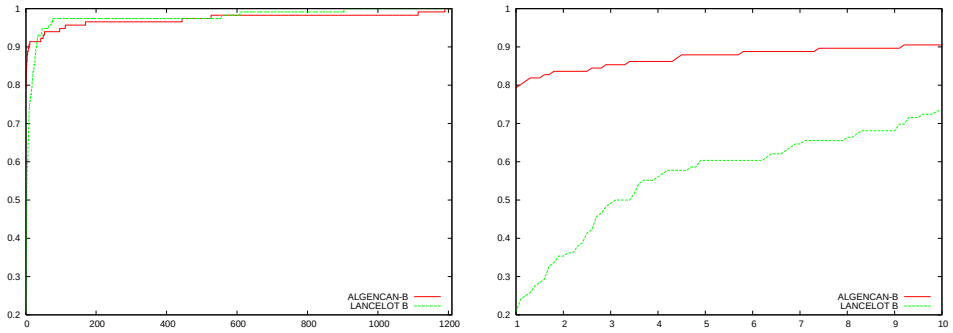
Na Tabela 1.6 temos um resumo da comparação entre ALGENCAN-B e LANCELOT B. Quando comparamos qual método gastou menos tempo para resolver os problemas, estamos considerando apenas os problemas para os quais pelo menos um dos métodos gastou mais de 0.01 segundo.

Considerando os 248 problemas para os quais tanto ALGENCAN-B como LANCELOT B chegaram a um ponto viável com mesmo valor de função, construímos um gráfico de perfil de desempenho. Usamos o tempo de CPU como medida de desempenho, eliminando os problemas

[9] HET-Z, HS19, HS20, SIPOW1, SIPOW2, HS62, OET1, OET2, TFI3, HS74, HS75, OET3, OET4, SIPOW3, SIPOW4, BT7, CSFI2, EXPFITC, HS83, HS84, HS85, HS86, OET6, HALDMADS, HS54, DUALC2, ELATTAR, HS100MOD, OET7, DUALC5, DUALC8, DUALC1, HS114, TRUSPYR1, MCONCON, TENBARS1, TENBARS2, TENBARS3, DEGENLPA, DEGENLPB, KSIP, MRIBASIS, LOADBAL, NET1, BATCH, DNIEPER, PRODPL0, PRODPL1, CORE1, QPCBLEND, QPNBLEND, NET2 e QPCBOEI2.

	ALGENCAN-B	LANCELOT B
Problemas viáveis	301/324	284/324
Satisfez critérios de convergência	283/301	230/284
Valor de função melhor	15/274	11/274
Tempo menor (10% tolerância)	88/116	23/116

Tabela 1.6: Comparação entre ALGENCAN-B e LANCELOT B para problemas selecionados da coleção CUTEr



(a) (b)

Figura 1.12: Curva de perfil de desempenho comparando ALGENCAN-B e LANCELOT B usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

para os quais ambos os métodos gastaram menos de 0.01 segundo (veja Figura 1.12). Note que ALGENCAN-B se mostrou mais robusto e eficiente, o que mostra que este é um método competitivo.

Vamos comparar o desempenho de ALGENCAN-A e LANCELOT B no conjunto de problemas grandes (mais de 150 variáveis), eliminando os 47 problemas sem função objetivo mencionados anteriormente. Note que, neste novo conjunto de 316 problemas, ALGENCAN-A pára em pontos viáveis com tolerância 10^{-8} em 180 problemas.

O iterando final de LANCELOT B satisfaz viabilidade com tolerância 10^{-8} em 242 dos 316 problemas restantes. Nos 74 demais problemas, a explicação é a seguinte:

- no problema SYNPOP24, LANCELOT B pára declarando sucesso, porém o ponto final é inviável;
- em 18 problemas[10], LANCELOT B pára por atingir o tempo limite de CPU de execução dos experimentos (10 minutos);
- em 48 problemas[11], LANCELOT B pára por atingir o número máximo de iterações;

[10] DRUGDISE, ROTDISC, DIXCHLNV, TWIRIMD1, HELSBY, FERRISDC, EIGENBCO, CORKSCRW, READING5, COSHFUN, READING2, BRAINPC0, BRAINPC3, BRAINPC4, BRAINPC5, LUKVLI18, NCVXQP2 e GASOIL.

[11] LEAKNET, AGG, NGONE, MADSSCHJ, PRIMALC1, YORKNET, QPCBOEI1, QPNBOEI1, NET3, STATIC3, STEENBRD, STEENBRC, STEENBRE, KTMODEL, SMMPSF, MSS2, CAMSHAPE, CHAIN, NUFFIELD, READING8, MSS3, ROCKET, CATENARY, CATENA, OPTMASS, LUBRIF, READING1, BLOWEYA, BLOWEYB, BLOWEYC, HVYCRASH, READING3,

- nos problemas LINCONT, SAWPATH, ORTHRDS2, ORTHREGD, ORTHRGDS, JUNKTURN e METHANOL, LANCELOT B pára declarando que o problema não aparenta ter uma solução viável.

Em 42 problemas[12], LANCELOT B pára por atingir o tempo limite de 10 minutos imposto para os experimentos. Nestes problemas não é possível obter a norma das restrições no ponto final. Por isso, estes 42 também serão excluídos das comparações.

Em 132 dos 200 problemas restantes, LANCELOT B pára satisfazendo seu critério de parada relativo a sucesso. Nos 68 problemas restantes, a explicação é a seguinte:

- no problema SREADIN3, LANCELOT B pára por atingir o tempo limite de CPU (10 minutos);
- nos problemas LUKVLI12 e LUKVLI9, LANCELOT B pára por atingir o número máximo de iterações;
- em 65 problemas[13], LANCELOT B pára declarando que o passo é muito pequeno e não gera mudança na função objetivo.

Considerando os 147 problemas nos quais ALGENCAN-A e LANCELOT B obtiveram ponto final viável, observamos que, na maior parte dos casos, os valores de função no ponto final são equivalentes. No entanto,

- em 8 problemas (GMNCASE1, EIGMINC, SINROSNB, ORTHREGF, DTOC2, LUKVLE11, LUKVLI7, GOULDQP3), ALGENCAN-A obteve valor de função menor do que LANCELOT B;
- em 15 problemas (C-RELOAD, TWIRISM1, STEENBRB, STEENBRF, STEENBRG, EIGENB2, EIGENCCO, ORTHREGE, LUKVLE16, LUKVLI16, LUKVLI17, LUKVLI9, BLOCKQP1, BLOCKQP3, BLOCKQP5), LANCELOT B obteve valor de função menor do que ALGENCAN-A.

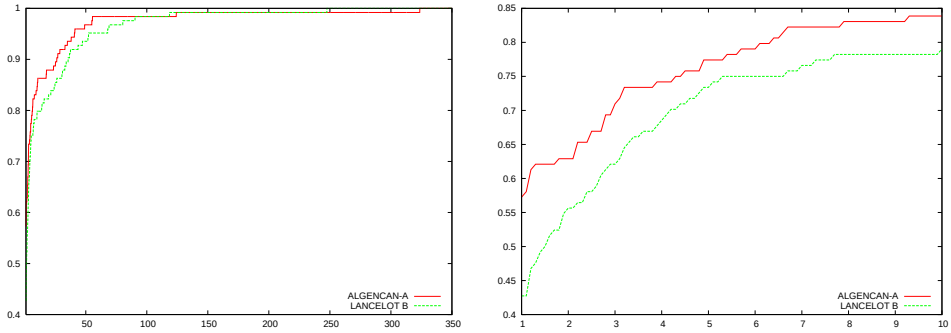
Nos 124 problemas restantes, tanto ALGENCAN-A como LANCELOT B obtiveram pontos viáveis com valores de função equivalentes. Como não dispomos do número de avaliações de função utilizados por LANCELOT B, analisaremos apenas o tempo gasto. Considerando estes 124 problemas:

- ALGENCAN-A foi mais rápido que LANCELOT B (com tolerância de 10%) em 71 casos;

ROBOTARM, READING4, GLIDER, CLNLBEAM, MANNE, DRUGDIS, LUKVLE12, LUKVLE15, LUKVLE17, LUKVLE18, LUKVLI15, LUKVLI14, LUKVLE2, LUKVLI1, LUKVLI2 e READING9.

[12] HAIFAL, GPP, DITTERT, BRIDGEND, TWIRIBG1, LUBRIFC, BRAINPC6, BRAINPC7, BRAINPC8, BRAINPC9, PINENE, LUKVLI11, CVXQP1, CVXQP3, NCVXQP3, NCVXQP9, EG3, LOBSTERZ, CONT6-QQ, AUG2DQP, GAUSSELM, A0NNDNIL, A0NSDSIL, A2NNDNDL, A2NNDNIL, A2NNDNSL, A2NSDSDL, A2NSDSIL, A5NNDNDL, A5NNDNIL, A5NNDNSL, A5NSDSDL, A5NSDSIL, A5NSDSSL, AUG3DQP, CONT5-QP, ALLINQP, QPBAND, QPNBAND, NET4, PORTSNQP e PORTSQP.

[13] SSEBLIN, SSEBNLN, PRIMALC2, C-RELOAD, QPCSTAIR, QPNSTAIR, PRIMALC8, STEERING, YAO, LISWET10, LISWET11, LISWET12, LISWET1, LISWET2, LISWET3, LISWET4, LISWET5, LISWET6, LISWET7, LISWET8, LISWET9, SSNLBEAM, HANGING, TRAINH, OPTCDEG2, OPTCTRL3, OPTCTRL6, HAGER1, HAGER2, HAGER3, HAGER4, HUESTIS, POWELL20, SVANBERG, GRIDNETA, GRIDNETD, GRIDNETG, ORTHREGE, ORTHRDM2, STNQP1, STNQP2, ORTHREGA, LUKVLE11, LUKVLE14, CVXQP2, DTOC6, LUKVLE7, LUKVLE8, LUKVLI7, LUKVLI8, NCVXQP1, NCVXQP4, NCVXQP5, NCVXQP6, NCVXQP7, NCVXQP8, ORTHRGDM, BLOCKQP3, BLOCKQP5, MARINE, A5ENSNDL, A5ESSNDL, AUG2DCQP, AUG2DC e AUG2D.



(a) (b)

Figura 1.13: Curva de perfil de desempenho comparando Algencan-A e Lancelot B usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

- Lancelot B foi mais rápido que Algencan-A (com tolerância de 10%) em 52 casos;
- Algencan-A e Lancelot B gastaram a mesma quantidade de tempo em um caso.

Na Tabela 1.7 temos um resumo da comparação entre Algencan-A e Lancelot B. Quando comparamos qual método gastou menos tempo para resolver os problemas, estamos considerando apenas os problemas para os quais pelo menos um dos métodos gastou mais de 0.01 segundo.

	Algencan-A	Lancelot B
Problemas viáveis	180/316	200/316
Satisfez critérios de convergência	151/180	132/284
Valor de função melhor	8/147	15/147
Tempo menor (10% tolerância)	71/124	52/124

Tabela 1.7: Comparação entre Algencan-A e Lancelot B para problemas selecionados da coleção CUTEr

Considerando os 124 problemas para os quais tanto Algencan-A como Lancelot B chegaram a um ponto viável com mesmo valor de função, construímos um gráfico de perfil de desempenho usamos o tempo de CPU como medida de desempenho (veja Figura 1.13). Observe que Algencan-A se mostrou menos robusto, porém um pouco mais eficiente. Estes resultados nos levam a buscar alternativas melhores para a resolução de problemas de médio e grande porte. Uma delas é utilizar a técnica de aceleração, apresentada em [7].

1.4 Conclusões

Implementamos, em Fortran 77, três novas versões do método de Lagrangiano aumentado Algencan que trabalha explicitamente com as restrições de caixa. A versão original de Algencan (que chamamos de Algencan-G) usa Gencan para resolver os problemas com

restrição de caixa que aparecem a cada iteração. Implementamos as versões que utilizam Betra (Algencan-B) ou Betra-esparso (Algencan-BS) como algoritmo interno. Implementamos ainda uma versão que escolhe automaticamente, entre Gencan, Betra e Betra-esparso, qual algoritmo interno será utilizado para resolver cada problema com restrição de caixa (Algencan-A).

Comparamos o desempenho das 4 versões de Algencan para resolver três problemas coleção La Cumparsita: Ellipsoid, Hard-spheres e Mountain. Os experimentos realizados com as instâncias pequenas do problema Ellipsoid mostram, como era esperado, que Algencan-B é a melhor opção para problemas de pequeno porte, isto é, com até 150 variáveis. Os experimentos com instâncias com mais de 150 variáveis dos problemas Hard-spheres e Mountain mostram que, para problemas com Hessiana do Lagrangiano aumentado (e sua fatoração) esparsa, Algencan-BS tem desempenho superior ao de Algencan-G. Além disso, a escolha automática do algoritmo interno (Algencan-A) apresentou resultados equivalentes ou melhores do que os apresentados por Algencan-BS ou Algencan-G.

Comparamos o desempenho de Algencan-B e Algencan-G para problemas da coleção CUTEr com até 150 variáveis. Apesar de ser um pouco menos robusto, Algencan-B foi mais eficiente do que Algencan-G para resolver estes problemas. Como Gencan, diferentemente de Betra, vem sendo usado há muitos anos para resolver diversos tipos de problemas (veja, por exemplo, [9, 10, 14]), seus parâmetros estão bem calibrados. Este fato justifica a leve perda de robustez de Algencan-B com relação a Algencan-G. Comparamos, também, o desempenho de Algencan-A e Algencan-G na resolução de problemas da coleção CUTEr com mais de 150 variáveis. Para este conjunto de problemas, Algencan-A se mostrou mais eficiente e mais robusto do que Algencan-G.

A partir desses experimentos, podemos concluir que Algencan-B é a melhor alternativa para resolver problemas pequenos, Algencan-BS é melhor para resolver problemas que possuem a matriz Hessiana do Lagrangiano aumentado (e sua fatoração) esparsa e Algencan-G é recomendado para os demais tipos de problemas. Além disso, Algencan-A é uma ótima alternativa quando não se tem informações sobre a estrutura do problema, podendo apresentar desempenho superior às versões que utilizam apenas um algoritmo interno.

Comparamos o desempenho de Algencan-B com Lancelot B para resolver os problemas pequenos (até 150 variáveis) de CUTEr. Algencan-B se mostrou mais robusto e eficiente, o que mostra que este é um método competitivo. Por fim, comparamos o desempenho de Algencan-A com Lancelot B para resolver os problemas grandes (mais 150 variáveis) de CUTEr. Algencan-A se mostrou menos robusto, porém um pouco mais eficiente. Estes resultados nos levam a buscar alternativas melhores para a resolução de problemas de médio e grande porte. Uma delas é utilizar a técnica de aceleração, apresentada em [7].

Agora que temos implementações de métodos de Lagrangiano aumentado que deixam as restrições de caixa no nível inferior e penalizam as demais restrições, estamos interessados em uma implementação deste método que deixe também as restrições lineares no nível inferior, já que restrições deste tipo possuem propriedades interessantes, que as tornam mais simples do que restrições não-lineares. Para isso, necessitamos de uma implementação eficiente de um método para minimizar funções suaves sujeitas a restrições lineares e de caixa. Um método para resolução deste problema, é apresentado no Capítulo 2.

Capítulo 2

Novo método para minimização com restrições lineares

Estamos interessados em resolver o seguinte problema:

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & x \in \Omega = \{x \in I\!R^n \mid Ax = b,\ Cx \leq d,\ \ell \leq x \leq u\}, \end{array} \tag{2.1}$$

onde f é uma função em $\mathcal{C}^2$, $x \in I\!R^n$, $A \in I\!R^{r\times n}$, $b \in I\!R^r$, $C \in I\!R^{m\times n}$, $d \in I\!R^m$, $\ell \in I\!R^n$, $u \in I\!R^n$ e $\ell \leq u$.

Métodos do tipo Lagrangiano aumentado são muito utilizados para minimização de funções sujeitas a restrições gerais. Como vimos no Capítulo 1, um método eficiente para resolver (2.1) pode ser utilizado para resolver os subproblemas de métodos do tipo Lagrangiano aumentado que penalizam somente as restrições não-lineares, podendo torná-los mais eficientes. A experiência positiva de métodos de restrições ativas como Betra [3] e Gencan [8], já usados como subalgoritmos de Algencan, nos leva a buscar métodos como estes para resolver problemas com restrições lineares.

Na Seção 2.1 introduzimos um método de restrições ativas para resolver (2.1), baseado nos métodos apresentados em [3, 8]. Na Seção 2.2, mostramos como podem ser calculadas as projeções de pontos em um conjunto de restrições lineares. Na Seção 2.3 introduzimos o método do Gradiente Espectral Projetado Parcial, utilizado como subalgoritmo do método de restrições ativas proposto para resolver o problema (2.1). Na Seção 2.4 apresentamos dois algoritmos "irrestritos" utilizados para resolver problema de minimização sujeita a restrições lineares de igualdade, que também serão usados como subalgoritmo do método de restrições ativas para resolver (2.1). Na Seção 2.5 mostramos a teoria de convergência do método proposto na Seção 2.1. Na Seção 2.6 apresentamos detalhes de implementação das duas versões do novo método. Na Seção 2.7 apresentamos os resultados numéricos obtidos pela aplicação das versões dos métodos novos aos problemas da coleção Cuter [15]. Comparamos os resultados com Algencan-G, Algencan-B, ve11 [47], Minos [41], Ipopt [54] e Lancelot B [18, 19, 20], outros conhecidos métodos para a resolução de (2.1). Apresentamos, também, os resultados da aplicação das versões dos métodos novos à resolução do problema de

estimar a espessura e constantes ópticas de filmes finos [5, 6, 16, 39]. Finalmente, na Seção 2.8 apresentamos as conclusões obtidas.

2.1 Método de restrições ativas

De maneira análoga à que é feita em BETRA e GENCAN, dividimos o conjunto viável do problema (2.1), $\Omega = \{x \in I\!R^n \mid Ax = b,\ Cx \leq d,\ \ell \leq x \leq u\}$, em faces disjuntas. Para todo $I \subset \{1, 2, \ldots, m\}$ e $J \subset \{1, 2, \ldots, 2n\}$ definimos

$$\begin{aligned} F_{I,J} = \ & \{x \in \Omega \mid a^k x = b_k, 1 \leq k \leq r, \\ & c^i x = d_i \ \text{ se } i \in I,\ c^i x < d_i \ \text{ se } i \notin I, \\ & x_j = l_j \text{ se } j \in J,\ x_j = u_j \text{ se } n + j \in J,\ l_j < x_j < u_j \ \text{ caso contrário}\}, \end{aligned}$$

onde a^k é a k-ésima linha da matriz A e c^i é a i-ésima linha da matriz C.

O poliedro Ω é a união das faces. As restrições lineares de igualdade, as restrições i tais que $i \in I$ e j tais que $j \in J$ são chamadas restrições *ativas*. Definimos $V_{I,J}$ o menor espaço afim que contém $F_{I,J}$ e $S_{I,J}$ o subespaço linear paralelo a $V_{I,J}$. Definimos o gradiente projetado da seguinte maneira:

$$g_P(z, \delta) = P_{\Omega(z,\delta)}(z - \nabla f(z)) - z,$$

com $\Omega(z, \delta) = \{x \in I\!R^n \mid Ax = b,\ \ell \leq x \leq u,\ C^j x \leq d_j,\ \forall j \in I(z, \delta)\}$ e $I(z, \delta) = \{j \mid C^j z - d_j \geq -\delta_j\}$. Quer dizer, $\Omega(z, \delta)$ é o conjunto das restrições que são satisfeitas por igualdade ou que estão relativamente próximas de serem satisfeitas por igualdade. Note que, para $\delta = \infty$, temos que $\Omega(z, \delta) = \Omega$. $P_{\Omega(x,\delta)}(x)$ é a projeção ortogonal do ponto x no conjunto $\Omega(x, \delta)$.

Note que o método de restrições ativas apresentado na Seção 1.1 (Algoritmo 1.4) utiliza muitas projeções a cada iteração. No caso de problemas com restrições de caixa, a projeção é trivial. Já no caso de problemas com restrições lineares, a projeção passa a ser um problema de minimização (veja Seção 2.2). Assim, para resolver o problema (2.1), utilizamos um método de restrições ativas similar ao Algoritmo 1.4, mas com algumas mudanças para diminuir o número e o custo das projeções:

- Eliminamos o teste para decidir se a face atual deve ser abandonada (teste 1.8): usando o Algoritmo 1.4, o gradiente projetado g_P deve ser calculado a cada iteração para realizar o teste (1.8). Como, na presença de restrições lineares, o custo computacional de um algoritmo "irrestrito" para minimização na face (Passo 4 do Algoritmo 1.4) é menor do que o custo da projeção, decidimos permanecer na mesma face até que seja encontrado o minimizador desta face ou até que uma borda seja atingida. Assim evitamos o cálculo de g_P e g_I a cada iteração.

- Fazendo apenas a mudança descrita no item acima, temos um inconveniente. É possível que, a cada iteração, cheguemos à borda da face e acrescentemos restrições ao conjunto de restrições ativas. Depois de muitas iterações deste tipo, ao encontrarmos o minimizador de uma face de dimensão menor, poderíamos descobrir que estávamos na face errada e todo o trabalho gasto nestas iterações poderia ter sido evitado pelo uso de

iterações de abandono de face. Para evitar que isto aconteça, decidimos sair da face se acrescentamos restrições ao conjunto de ativas por 20 iterações consecutivas.

- Para tentar diminuir o custo de cada projeção, utilizamos como critério de convergência no Passo 3 do Algoritmo 1.4, $\|g_P(z,\delta)\| = 0$, para algum $\delta > 0$. No caso de $\delta = \infty$, temos o mesmo critério de convergência do Algoritmo 1.4. Se utilizamos algum δ finito, podemos diminuir o número de restrições consideradas na projeção e, possivelmente, diminuir seu custo. Quanto menor o valor de δ, menor pode ser o número de restrições consideradas na projeção. No entanto, para garantir a convergência do método (veja Seção 2.5), é necessário que $\delta > 0$.

- No Passo 4 do Algoritmo 1.4, quando decidimos abandonar a face, passamos a utilizar uma iteração do método do Gradiente Espectral Projetado Parcial (PSPG). A diferença entre o SPG tradicional e o PSPG é o fato de PSPG projetar apenas em um subconjunto das restrições lineares, possivelmente diminuindo o custo de suas projeções.

Com estas mudanças, o método de restrições ativas para resolver (2.1) pode ser escrito da seguinte maneira:

Algoritmo 2.1 (Método de restrições ativas para problemas com restrições lineares)

Dados $x_0 \in I\!R^n$ *e* $\delta^1 \geq \delta_{\min} > 0$.

Passo 1. Calcule $x_1 = P_\Omega(x_0)$.

Se é possível calcular x_1 então faça $k \leftarrow 1$ e $bd \leftarrow 0$. Senão, pare declarando *"Problema inviável"*.

Passo 2. *Atingiu borda por 20 iterações seguidas. Sai da face*

Se $bd = 20$ então vá para o Passo 3. Senão, vá para o Passo 4.

Passo 3. *Critério de convergência*

Se $g_P(x_k, \delta^k) = 0$ então pare e devolva x_k como solução. Senão, use uma iteração do PSPG para calcular x_{k+1}, faça $bd \leftarrow 0$, $k \leftarrow k + 1$ e vá para o Passo 4.

Passo 4. x_k *é um minimizador na face*

Se $P_{S_{I,J}}(-\nabla f(x_k)) = 0$ então volte para o Passo 3.

Passo 5. *Permanece na mesma face*

Suponha, sem perda de generalidade, que as variáveis livres em $F_{I,J}$ são $\bar{x} = (x_1, ..., x_s)^T$ e que as variáveis restantes estão fixas em $\hat{x}_{s+1}, ..., \hat{x}_n$ ($\hat{x}_i \in \{l_i, u_i\}$, para $s+1 \leq i \leq n$).

Defina $\bar{f}(\bar{x}) = f(x_1, ..., x_s, \hat{x}_{s+1}, ..., \hat{x}_n)$, para todo $x_1, ..., x_s \in I\!R$. Defina $\bar{\Omega} = \{x \in I\!R^s \mid (x, \hat{x})^T \in F_{I,J}\}$, $\bar{A}$ e $\bar{b}$ correspondentes às restrições ativas de $\bar{\Omega}$ em x_k.

Use um algoritmo "irrestrito" que minimize $\bar{f}$ sujeita a $\bar{A}\bar{x} = \bar{b}$ para calcular $x_{k+1} \in \bar{F}_{I,J}$ com x_k como ponto inicial.

Passo 6. *Ponto* x_{k+1} *está na borda*

Se o algoritmo "irrestrito" terminar sua execução com x_{k+1} declarando *"Ponto na borda"* então faça $bd \leftarrow bd + 1$. Senão, faça $bd \leftarrow 0$.

Passo 7. Faça $k \leftarrow k + 1$ e volte para o Passo 2.

Para que o Algoritmo 2.1 esteja totalmente definido, precisamos mostrar como é calculada a projeção de um ponto nas restrições lineares, definir o método PSPG e escolher um algoritmo "irrestrito" para ser usado no Passo 5. Isso será feito nas seções a seguir.

2.2 Cálculo da projeção

Note que projetar um ponto $\bar{x} \in I\!R^n$ em um conjunto $\bar{\Omega} = \{x \in I\!R \mid Ax = b,\ \bar{C}x \leq \bar{d},\ \ell \leq x \leq u\}$ significa encontrar um ponto $\bar{x}^*$ que seja solução do seguinte problema:

$$\begin{array}{ll} \text{Minimizar} & \frac{1}{2}\|\bar{x} - x\|_2^2 \\ \text{sujeita a} & Ax = b, \\ & \bar{C}x \leq \bar{d}, \\ & \ell \leq x \leq u, \end{array} \tag{2.2}$$

ou, equivalentemente,

$$\begin{array}{ll} \text{Minimizar} & \frac{1}{2}x^T I x - \bar{x}^T x \\ \text{sujeita a} & Ax = b, \\ & \bar{C}x \leq \bar{d}, \\ & \ell \leq x \leq u, \end{array} \tag{2.3}$$

onde I é a matriz identidade em $I\!R^{n\times n}$, $\bar{C} \in I\!R^{\bar{m}\times n}$ e $\bar{d} \in I\!R^{\bar{m}}$ são as restrições lineares de desigualdade utilizadas na projeção.

Como este é um problema de programação quadrática convexa, existem vários algoritmos conhecidos que podem ser utilizados para resolver (2.3). Um algoritmo clássico para a resolução deste problema é o proposto por Goldfarb e Idnani [27, 45].

Uma outra maneira de calcular a projeção é considerar o problema dual de (2.3), dado por

$$\begin{array}{ll} \text{Minimizar} & \frac{1}{2}y^T \tilde{A}\tilde{A}^T y + (\tilde{b} - \tilde{A}\bar{x})^T y \\ \text{sujeita a} & y \geq 0, \end{array} \tag{2.4}$$

onde $\tilde{A} \in I\!R^{(2r+\bar{m}+2n)\times n}$, $\tilde{A} = \begin{bmatrix} A \\ -A \\ \bar{C} \\ -I \\ I \end{bmatrix}$ e $\tilde{b} \in I\!R^{2r+\bar{m}+2n}$, $\tilde{b} = \begin{bmatrix} b \\ -b \\ \bar{d} \\ -l \\ u \end{bmatrix}$. Assim, qualquer algoritmo para minimização de quadráticas em caixas pode ser utilizado para encontrar a solução y^* de (2.4). Dado y^*, a solução primal (ou seja, o ponto $\bar{x}^*$ projeção de $\bar{x}$ em $\bar{\Omega}$) é dada por $\bar{x}^* = \bar{x} - \tilde{A}^T y^*$.

Embora possa sofrer de problemas de condicionamento, esta formulação é mais interessante para o caso esparso, para o qual não há muitas implementações de métodos para resolver o problema de programação quadrática (2.3). No caso de problemas esparsos, é necessário um método para resolver (2.4) que não fatore a matriz Hessiana do problema. Métodos baseados em gradientes conjugados são uma boa alternativa, pois calculam apenas o produto da Hessiana por um vetor.

2.3 Método do Gradiente Espectral Projetado Parcial

Quando o Algoritmo 2.1 decide mudar de face, ou seja, decide mudar o conjunto de restrições ativas corrente, usamos o método do Gradiente Espectral Projetado Parcial (chamado de PSPG) para definir qual será a nova face a ser trabalhada. Em outras palavras, usamos o método do Gradiente Espectral Projetado Parcial, desenvolvido neste trabalho especialmente para este propósito, para escolher quais restrições devem entrar no conjunto das restrições ativas e quais devem sair deste conjunto.

O método do Gradiente Espectral Projetado Parcial é usado para resolver o seguinte problema:

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & x \in \Omega, \end{array}$$

onde f é uma função em $\mathcal{C}^1$, $x \in I\!R^n$ e Ω é um conjunto convexo fechado. No nosso caso, $\Omega = \{x \in I\!R^n \mid Ax = b,\ Cx \leq d,\ \ell \leq x \leq u\}$.

A idéia do algoritmo é a seguinte: na iteração k, calculamos uma direção de descida p_k e damos um passo α_k tal que $x_{k+1} = x_k + \alpha_k p_k \in \Omega$ satisfaça a condição de Armijo. A direção p_k é calculada com base na projeção ortogonal de $-\lambda_k^{\text{PSPG}} \nabla f(x_k)$ no conjunto $\Omega(x_k, \delta)$, onde λ_k^{PSPG} (chamado de *passo espectral*) é um escalar calculado a partir de informações fornecidas pelo ponto atual x_k e pelo ponto x_{k-1} da iteração anterior.

Denotamos por Ω^0 o conjunto de pontos interiores de Ω. Os passos de uma iteração de PSPG são dados por:

Algoritmo 2.2 (Iteração de PSPG)

Seja k a iteração atual. Suponha que $\nabla f(x_k) \neq 0$. Dados $x_k \in \Omega^0$, $0 < \lambda_{\min}^{PSPG} < \lambda_{\max}^{PSPG} < \infty$, $\gamma \in (0,1)$, $\delta^k \geq \delta_{\min} > 0$.

Passo 1. *Cálculo do λ_k^{PSPG}*

Se $k = 0$ ou $(x_k - x_{k-1})^T(\nabla f(x_k) - \nabla f(x_{k-1})) \leq 0$ então faça $\lambda_k^{\text{PSPG}} = 1$. Senão, calcule $s_k = x_k - x_{k-1}$, $y_k = \nabla f(x_k) - \nabla f(x_{k-1})$ e faça

$$\lambda_k^{\text{PSPG}} = \min\left\{\lambda_{\max}^{\text{PSPG}}, \max\left\{\lambda_{\min}^{\text{PSPG}}, \frac{{s_k}^T s_k}{{s_k}^T y_k}\right\}\right\}.$$

Passo 2. Calcule $p_k^{\text{PSPG}} = P_{\Omega(x_k,\delta^k)}(x_k - \lambda_k^{\text{PSPG}} \nabla f(x_k)) - x_k$.

Passo 3. Calcule $\alpha_{\max} = \max\{\alpha \in [0,1] \mid [x_k, x_k + \alpha p_k^{\text{PSPG}}] \subset \Omega\}$.

Passo 4. Faça $\alpha_{\text{tent}} \leftarrow \alpha_{\max}$.

Passo 5. Se

$$f(x_k + \alpha_{\text{tent}} p_k) \leq f(x_k) + \gamma \alpha_{\text{tent}} \nabla f(x_k)^T p_k^{\text{PSPG}}$$

então faça $\alpha_k = \alpha_{\text{tent}}$ e termine a iteração com $x_{k+1} = x_k + \alpha_k p_k^{\text{PSPG}}$. Senão, escolha $\alpha_{\text{novo}} \in [0.1\alpha_{\text{tent}}, 0.9\alpha_{\text{tent}}]$, faça $\alpha_{\text{tent}} \leftarrow \alpha_{\text{novo}}$ e Repita o Passo 5.

Observações:

- No Passo 5, escolhemos α_{novo} usando interpolação quadrática uni-dimensional com salvaguardas. A quadrática interpolante é dada por $q(w)$, com $q(0) = f(x_k)$, $q(\alpha_k) = f(x_k + \alpha_k p_k^{\text{PSPG}})$ e $\nabla q(0) = \nabla f(x_k)^T p_k^{\text{PSPG}}$. Definimos $\bar{\alpha}_{\text{novo}}$ como o minimizador de $q(\cdot)$ com salvaguardas.

  Passo 1. Faça

$$\bar{\alpha}_{\text{novo}} \leftarrow -\frac{\alpha_{\text{tent}}^2 \nabla f(x_k)^T p_k^{\text{PSPG}}}{2(f(x_k + \alpha_{\text{tent}} p_k^{\text{PSPG}}) - f(x_k) - \alpha_{\text{tent}} \nabla f(x_k)^T p_k^{\text{PSPG}})}.$$

  Passo 2. *Salvaguarda de α_{novo}*

  Se $\bar{\alpha}_{\text{novo}} \geq 0.1\alpha_{\text{tent}}$ e $\bar{\alpha}_{\text{novo}} \leq 0.9\alpha_{\text{tent}}$ então faça $\alpha_{\text{novo}} \leftarrow \bar{\alpha}_{\text{novo}}$. Senão, faça $\alpha_{\text{novo}} \leftarrow 0.5\ \alpha_{\text{tent}}$.

- Como em [3, 8], definimos $\lambda_{\min}^{\text{PSPG}} = 10^{-10}$, $\lambda_{\max}^{\text{PSPG}} = 10^{10}$.

Note que, para $\delta = \infty$, o Algoritmo 2.2 se reduz ao método SPG.

2.4 Algoritmos "irrestritos"

Estamos agora interessados em resolver o seguinte problema:

$$\begin{array}{ll} \text{Minimizar} & \bar{f}(x) \\ \text{sujeita a} & \bar{A}x = \bar{b}, \end{array} \tag{2.5}$$

onde $\bar{f}$ é uma função em $\mathcal{C}^2$, $x \in I\!R^s$, $\bar{b} \in I\!R^t$ e $\bar{A} \in I\!R^{t \times s}$.

Ao trabalhar com o problema (2.5), trataremos apenas do caso em que a matriz $\bar{A}$ possui linhas linearmente independentes. Isso é razoável na prática, pois quando alguma linha de $\bar{A}$ depende linearmente de outra, ou a linha em questão pode ser eliminada sem nenhuma alteração da solução de (2.5) ou a região viável é vazia, não havendo solução para o problema.

Antes de prosseguir, damos algumas definições: Z é uma matriz de $I\!R^{s \times (s-t)}$ cujas colunas formam uma base para núcleo do espaço gerado pelas linhas da matriz $\bar{A}$; e Y é uma matriz de $I\!R^{s \times t}$ cujas colunas formam uma base para a imagem do espaço formado pelas linhas de $\bar{A}$. Assim, temos que $\bar{A}Z = 0$ e $\bar{A}Y$ é não-singular.

Note que, uma vez escolhidas as matrizes Z e Y, todo vetor x em $I\!R^s$ possui uma única representação da forma

$$x = Yx_y + Zx_z,$$

com $x_y \in I\!R^t$ e $x_z \in I\!R^{(s-t)}$. Em particular, tomando x^* solução de (2.5), temos $x^* = Yx_y^* + Zx_z^*$. Como x^* é viável, vale que

$$\bar{A}x^* = \bar{A}(Yx_y^* + Zx_z^*) = \bar{b}.$$

Como $\bar{A}Z = 0$, temos

$$x_y^* = (\bar{A}Y)^{-1}\bar{b},$$

pois $\bar{A}Y$ é não-singular.

Assim, a solução x^* de (2.5) (assim como qualquer ponto viável de (2.5)) pode ser vista como a soma de uma solução particular para o sistema linear $\bar{A}x = \bar{b}$ (fazendo-se, por exemplo, $x_z^* = 0$) e de um ponto no núcleo do espaço gerado pelas linhas de $\bar{A}$ (ou seja, Zx_z^*). Desta forma, reduzimos o problema restrito original (2.5) para um problema irrestrito de $s - t$ variáveis, dado por

$$\text{Minimizar} \quad \bar{f}_1(x_z) = \bar{f}(Y(\bar{A}Y)^{-1}\bar{b} + Zx_z). \tag{2.6}$$

Note que o gradiente e a Hessiana de $\bar{f}_1$ são dados por

$$\nabla \bar{f}_1(x_z) = Z^T \nabla \bar{f}(x)$$

e

$$\nabla^2 \bar{f}_1(x_z) = Z^T \nabla^2 \bar{f}(x) Z.$$

Formularemos, então, um algoritmo para resolver (2.6) que gera uma seqüência de pontos viáveis. Para manter a viabilidade de cada ponto gerado pelo algoritmo, a cada iteração k partimos de um ponto x_k viável e calculamos um passo p_k tal que

$$\bar{A}p_k = 0.$$

Desta forma, temos que o ponto $x_{k+1} = x_k + \alpha_k p_k$ é viável, pois

$$\bar{A}x_{k+1} = \bar{A}(x_k + \alpha_k p_k) = \bar{A}x_k + \alpha_k \bar{A}p_k = \bar{A}x_k = \bar{b},$$

já que x_k também é viável. Note que

$$\bar{A}p_k = 0 \Leftrightarrow p_k = Zp^z,$$

para algum vetor $p^z \in I\!R^{(s-t)}$.

Podemos agora escrever algoritmos para resolver (2.6) baseado nos algoritmos "irrestritos" apresentados na Seção 1.1. Na Seção 2.4.1 apresentamos a adaptação do Algoritmo 1.5 para a resolução do problema (2.6). Na Seção 2.4.2 apresentamos a adaptação do Algoritmo 1.6 para a resolução do problema (2.6). Na Seção 2.4.3 mostramos como pode ser calculada a matriz Z, usada em ambos os algoritmos.

2.4.1 Algoritmo "irrestrito" com regiões de confiança

Na Seção 1.1, apresentamos o algoritmo "irrestrito" baseado em regiões de confiança introduzido em [3]. Vamos adaptá-lo ao caso em que o problema original possui restrições lineares.

Algoritmo 2.3 (Iteração do Algoritmo 1.5 na presença de restrições lineares)

Seja k a iteração atual. Seja $\bar{\Omega}$ o conjunto de restrições de igualdade e desigualdade do problema original. Dados $\bar{A} \in I\!R^{t\times s}$, x_k tal que $\bar{A}x_k = \bar{b}$, $Z \in I\!R^{s\times(s-t)}$ tal que $\bar{A}Z = 0$, $\Delta_{\min} > 0$, $\Delta_k \geq \Delta_{\min}$ e $\sigma > 0$.

Passo 1. Se $Z^T \nabla \bar{f}(x_k) = 0$ então pare e devolva x_k como solução declarando *"Ponto estacionário de primeira ordem"*.

Passo 2. Calcule Δ_{borda} a distância de x_k à borda da face.

Se $\Delta_{\text{borda}} < 2\Delta_{\min}$ então faça uma iteração de SPG restrito a $F_{I,J}$ (face atual) para calcular x_{k+1} e pare.

Passo 3. Calcule s_k uma solução aproximada de

$$\begin{array}{ll} \text{Minimizar} & \varphi(w) \equiv \frac{1}{2} w^T Z^T \nabla^2 \bar{f}(x_k) Z w + (Z^T \nabla \bar{f}(x_k))^T w \\ \text{sujeita a} & \|w\| \leq \Delta_k. \end{array} \tag{2.7}$$

Passo 4. Se $\varphi(s_k) = 0$ então pare e devolva x_k como solução declarando *"Ponto estacionário de segunda ordem"*. Senão, calcule $p_k = Z s_k$.

Passo 5. Se $x_k + p_k$ não pertence à face atual então calcule

$$\alpha_{\max} = \max\{\alpha \in [0,1] \mid [x_k, x_k + \alpha p_k] \subset \bar{\Omega}\}.$$

Se $\bar{f}(x_k + \alpha_{\max} p_k) < \bar{f}(x_k)$ então faça $x_{k+1} = x_k + \alpha_{\max} p_k$ e pare declarando *"Iterando na borda"*. Senão, faça

$$\Delta_k = \Delta_{\min} + 0.7 \left(\frac{\Delta_{\text{borda}}}{1 + \sigma} - \Delta_{\min} \right)$$

e volte para o Passo 3.

Passo 6. Calcule $\rho_k = \frac{\bar{f}(x_k + p_k) - \bar{f}(x_k)}{\varphi(s_k)}$.

Passo 7. Se $\rho_k \leq 0.1$ então faça $\Delta_k = \Delta \in [0.1\|p_k\|, 0.9\|p_k\|]$ e Volte para o Passo 2.

Passo 8. Faça $x_{k+1} = x_k + p_k$.

Passo 9. Escolha $\Delta_{k+1} \geq \Delta_{\min}$.

Observações:

- Valem as mesmas observações feitas na apresentação do Algoritmo 1.5, lembrando que, no Passo 1, o critério passa a ser $\|Z^T \nabla f(x_k)\|_\infty \leq \varepsilon$ e, no Passo 4, o critério passa a ser $|\varphi(s_k)| \leq \varepsilon_\varphi$.
- Note que, no Passo 5, quando reduzimos o raio da região de confiança, utilizamos o fator 0.7 em vez de 0.9 (como usado no Algoritmo 1.5). Isso foi feito porque, na presença de restrições lineares, temos mais erros numéricos, já que não estamos lidando com a função original, mas com sua projeção no espaço reduzido.

Na Seção 2.4.3 apresentamos como pode ser calculada a matriz Z. Assim, a descrição do Algoritmo 2.3 se torna completa.

2.4.2 Algoritmo "irrestrito" com busca linear

Uma outra possibilidade para definir um algoritmo "irrestrito" para resolver o problema (2.6) é adaptar o Algoritmo 1.6:

Algoritmo 2.4 (Iteração do Algoritmo 1.6 na presença de restrições lineares)

Seja k a iteração atual. Seja $\bar{\Omega}$ o conjunto de restrições de igualdade e desigualdade do problema original. Dados $\bar{A} \in I\!R^{t\times s}$, x_k tal que $\bar{A}x_k = \bar{b}$, $Z \in I\!R^{s\times(s-t)}$ tal que $\bar{A}Z = 0$, $\theta \in (0,1)$ e $\gamma \in (0,1)$.

Passo 1. Se $Z^T\nabla\bar{f}(x_k) = 0$ então pare e devolva x_k como solução declarando *"Ponto estacionário de primeira ordem"*.

Passo 2. Calcule um vetor não-nulo $p^z \in I\!R^{(s-t)}$ que satisfaça

$$(Z^T\nabla\bar{f}(x_k))^T p^z \leq -\theta\|Z^T\nabla\bar{f}(x_k)\|\|p^z\|.$$

Calcule a direção $p_k = Zp^z$.

Passo 3. Calcule $\alpha_{\max} = \max\{\alpha \in [0,1] \mid [x_k, x_k + \alpha p_k] \subset \bar{\Omega}\}$.

Se $\alpha_{\max} < 1$ e $\bar{f}(x_k + \alpha_{\max}p_k) < \bar{f}(x_k)$ então faça $x_{k+1} = x_k + \alpha_{\max}p_k$ e pare declarando *"Iterando na borda"*.

Passo 4. Calcule $\alpha_k < \alpha_{\max}$ para o qual valha

$$\bar{f}(x_k + \alpha_k p_k) \leq \bar{f}(x_k) + \gamma\alpha_k(Z^T\nabla\bar{f}(x_k))^T p^z.$$

Passo 5. Faça $x_{k+1} = x_k + \alpha_k p_k$.

Valem todas as observações feitas para o Algoritmo 1.6, lembrando que, neste caso, o gradiente é dado por $Z^T\nabla\bar{f}$ e a Hessiana por $Z^T\nabla^2\bar{f}Z$. Para que a descrição do Algoritmo 2.4 esteja completa, restará apenas mostrar como calcular a matriz Z, o que é feito na Seção 2.4.3.

2.4.3 Cálculo da base do núcleo

Apresentamos aqui duas técnicas que podem ser utilizadas para calcular a matriz Z usada nos Algoritmos 2.3 e 2.4.

2.4.3.1 Fatoração LQ

A primeira idéia para o cálculo da matriz Z é o uso da fatoração LQ da matriz $\bar{A}$. Nesta fatoração, que gasta $O(st^2)$ operações, temos uma matriz ortogonal $Q \in I\!R^{s\times s}$ tal que

$$\bar{A}Q = (L \quad 0),$$

onde $L \in I\!R^{t\times t}$ é triangular inferior. Como estamos supondo que as linhas da matriz $\bar{A}$ são linearmente independentes, a matriz L é não-singular.

Escrevendo a matriz Q como $Q = [Q_1; Q_2]$, com $Q_1 \in I\!R^{s\times t}$ e $Q_2 \in I\!R^{s\times(s-t)}$, temos que

$$\bar{A}Q_1 = L \text{ e } \bar{A}Q_2 = 0.$$

É fácil ver que as colunas de Q_1 e Q_2 formam, respectivamente, uma base ortogonal para a imagem e o núcleo do espaço gerado pelas linhas de $\bar{A}$. Assim, uma escolha natural para a matriz Z é Q_2.

Note que, se utilizarmos técnicas do tipo Newton para calcular o passo p^z (Passo 2 do Algoritmo 2.4), é importante levar em conta o condicionamento da Hessiana reduzida $Z^T\nabla^2\bar{f}(x_k)Z$. Neste caso, temos que

$$\text{cond}(Z^T\nabla^2\bar{f}(x_k)Z) \leq \text{cond}(\nabla^2\bar{f}(x_k))(\text{cond}(Z))^2.$$

Assim, como este limitante é justo (pode valer a igualdade), se o número de condição da matriz Z fosse grande, a matriz Hessiana reduzida $Z^T\nabla^2\bar{f}(x_k)Z$ poderia ser mal-condicionada, independentemente do número de condição da matriz Hessiana $\nabla^2\bar{f}(x_k)$. No entanto, utilizando a fatoração LQ para determinar Z, temos $\text{cond}(Z) = 1$. Ou seja,

$$\text{cond}(Z^T\nabla^2\bar{f}(x_k)Z) \leq \text{cond}(\nabla^2\bar{f}(x_k)),$$

o que mostra que, neste caso, o condicionamento do problema (2.6) não é pior que o condicionamento do problema (2.5). Portanto, uma grande vantagem do uso desta técnica é o fato de que esta escolha de Z não piora o condicionamento do problema de encontrar uma solução para (2.6).

Para formar a matriz Q usando a fatoração LQ é necessário calcular o produto de transformações ortogonais usadas para triangularizar a matriz $\bar{A}$. Na implementação de métodos que usam a matriz Z somente em produtos matriz-vetor, pode não ser eficiente calcular a matriz Q (e, conseqüentemente, Z) explicitamente, principalmente quando o número de restrições é pequeno. Neste caso é mais interessante armazenar as transformações ortogonais de forma compacta e aplicá-las ao vetor apropriado para calcular o produto matriz-vetor sempre que necessário.

2.4.3.2 Redução de variáveis

Apesar das vantagens do uso da fatoração LQ para o cálculo da matriz Z, esta é uma operação computacionalmente custosa e que não aproveita uma eventual esparsidade de $\bar{A}$. Uma alternativa é a técnica de redução de variáveis.

Tomando a matriz $\bar{A}$ com linhas linearmente independentes, podemos particioná-la em

$$\bar{A} = (V \quad U),$$

onde $V \in I\!R^{t\times t}$ é não-singular e $U \in I\!R^{t\times(s-t)}$. Usando esta partição de $\bar{A}$, a matriz Z pode ser definida como

$$Z = \begin{pmatrix} -V^{-1}U \\ I \end{pmatrix},$$

onde I é a matriz identidade $(s-t)\times(s-t)$. Note que

$$\bar{A}Z = (V \quad U)\begin{pmatrix} -V^{-1}U \\ I \end{pmatrix} = -VV^{-1}U + U = 0,$$

como desejamos.

Geralmente a matriz Z, tal como descrita acima, não é armazenada explicitamente. Produtos matriz-vetor envolvendo Z e Z^T podem ser obtidos resolvendo sistemas lineares envolvendo V e V^T. A grande dificuldade desta técnica é encontrar uma maneira eficiente de particionar $\bar{A}$ de forma que a matriz V escolhida fique relativamente bem-condicionada.

2.5 Teoria de convergência

Para simplificar a teoria de convergência, vamos escrever o problema (2.1) como

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & x \in \Omega = \{x \in I\!R^n \mid C_i(x) \leq 0, i = 1, \ldots, p\}, \end{array} \tag{2.8}$$

com

$$C_i(x) = (a^i)^T x - b_i,\ a^i = (a^i_1, ..., a^i_n)^T.$$

Vamos supor que Ω é compacto. Claramente, os problemas (2.1) e (2.8) são equivalentes. Dado $I \subset \{1, \ldots p\}$, definimos F_I de maneira análoga a $F_{I,J}$, ou seja,

$$F_I = \{x \in \Omega \mid C_i(x) = 0 \text{ se e somente se } i \in I\}.$$

Note que Ω é a união dos conjuntos F_I, para $I \subset \{1, \ldots, p\}$. Ainda, $F_I \neq F_J$ implica que $F_I \cap F_J = \emptyset$. O menor espaço afim que contém a face não-vazia F_I será denotado por V_I e o subespaço linear paralelo a V_I será denotado por S_I.

Dado $z \in \Omega$ e $\delta \in [0, \infty]^p$, definimos

$$\Omega(z, \delta) = \{x \in I\!R^n \mid C_i(x) \leq 0 \text{ para todo } i \in I(z, \delta)\},$$

onde

$$I(z, \delta) = \{i \mid C_i(z) \geq -\delta_i\}.$$

Claramente, $\Omega \subset \Omega(z, \delta)$ e $\Omega(z, \infty) = \Omega$.

Observe que, se $z \in \Omega$ satisfaz as condições KKT de (2.8) então z também satisfaz as condições KKT de

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & x \in \Omega(z, \delta), \end{array} \tag{2.9}$$

para todo $\delta \geq 0$. Reciprocamente, se z satisfaz as condições KKT de (2.9) para algum $\delta \geq 0$, então z satisfaz as condições KKT de (2.8).

Dado $z \in \Omega$, definimos

$$g_P(z) = P_\Omega(z - \nabla f(z)) - z.$$

Como visto na Seção 2.1, definimos

$$g_P(z,\delta) = P_{\Omega(z,\delta)}(z - \nabla f(z)) - z.$$

Além disso, para todo $\sigma > 0$, definimos

$$g_P(z,\delta,\sigma) = P_{\Omega(z,\delta)}(z - \sigma\nabla f(z)) - z.$$

Finalmente, se $z \in F_I \subset \Omega$, definimos

$$g_S(z) = P_{S_I}(-\nabla f(z)) = -Z^T \nabla f(z).$$

Note que vale que

$$g_P(z) = g_P(z,\infty) = g_P(z,\infty,1)$$

e

$$g_P(z,\delta) = g_P(z,\delta,1).$$

Dados $\delta \in [0,\infty]^p$ e $\sigma > 0$, satisfazer as condições KKT para $z \in \Omega$ é equivalente a $g_P(z,\delta,\sigma) = 0$. Ainda, se $\delta_i^k \geq \delta_{\min} > 0$ para todo $i = 1,\ldots,p$, $k \in I\!N$, $z_k \to z$ e z é um ponto KKT, temos que $\|g_P(z_k,\delta^k)\| \to 0$. Isso justifica o uso de $\|g_P(z_k,\delta^k)\|$ como critério de parada do método.

Para provar a convergência do Algoritmo 2.1, precisamos definir e apresentar o teorema de convergência de um algoritmo auxiliar. Primeiramente, vejamos a definição do método de Métrica Variável Inexata [13]. Este método é usado para resolver o problema de minimizar uma função $f(x)$ sujeita a $x \in \Omega$, com Ω fechado e convexo, e f com derivadas parciais contínuas em um conjunto aberto que contém Ω.

Defina $\mathcal{B}$ o conjunto das matrizes em $I\!R^{n\times n}$ positivas definidas tais que, para $B \in \mathcal{B}$, $\|B\| \leq L$ e $\|B^{-1}\| \leq L$, para algum L. Defina $g_k = \nabla f(x_k)$.

Algoritmo 2.5 (Iteração do método de Métrica Variável Inexata)

Sejam k a iteração atual e $x_0 \in \Omega$ um ponto inicial arbitrário. Dados $\eta \in (0,1]$, $\gamma \in (0,1)$, $0 < \sigma_1 < \sigma_2 < 1$, M um inteiro positivo, $x_k \in \Omega$, $B_k \in \mathcal{B}$.

Passo 1. *Cálculo da direção de busca*

Considere o subproblema

$$\begin{array}{ll} \text{Minimizar} & Q_k(d) \\ \text{sujeita a} & x_k + d \in \Omega, \end{array} \tag{2.10}$$

com

$$Q_k(d) = \frac{1}{2} d^T B_k d + g_k^T d.$$

Seja $\bar{d}_k$ o minimizador de (2.10). Seja d_k tal que $x_k + d_k \in \Omega$ e

$$Q_k(d_k) \leq \eta Q_k(\bar{d}_k). \tag{2.11}$$

Se $d_k = 0$, então pare declarando *"Ponto estacionário"*.

Passo 2. *Cálculo do tamanho do passo*

Faça $\alpha = 1$ e $f_{\max} = \max\{f(x_{k-j+1}) \mid 1 \leq j \leq \min\{k+1, M\}\}$.

Passo 3. *Teste de decréscimo da função*

Se

$$f(x_k + \alpha d_k) \leq f_{\max} + \gamma \alpha g_k^T d_k \tag{2.12}$$

então faça $\alpha_k = \alpha$, $x_{k+1} = x_k + \alpha_k d_k$ e pare. Senão, escolha $\alpha_{\text{novo}} \in [\sigma_1 \alpha, \sigma_2 \alpha]$, faça $\alpha \leftarrow \alpha_{\text{novo}}$ e repita do Passo 3.

Lema 2.5.1. (Lema 2.1 de [13]) *O método de Métrica Variável Inexata é bem-definido.*

Teorema 2.5.1. (Teorema 2.1 de [13]) *Suponha que o conjunto de nível $\{x \in \Omega \mid f(x) \leq f(x_0)\}$ é limitado. Então, ou o método de Métrica Variável Inexata pára em algum ponto estacionário x_k, ou todo ponto limite de seqüência gerada é estacionário.*

Vejamos agora a definição de uma iteração da versão monótona do método SPG [11, 12], que será usado na prova da convergência.

Algoritmo 2.6 (Iteração de SPG monótono)

Seja $\widehat{\Omega} \subset I\!R^n$ um conjunto convexo fechado. Suponha que $\alpha \in (0,1)$, $0 < \sigma_{\min} < \sigma_{\max} < \infty$. Seja $x_0 \in \widehat{\Omega}$ um ponto inicial arbitrário. Seja k a iteração atual. Dados $x_k \in \widehat{\Omega}$, $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$.

Passo 1. *Cálculo da direção de busca*

Calcule

$$p_k = P_{\widehat{\Omega}}\left(x_k - \frac{1}{\sigma_k}\nabla f(x_k)\right) - x_k.$$

Se $p_k = 0$, pare com x_k declarando *"Ponto estacionário"*.

Passo 2. *Cálculo do tamanho do passo*

Escolha $t_{\text{inicial}} \geq t_{\min}$ e faça $t \leftarrow t_{\text{inicial}}$.

Passo 3. *Teste de decréscimo da função*

Se

$$f(x_k + tp_k) \leq f(x_k) + \alpha t \nabla f(x_k)^T p_k.$$

então faça x_{k+1} tal que $f(x_{k+1}) \leq f(x_k + tp_k)$ e pare. Senão, escolha $t_{\text{novo}} \in [0.1t, 0.9t]$, faça $t \leftarrow t_{\text{novo}}$ e repita o Passo 3.

Teorema 2.5.2. *Seja $\widehat{\Omega}$ convexo e fechado. Suponha que a seqüência gerada pelo Algoritmo 2.6 seja limitada. Então*

1. *Se o Algoritmo 2.6 não pára em $x_k \in \widehat{\Omega}$, então x_{k+1} é bem-definido.*
2. *Se o Algoritmo 2.6 pára em x_k, então x_k é um ponto estacionário do problema*

$$\textit{Minimizar } f(x) \quad \textit{sujeita a} \quad x \in \widehat{\Omega}. \tag{2.13}$$

3. *Se x^* é um ponto limite da seqüência gerada pelo Algoritmo 2.6, então x^* é um ponto KKT de (2.13). Mais ainda,*

$$\lim_{k\to\infty} \|p_k\| = \lim_{k\to\infty} \|P_{\widehat{\Omega}}(x_k - \nabla f(x_k)) - x_k\| = 0.$$

Prova. Note que o método de SPG monótono se reduz ao Método de Métrica Variável Inexata apresentado anteriormente. Basta tomar $M = 1$ e $Q_k(d) = \frac{\|d\|^2\sigma_k}{2} + g_k^T d$. Assim, a prova segue diretamente do Lema 2.5.1 e do Teorema 2.5.1. ■

Agora podemos provar a convergência do Algoritmo 2.1. O algoritmo "irrestrito" a ser utilizado no Passo 5 do Algoritmo 2.1 (apresentado na Seção 2.1) admite várias implementações. Basicamente, é um algoritmo para minimização irrestrita que pára quando atinge a borda da região viável. Como visto na Seção 2.4, utilizamos os algoritmos "irrestritos" com regiões de confiança e com busca linear, como em BETRA e GENCAN, respectivamente. Para provar a convergência do Algoritmo 2.1, estes algoritmos devem satisfazer

Hipótese H1

- *Se x_{k+1} é calculado pelo algoritmo "irrestrito" e $x_k \in F_I$, então $x_{k+1} \in \bar{F}_I$ e $f(x_{k+1}) \leq f(x_k)$.*
- *Se $x_{k+j} \in F_I$ é calculado pelo algoritmo "irrestrito" para todo $j \geq 1$, existe j tal que $\|g_S(x_{k+j})\|_\infty \leq \varepsilon$.*

Teorema 2.5.3. *O Algoritmo 2.3 satisfaz Hipótese H1.*

Prova. Pela construção do algoritmo, se ele pára em x_k, vale a Hipótese H1. Basta verificar o caso em que o algoritmo gera uma seqüência infinita $\{x_k\}$. Neste caso, ou temos infinitas iterações de SPG ou temos infinitas iterações de regiões de confiança. Se são realizadas infinitas iterações de SPG, temos que o ponto limite da seqüência $\{x_k\}$ é um ponto estacionário. Se são realizadas infinitas iterações de regiões de confiança, por argumentos clássicos de minimização irrestrita, temos, também, que o ponto limite da seqüência $\{x_k\}$ é um ponto estacionário. Ou seja, em ambos os casos, para k suficientemente grande, $\|g_S(x_{k+j})\|_\infty \leq \varepsilon$. ■

Teorema 2.5.4. *O Algoritmo 2.4 satisfaz Hipótese H1.*

Prova. Pela construção do algoritmo, se ele pára em x_k, vale a Hipótese H1. Basta verificar o caso em que o algoritmo gera uma seqüência infinita $\{x_k\}$. Neste caso, ou temos infinitas iterações irrestritas de Newton truncado, ou o algoritmo atinge a borda e realiza infinitas iterações de backtracking para obter um ponto onde haja decréscimo suficiente da função. Em ambos os casos, argumentos clássicos de minimização irrestrita garantem que o ponto limite da seqüência $\{x_k\}$ é um ponto estacionário. Ou seja, em ambos os casos, para k suficientemente grande, $\|g_S(x_{k+j})\|_\infty \leq \varepsilon$. ■

Teorema 2.5.5. *Suponha que o Algoritmo 2.1 seja aplicado ao problema (2.8). Então:*

1. *Para todo $k = 0, 1, 2, \ldots$, se o Algoritmo 2.1 não termina em x_k, então x_{k+1} é bem-definido.*

2. *A seqüência $\{x_k\}$ gerada pelo Algoritmo 2.1 termina em um número finito de iterações em um ponto no qual $\|g_P(x_k, \delta^k)\|_\infty \leq \varepsilon$.*

Prova. A primeira parte da tese segue da Hipótese H1 e dos Teoremas 2.5.3 e 2.5.4. Vamos provar a segunda parte. Suponha, por contradição, que o Algoritmo 2.1 gera infinitas iterações. Consideramos dois casos:

1. há infinitas iterações de PSPG;

2. há um número finito de iterações de PSPG.

Considere o primeiro caso. Seja $K \subset \mathbb{N}$ o conjunto de índices k tal que x_{k+1} é computados pelo método PSPG. Como o número de faces F_I é finito, existe $I \subset \{1, \ldots, p\}$ e K_1 um subconjunto infinito de K tal que $x_k \in F_I$ para todo $k \in K_1$. Note que o número de subconjuntos diferentes $\Omega(x_k, \delta^k)$ também é finito e, portanto, existe $\widehat{\Omega}$ e K_2, um subconjunto infinito de K_1, tal que

$$\Omega(x_k, \delta^k) = \widehat{\Omega} \text{ para todo } k \in K_2.$$

Portanto, para todo $k \in K_2$,

$$p_k = P_{\widehat{\Omega}}(x_k - \lambda_k^{\text{PSPG}} \nabla f(x_k)) - x_k.$$

Suponha, sem perda de generalidade, que $\widehat{\Omega}$ seja definido pelas desigualdades

$$(a^i)^T x - b_i \leq 0 \text{ para todo } i = 1, \ldots, q.$$

Seja $i \in \{q+1, \ldots, p\}$. Então,

$$(a^i)^T x_k - b_i < -\delta_i^k.$$

Suponha que $\alpha \geq 0$ seja tal que

$$(a^i)^T (x_k + \alpha p_k) - b_i = 0.$$

Então,

$$\alpha = \frac{b_i - (a^i)^T x_k}{(a^i)^T p_k}.$$

Portanto,

$$\alpha > \frac{\delta_i^k}{\|a^i\| \|p_k\|}.$$

Como Ω é compacto, a seqüência $\{x_k\}$ é limitada e, então, a seqüência $\{p_k\}$ também é limitada. Ou seja, $\|p_k\| \leq c$ para todo k. Portanto,

$$\alpha > \frac{\delta_i^k}{\|a^i\| c} \geq \frac{\min\{\delta_i^k\}}{\max\{\|a^i\|\} c} \equiv \alpha_{\min}.$$

Isso implica que o primeiro ponto da forma $x_k + \alpha_{\max} p_k$ que é testado no método PSPG está longe de $\alpha_{\min} > 0$. Portanto, como $f(x_{k+1}) < f(x_k)$ para todo k, a seqüência $\{x_k\}_{k \in K_2}$ pode ser pensada, depois de renomeados seus índices, como sendo gerada pelo Algoritmo 2.6 aplicado à minimização de f em $\widehat{\Omega}$ (tomando $t_{\text{inicial}} = \alpha_{\max}$). Portanto, qualquer ponto limite x^* da seqüência é um ponto estacionário. Como $\Omega \subset \widehat{\Omega}$, isso implica que x^* é um ponto estacionário de (2.8). Então, $\lim_{k \to K_2} \|g_P(x_k, \delta^k)\| = 0$. Portanto, para $k \in K_2$ suficientemente grande, temos que $\|g_P(x_k, \delta^k)\|_\infty \leq \varepsilon$ e o Algoritmo 2.1 teria parado em x_k. Isso contradiz a hipótese de que $\{x_k\}$ é infinita.

No caso em que há um número finito de iterações de PSPG, temos que, para k suficientemente grande, todas as iterações são internas. Mais ainda, depois de um número finito de iterações, todos os iterandos pertencem à mesma face F_I. Pela Hipótese H1, isso implica que existe k tal que $\|g_S(x_k)\|_\infty \leq \varepsilon$. Portanto, nesta iteração devemos ter $\|g_P(x_k, \delta^k)\|_\infty \leq \varepsilon$, caso contrário o próximo iterando seria obtido por PSPG. ■

2.6 Detalhes de implementação

Implementamos, em Fortran 77, duas versões do Algoritmo 2.1. Chamamos a implementação que utiliza o Algoritmo 2.3 de BETRALIN e a implementação que utiliza o Algoritmo 2.4 de GENLIN. A seguir estão alguns detalhes de implementação destas duas versões.

2.6.1 Tratamento de inviabilidade

Em BETRALIN e GENLIN, mantemos a viabilidade das caixas exatamente e das restrições lineares com precisão $\varepsilon_{\text{viab}} = 10^{-8}$ (norma infinito das restrições lineares é, no máximo, 10^{-8}) a cada iteração.

Na ausência de erros de arredondamento, todos os iterandos do método são viáveis. No entanto, na computação com aritmética de ponto flutuante, é necessário considerar que pode haver perda de viabilidade. Definimos um parâmetro para o usuário chamado ACEITA-PERDA-PARCIAL-DE-VIABILIDADE. Se este parâmetro é verdadeiro, as perdas de viabilidades de até $\sqrt{\varepsilon_{\text{viab}}}$ ocorridas por erros numéricos são ignoradas. Mesmo que isto aconteça, a viabilidade pode ser recuperada durante a execução do método. Note que o ponto final calculado pelo método é viável (talvez com precisão $\sqrt{\varepsilon_{\text{viab}}}$), a menos que a projeção do ponto inicial falhe.

Para as projeções P_Ω e $P_{\Omega(x,\delta)}$, utilizamos a rotina QL. Esta é uma implementação em Fortran 77 do algoritmo para minimização de quadráticas sujeitas a restrições lineares de Goldfarb e Idnani [27], implementada por Powell [44] e modificada por K. Schittkowski [50]. Utilizamos $\varepsilon_{\text{QL}} = (\varepsilon_{\text{viab}})^{1.25}$ como precisão neste algoritmo e 5000 como número máximo de iterações. A rotina possui um parâmetro de saída INFO_{QL}, que pode ter os seguintes valores:

- $\text{INFO}_{\text{QL}} = 0$: Sucesso. A norma infinito das restrições normalizadas é menor ou igual a ε_{QL};

- INFO$_{\rm QL}$=1: Número máximo de iterações foi atingido;
- INFO$_{\rm QL}$=2: Precisão insuficiente para manter valores de função crescentes;
- INFO$_{\rm QL}$< 0: Problema inviável.

Note que não estamos, necessariamente, lidando com restrições normalizadas. Portanto, independentemente do critério de convergência satisfeito por QL, verificamos a viabilidade do ponto obtido. Nosso objetivo na projeção é obter um ponto viável com tolerância $\varepsilon_{\rm viab}$, ou seja, estamos interessados em um ponto x que satisfaça $\rho_{\rm viab}(x) \leq \varepsilon_{\rm viab}$, com

$$\rho_{\rm viab}(x) = \max\{0,\ |Ax - b|,\ Cx - d,\ \ell - x,\ x - u\}.$$

Para que isto aconteça, utilizamos o seguinte procedimento:

Passo 1. *Primeira tentativa*

Chame a rotina QL com $\varepsilon_{\rm QL} = (\varepsilon_{\rm viab})^{1.25}$. Seja $\bar{x}$ o ponto projetado e INFO$_{\rm QL}$ o status de saída da rotina. Seja $R = \{\bar{x}\}$.

Se $\rho_{\rm viab}(\bar{x}) \leq \varepsilon_{\rm viab}$ então devolva $\bar{x}$ como solução declarando *"Sucesso na projeção"*. Senão, se INFO$_{\rm QL}$=0 então vá para o Passo 2. Senão, vá para o Passo 3.

Passo 2. *Tentativa de satisfazer $\rho_{\rm viab}(x) \leq \varepsilon_{\rm viab}$ utilizando tolerância mais justa*

Passo 2.1. Defina $\varepsilon_{\rm QL} \leftarrow \varepsilon_{\rm QL}/100$.

Passo 2.2. Chame a rotina QL com $\varepsilon_{\rm QL}$. Seja $\bar{x}$ o ponto projetado e INFO$_{\rm QL}$ o status de saída da rotina. Seja $R = R \cup \{\bar{x}\}$.

Se $\rho_{\rm viab}(\bar{x}) \leq \varepsilon_{\rm viab}$ então devolva $\bar{x}$ como solução declarando *"Sucesso na projeção"*.

Passo 2.3. Se INFO$_{\rm QL}$=0 e $\varepsilon_{\rm QL} > \varepsilon_{\rm mach}$ então repita o Passo 2. Senão, vá para o Passo 4.

Passo 3. *Tentativa de tolerância menos restrita*

Chame a rotina QL com $\varepsilon_{\rm QL} = \sqrt{(\varepsilon_{\rm viab})^{1.25}}$. Seja $\bar{x}$ o ponto projetado e INFO$_{\rm QL}$ o status de saída da rotina. Seja $R = R \cup \{\bar{x}\}$.

Se $\rho_{\rm viab}(\bar{x}) \leq \varepsilon_{\rm viab}$ então devolva $\bar{x}$ como solução declarando *"Sucesso na projeção"*.

Passo 4. *Tratamento de perda de viabilidade.*

Seja $\hat{x} = \text{argmin}_{x \in R}\{\rho_{\rm viab}(x)\}$. Se o parâmetro do usuário ACEITA-PERDA-PARCIAL-DE-VIABILIDADE é verdadeiro e $\rho_{\rm viab}(\hat{x}) \leq \sqrt{\varepsilon_{\rm viab}}$ para todo $i = 1, \ldots, p$, então devolva $\hat{x}$ como solução declarando *"Perda parcial de viabilidade"*. Senão, pare declarando *"Falha na projeção"*.

No Passo 3 do Algoritmo 2.2, quando calculamos $\alpha_{\max} = \max\{\alpha \in [0,1] \mid [x_k, x_k + \alpha p_k] \subset \Omega\}$, não levamos em consideração as restrições de desigualdade usadas para definir $\Omega(x, \delta)$, ou seja, as restrições lineares de desigualdade j tais que $C^j x - d_j \geq -\delta_j$ não interferem no valor de $\alpha_{\max}$. Assim, se o parâmetro ACEITA-PERDA-PARCIAL-DE-VIABILIDADE for verdadeiro, o ponto x_{k+1} calculado por PSPG pode ter perda de viabilidade de até $\sqrt{\varepsilon_{\rm viab}}$.

Também pode haver perda de viabilidade quando um algoritmo "irrestrito" é usado para calcular x_{k+1}. Por erros numéricos, ao computar um ponto $x_k + p_k$ podemos obter um ponto inviável (com precisão $\varepsilon_{\text{viab}}$), mesmo que x_k seja viável. Neste caso, se o parâmetro ACEITA-PERDA-PARCIAL-DE-VIABILIDADE for verdadeiro, inviabilidade de até $\sqrt{\varepsilon_{\text{viab}}}$ é ignorada. Caso contrário, o algoritmo "irrestrito" pára e uma iteração de PSPG (Algoritmo 2.2) é feita para calcular x_{k+1}.

Note que, se o parâmetro ACEITA-PERDA-PARCIAL-DE-VIABILIDADE for verdadeiro, o ponto final x^* pode ser viável apenas com tolerância $\sqrt{\varepsilon_{\text{viab}}}$, ou seja, $\rho_{\text{viab}}(x^*) \leq \sqrt{\varepsilon_{\text{viab}}}$. Neste caso, é recomendável que o usuário recomece o método utilizando x^* como ponto inicial e definindo o parâmetro ACEITA-PERDA-PARCIAL-DE-VIABILIDADE como falso.

2.6.2 Cálculo da fatoração LQ

Para definir a matriz $\bar{A}$, usada no Algoritmo 2.1, montamos a matriz $\tilde{A} \in I\!R^{\bar{t}\times s}$, composta pelas colunas de A e C correspondentes às variáveis livres, removendo as linhas de C correspondentes às restrições lineares que não são ativas. Calculamos, então, a fatoração QR com permutação de colunas de $\tilde{A}^T$. Para isso, usamos a implementação da fatoração QR da coleção LAPACK chamada DGEQP3.

Com esta fatoração, obtemos

$$\tilde{A}^T P = Q \begin{bmatrix} R_1 & R_2 \\ 0 & 0 \end{bmatrix},$$

onde $P \in I\!R^{\bar{t}\times\bar{t}}$ é uma matriz de permutação, $Q \in I\!R^{s\times s}$ é matriz ortogonal, $R_1 \in I\!R^{t\times t}$ é uma matriz triangular superior não-singular e $R_2 \in I\!R^{t\times(\bar{t}-t)}$ é qualquer. Temos que as primeiras t colunas de $\tilde{A}^T P$ (ou as primeiras t linhas de $P^T\tilde{A}$) são linearmente independentes, o que define $\bar{A}^T$.

Note que a matriz Q calculada nesta fatoração é tal que

$$\bar{A} = \begin{bmatrix} R_1^T & 0 \end{bmatrix} Q^T.$$

Ou seja, a matriz Z é formada pelas ultimas $s - t$ colunas de Q (veja 2.4.3.1).

A matriz Q, obtida pela fatoração QR da matriz $\tilde{A}^T$, é representada pelo produto de refletores elementares

$$Q = W_1 W_2 \ldots W_t,$$

cada W_i é tal que

$$W_i = I - \tau_i v_i v_i^T,$$

onde τ_i é um escalar e v_i é um vetor com cujas componentes de 1 a $i-1$ são nulas e a componente i é igual a 1. W é chamada matriz de Householder.

Como Z é formada pelas últimas $s-t$ colunas de Q, para calcular o produto Zv, $v \in I\!R^{s-t}$, basta multiplicar as últimas $s - t$ colunas de W_k pelo vetor v, obtendo $v_k \in I\!R^s$. A partir

daí, basta calcular $v_i = W_{i+1}v_{i+1}$, para $i = k-1$ até $i = 1$. Note que, seguindo estes passos, temos que $v_1 = Zv$.

De maneira análoga, para calcular o produto Z^Tv, $v \in I\!R^s$, basta calcular $v_i = W_i^T v_i$, para $i = 1$ até $i = k-1$. Então, multiplicam-se as $s-t$ últimas linhas de W_k^T por v_{k-1}, obtendo v_k, que é exatamente o produto Z^Tv. Estas técnicas para o cálculo de produtos matriz-vetor são interessantes, pois tiram proveito da estrutura da matriz Q.

Note que, quando o passo máximo é tomado no algoritmo "irrestrito" na iteração k do Algoritmo 2.1, algum limitante é atingido ou alguma restrição linear é satisfeita por igualdade. Denotemos por $\bar{A}_k^T$ a matriz definida na iteração k, Q_k a matriz ortogonal calculada na iteração k e R_{1k} a matriz triangular superior não-singular calculada na iteração k. Quando um limitante é atingido, $\bar{A}_{k+1}^T$ é obtida pela remoção da linha de $\bar{A}_k^T$ correspondente à variável que passou a ser fixa. Quando uma restrição linear é satisfeita por igualdade (passa a ser ativa), $\bar{A}_{k+1}^T$ é obtida pela inserção da linha de A correspondente a esta restrição (a menos das componentes correspondentes a variáveis fixas) como coluna de $\bar{A}_k^T$.

Como, neste caso, temos a fatoração QR de $\bar{A}_k^T$ e a matriz $\bar{A}_{k+1}^T$ é obtida através da remoção de uma linha ou inserção de uma coluna de $\bar{A}_k^T$, o cálculo da fatoração QR de $\bar{A}_{k+1}^T$ pode ser feito apenas atualizando os fatores Q_k e R_{1k}. Para atualizar Q_k e R_{1k} após a inserção de uma coluna v em $\bar{A}_k^T$, calculamos $\bar{v} = Q_k^T v$. Se as componentes de $t+1$ até s de $\bar{v}$ forem nulas, significa que o vetor v é linearmente dependente das colunas de $\bar{A}_k^T$. Neste caso, os fatores Q_k e R_{1k} permanecem os mesmos. Caso contrário, calculamos uma matriz $W_{t+1} = I - \tau_{t+1}v_{t+1}v_{t+1}^T$ (veja definição de Q acima) tal que $W_{t+1}\bar{v}$ possua as componentes de $t+2$ até s nulas. Assim, temos $Q_{k+1} = Q_k W_{t+1}$ e $R_{1k+1} = [R_{1k} \quad W_{t+1}\bar{v}]$.

Para atualizar Q_k e R_{1k} após a remoção de uma linha i de $\bar{A}_k^T$, teríamos de usar a matriz Q_k de forma explícita (veja [28], por exemplo). Então, para aproveitar a estrutura da matriz ortogonal, optamos por atualizar a fatoração tratando a variável que se tornou fixa na iteração k como uma restrição linear do tipo $e_i x_k \geq l_i$ (ou $e_i x_k \leq u_i$) que se tornou ativa, onde e_i é a i-ésima linha da matriz identidade. Desta forma, podemos atualizar a fatoração utilizando o mesmo procedimento exposto acima.

2.6.3 Outros detalhes

Na implementação do Algoritmo 2.3 fizemos todas as mudanças relatadas na Seção 1.2. Além disso, sempre que um ponto x_{k+1} está em uma face diferente de x_k, o valor inicial de λ no Algoritmo 2.5 passa a ser 0. Caso a face seja a mesma, o valor de λ é reaproveitado da iteração anterior.

Quando um ponto (não estacionário) é obtido por algum dos algoritmos "irrestritos", poderíamos aplicar a técnica de extrapolação apresentada em [8]. No entanto, como ela necessita de várias projeções (veja Algoritmo 1.7), que são custosas na presença de restrições lineares, optamos por não utilizá-la.

Se o raio da região de confiança do Algoritmo 2.3 se tornou muito pequeno ou a busca linear chegou a um tamanho de passo muito pequeno sem obter decréscimo suficiente da

função objetivo, uma iteração de PSPG (Algoritmo 2.2) é feita para calcular x_{k+1}. Se a busca linear em PSPG falhar, o algoritmo principal também pára.

Observe que, no Algoritmo 2.1, sempre que calculamos gradiente projetado $g_P(x, \delta)$ (Passo 3), ou terminamos a execução do método ou utilizamos uma iteração do PSPG (Algoritmo 2.2). Uma possibilidade para diminuir o número de projeções é primeiro calcularmos a direção de gradiente espectral projetado parcial p^{PSPG} e, a partir dela, tentar inferir algo sobre a norma de $g_P(x, \delta)$. Se isso for possível, usaremos apenas uma projeção, em vez de duas. Lembre-se que

$$g_P(x, \delta) = P_{\Omega(x,\delta)}(x - \nabla f(x)) - x,$$

e

$$p^{\text{PSPG}} = P_{\Omega(x,\delta)}(x - \lambda^{\text{PSPG}} \nabla f(x)) - x.$$

Podemos afirmar que, para $\lambda^{\text{SPG}} > 0$, vale que

- se $\lambda^{\text{SPG}} > 1$,
$$\|p^{\text{PSPG}}\| < \varepsilon \Rightarrow \|g_P(x, \delta)\| < \varepsilon,$$
$$\|p^{\text{PSPG}}\| > \lambda^{\text{PSPG}} \varepsilon \Rightarrow \|g_P(x, \delta)\| > \varepsilon;$$
- se $0 < \lambda^{\text{PSPG}} < 1$,
$$\|p^{\text{PSPG}}\| < \lambda^{\text{PSPG}} \varepsilon \Rightarrow \|g_P(x, \delta)\| < \varepsilon,$$
$$\|p^{\text{PSPG}}\| > \varepsilon \Rightarrow \|g_P(x, \delta)\| > \varepsilon.$$

Se, pela norma de p^{PSPG}, sabemos que não estamos na solução, realizamos uma iteração de PSPG (Algoritmo 2.2) com a direção já calculada, economizando aqui uma projeção. Se, pela norma de p^{PSPG}, sabemos que estamos na solução, paramos o algoritmo. Neste caso teremos calculado a direção p^{PSPG} em vão. Se a norma de p^{PSPG} não fornece informação sobre a norma de $g_P(x, \delta)$, calculamos este último. Note que, quando decidimos que precisamos, de fato, calcular $g_P(x, \delta)$, estamos gastando o mesmo número de projeções do que em uma versão que não faz uso destas propriedades, a menos do caso em que chegamos à solução. Nessa situação, teremos realizado uma projeção a mais, já que a direção p^{PSPG} não será utilizada. Mas, como isto acontece apenas uma vez na execução do algoritmo, no pior caso, temos uma projeção a mais. Ou seja, essa técnica pode reduzir muito o número de projeções feitas ao longo do algoritmo.

Incorporamos esta idéia para economizar projeções em Betralin e Genlin. Os valores de δ e $\delta_{\min}$ são discutidos na Seção 2.7.

2.7 Resultados numéricos

Para definir alguns parâmetros de Betralin e Genlin, utilizaremos os problemas da coleção CUTEr que possuem um número limitado de variáveis e apenas restrições lineares. Estes problemas também serão usados para verificar o desempenho dos novos métodos e compará-los com métodos conhecidos desenvolvidos para resolver problemas com restrições lineares. Todos os resultados do uso de Betralin e Genlin aplicados a problemas da coleção

CUTEr são apresentados na Seção 2.7.1. O critério de convergência usados nestes experimentos é que a norma infinito do gradiente projetado no ponto deve ser menor ou igual a $\varepsilon_{\text{otim}} = 10^{-8}$ ($\|g_P(x_k, \delta^k)\|_\infty \leq 10^{-8}$). Nos algoritmos "irrestritos", usamos $\varepsilon = 10^{-8}$ (ou seja, $\|Z^T \nabla f(x_k)\|_\infty \leq 10^{-8}$). Para o algoritmo "irrestrito" de regiões de confiança, temos ainda o critério e $|\varphi(s_k)| \leq 10^{-5}$ e os parâmetros $\Delta_{\min} = 10^{-8}$ e $\sigma = 0.1$. Para o algoritmo "irrestrito" com busca linear utilizamos $\gamma = 10^{-4}$ e $\theta = 10^{-6}$. Também foi determinado um número máximo de iterações (200000) e um número máximo de avaliações de função (1000000). Utilizamos ACEITA-PERDA-PARCIAL-DE-VIABILIDADE verdadeiro.

Verificaremos também, na Seção 2.7.2, como resolver o problema de estimação de constantes ópticas e da espessura de filmes finos [5, 6, 16, 39] usando BETRALIN e GENLIN e os resultados por eles obtidos.

2.7.1 Comparações utilizando a coleção Cuter

Denotamos por n_{lim} o número de limitantes nas variáveis do problema. Consideramos que variáveis com limitantes inferior e superior contribuem com dois limitantes. Denotamos por r o número de restrições lineares de igualdade, m o número de restrições lineares de desigualdade e n o número de variáveis verdadeiras.

Selecionamos para nossos experimentos todos os problemas da coleção CUTEr [15] com, no máximo, 500 variáveis e com $1 \leq r + m \leq 2000$, totalizando 133 problemas. As principais características destes problemas estão na Tabela 2.1. Alguns problemas da coleção CUTEr possuem variáveis nas quais os limitantes inferior e superior coincidem. Na Tabela 2.1, $n = n_1(n_2)$ significa que o problema possui n_1 variáveis verdadeiras e n_2 variáveis fixas. Neste caso, n_{lim} é o número de limitantes das variáveis verdadeiras.

Os problemas NASH, MODEL, ARGLALE, ARGLBLE, ARGLCLE e LINCONT são inviáveis e, portanto, tanto BETRALIN como GENLIN param no Passo 1 declarando *"Problema inviável"*. Estes 6 problemas foram eliminados dos experimentos e, daqui em diante, serão considerados os 127 problemas restantes.

2.7.1.1 Escolha de δ

Testamos duas escolhas diferentes para o parâmetro δ, usado em PSPG e no cálculo do gradiente projetado:

- Escolha 1: $\delta_j = \bar{\delta} \max(1, |d_j|), 1 \leq j \leq m$.

  $\bar{\delta}$ começa com valor $\frac{1}{10}$ e, sempre que uma iteração de PSPG é feita, se $\alpha_{\max} < 0.1$ então $\bar{\delta} \leftarrow 100\ \bar{\delta}$.

- Escolha 2: $\delta \equiv \infty$.

Note que, usando a escolha 2, uma iteração de PSPG se reduz a uma iteração de SPG e o gradiente projetado se reduz ao gradiente projetado contínuo.

Problema	n	n_{lim}	r	m	Problema	n	n_{lim}	r	m
EXTRASIM	2	1	1	0	AVGASB	8	16	0	10
HS9	2	0	1	0	DUALC5	8	16	1	277
TAME	2	2	1	0	DUALC8	8	16	1	502
HS21	2	4	0	1	DUALC1	9	18	1	214
HS35MOD	2(1)	2	0	1	HS112	10	10	3	0
HUBFIT	2	1	0	1	ODFITS	10	10	6	0
LSQFIT	2	1	0	1	GENHS28	10	0	8	0
BOOTH	2	0	2	0	PORTFL1	12	24	1	0
HIMMELBA	2	0	2	0	PORTFL2	12	24	1	0
SUPERSIM	2	1	2	0	PORTFL3	12	24	1	0
SIMPLLPA	2	2	0	2	PORTFL4	12	24	1	0
ZECEVIC2	2	4	0	2	PORTFL6	12	24	1	0
HS24	2	2	0	3	LOTSCHD	12	12	7	0
SIMPLLPB	2	2	0	3	HS118	15	30	0	29
PT	2	0	0	501	HS119	16	32	8	0
SIPOW1M	2	0	0	2000	NASH	18(54)	6	24	0
SIPOW1	2	0	0	2000	FCCU	19	19	8	0
SIPOW2M	2	0	0	2000	RES	20	40	12	2
SIPOW2	2	0	0	2000	DEGENLPA	20	40	15	0
HS28	3	0	1	0	DEGENLPB	20	40	15	0
HS62	3	6	1	0	KSIP	20	0	0	1001
HS35I	3	6	0	1	MAKELA4	21	0	0	40
HS35	3	3	0	1	WATER	31	62	10	0
HS36	3	6	0	1	LOADBAL	31	42	11	20
HS37	3	6	0	2	MODEL	42(1500)	84	23	15
STANCMIN	3	3	0	2	HIMMELBJ	43(2)	43	14	0
ZANGWIL3	3	0	3	0	DALLASS	46	92	31	0
TFI2	3	0	0	101	AVION2	49	98	15	0
TFI3	3	0	0	101	GOFFIN	51	0	0	50
OET1	3	0	0	1002	DUAL4	75	150	1	0
HONG	4	8	1	0	LINSPANH	81(16)	162	33	0
HS41	4	8	1	0	SPANHYD	81(16)	162	33	0
LIN	4	8	2	0	QPCBLEND	83	83	43	31
HS76I	4	8	0	3	QPNBLEND	83	83	43	31
HS76	4	4	0	3	DUAL1	85	170	1	0
S277-280	4	4	0	4	DUAL2	96	192	1	0
HS44NEW	4	4	0	6	HIMMELBI	100	200	0	12
HS44	4	4	0	6	DUAL3	111	222	1	0
BIGGSC4	4	8	0	13	SMBANK	117	234	64	0
HATFLDH	4	8	0	13	QPCBOEI2	143	197	4	181
OET3	4	0	0	1002	QPNBOEI2	143	197	4	181
SIPOW3	4	0	0	2000	AGG	163	163	36	452
SIPOW4	4	0	0	2000	HYDROELS	167(2)	334	0	336
HS48	5	0	2	0	GMNCASE1	175	0	0	300
HS49	5	0	2	0	GMNCASE4	175	0	0	350
BT3	5	0	3	0	GMNCASE2	175	0	0	1050
HS50	5	0	3	0	GMNCASE3	175	0	0	1050
HS51	5	0	3	0	SSEBLIN	192(2)	360	48	24
HS52	5	0	3	0	DALLASM	196	392	151	0
HS53	5	10	3	0	ARGLCLE	200	0	399	0
LSNNODOC	5	6	4	0	ARGLALE	200	0	400	0
HS268	5	0	0	5	ARGLBLE	200	0	400	0
S268	5	0	0	5	PRIMALC1	230	215	0	9
HS86	5	5	0	10	PRIMALC2	231	229	0	7
EXPFITA	5	0	0	22	LINCONT	249(1008)	0	419	0
EXPFITB	5	0	0	102	PRIMALC5	287	278	0	8
EXPFITC	5	0	0	502	PRIMAL1	325	1	0	85
HS54	6	12	1	0	QPCBOEI1	384	540	9	431
HS55	6	8	6	0	QPNBOEI1	384	540	9	431
PENTAGON	6	0	0	15	QPCSTAIR	385(82)	385	209	147
HS21MOD	7	8	0	1	QPNSTAIR	385(82)	385	209	147
EQC	7(2)	14	0	3	STEENBRA	432	432	108	0
QCNEW	7(2)	14	0	3	STATIC3	434	144	96	0
QC	7(2)	14	0	4	STEENBRB	468	468	108	0
DUALC2	7	14	1	228	STEENBRD	468	468	108	0
HS105	8	16	0	1	STEENBRF	468	468	108	0
AVGASA	8	16	0	10					

Tabela 2.1: Problemas com apenas restrições lineares selecionados da coleção CUTEr.

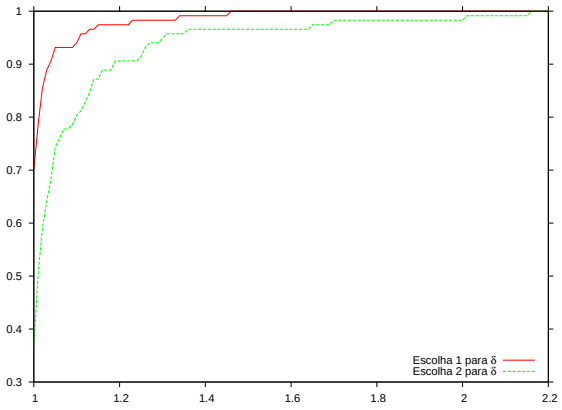


Figura 2.1: Curva de perfil de desempenho comparando escolhas 1 e 2 para o parâmetro δ usado nas projeções.

Como o comportamento das versões de BETRALIN e GENLIN se mostraram muito similares, apresentaremos aqui apenas os resultados obtidos pelas versões de GENLIN que utilizam a escolha 1 ou a escolha 2 para o cálculo de δ.

Há 10 problemas (BOOTH, HIMMELBA, SUPERSIM, TAME, STANCMIN, ZANGWIL3, HS55, RES, LINSPANH e GMNCASE4) para os quais a solução é dada por x_1, a projeção do ponto inicial x_0 no conjunto viável. Como as versões que utilizam a escolha 1 e a escolha 2 para δ coincidem nestes casos, estes problemas foram eliminados dos experimentos desta seção. As duas versões encontraram valores de função equivalentes no iterando final em todos os 117 problemas (usando o critério (1.13)).

Para comparar as duas versões de GENLIN usamos o tempo de CPU como medida de desempenho. Note que o custo das projeções pode depender do valor de δ. Assim, as duas versões podem ter custo computacional diferente a cada iteração. A Figura 2.1 mostra o gráfico de perfil de desempenho (veja Apêndice A). A figura mostra que, como mencionado anteriormente, não há diferença na robustez de ambas as versões. Por outro lado, usando a escolha 1 temos uma versão muito mais eficiente. A escolha 1 faz com que o método seja mais rápido em 70.08% dos problemas, enquanto a versão que utiliza a escolha 2 é mais rápida em 36.75% dos problemas. Finalmente, a Figura 2.1 mostra que a versão que usa a escolha 2 raramente utiliza mais do que o dobro do tempo de CPU gasto pela versão com a escolha 1.

Testamos também uma outra forma de escolher o valor de δ, que consiste em considerar um valor fixo e finito de δ. Utilizando esta forma, há um problema (chamado HYDROELS) para o qual a escolha 1 de δ sem a atualização de $\bar{\delta}$ consumiu muito mais tempo do que a escolha 2. A explicação para isto é a seguinte: há 94 restrições lineares ativas e 73 limitantes ativos na solução. Usando $\delta \equiv \infty$, várias restrições são adicionadas ao conjunto de restrições ativas de uma só vez (em uma mesma iteração). Usando a escolha "estática" da versão que usa a escolha 1 para δ (e calculando o passo máximo para manter viabilidade com relação às restrições lineares não incluídas na projeção), restrições são adicionadas ao conjunto de restrições ativas lentamente. Este é o risco de utilizar projeções parciais e a motivação para a definição dinâmica de δ.

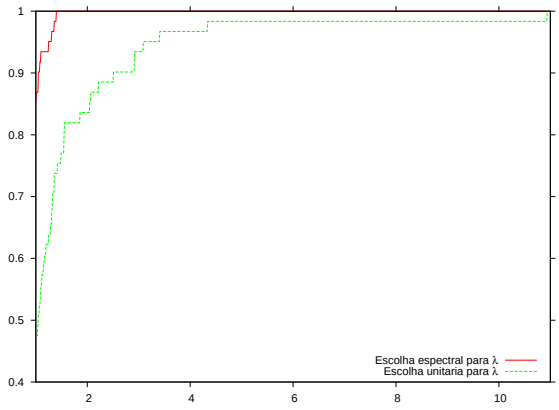


Figura 2.2: Curva de perfil de desempenho comparando a versão com passo espectral com salvaguarda λ_{espec} e a versão com passo unitário $\lambda_k \equiv 1$.

2.7.1.2 Escolha de λ_k^{PSPG}

Pelo experimento da seção anterior, fixamos δ como a escolha 1. Desejamos verificar agora que, no Algoritmo 2.2, usar λ_k^{PSPG} como o passo espectral λ_{espec} com salvaguardas, onde

$$\lambda_{\text{espec}} = \begin{cases} \frac{\|x_k - x_{k-1}\|^2}{(x_k - x_{k-1})^T(\nabla f(x_k) - \nabla f(x_{k-1}))}, & \text{se } (x_k - x_{k-1})^T(\nabla f(x_k) - \nabla f(x_{k-1})) > 0, \\ 1, & \text{caso contrário,} \end{cases}$$

é melhor que usar $\lambda_k \equiv 1$.

Como ambas as opções coincidem para problemas com função objetivo linear ou constante, ignoraremos os 26 problemas deste tipo da Tabela 2.1. Assim, temos 101 problemas. Ambas as escolhas também coincidem se nenhuma iteração de PSPG é feita, o que acontece em outros 40 problemas (incluindo aqueles que têm x_1 como solução). Restam, então, 61 problemas que serão usados nesta seção.

Como o custo de calcular λ_k é desprezível quando comparado a toda a álgebra linear envolvida em todo o método, usamos o número de avaliações de função como medida de desempenho. A Figura 2.2 mostra a curva de perfil de desempenho comparando ambas as versões de Genlin.

Ambas as alternativas encontraram valor de função objetivo equivalentes para todos os problemas. A Figura 2.2 mostra que usar o passo espectral, de fato, torna o método mais eficiente.

2.7.1.3 Comparação de Betralin e Genlin com Algencan-B e Algencan-G

Como visto no Capítulo 1, Algencan-B e Algencan-G são métodos de Lagrangiano aumentado que utilizam, respectivamente, Betra e Gencan para resolver os subproblemas. Esses métodos trabalham explicitamente com restrições de caixa e penalizam as demais restrições existentes no problema. Estamos interessados em comparar o desempenho de métodos

que lidam explicitamente com restrições lineares (BETRALIN e GENLIN) com métodos que as penalizam (ALGENCAN-B e ALGENCAN-G).

Primeiramente, vamos comparar BETRALIN e ALGENCAN-B. Em todos os 127 problemas, o iterando final de BETRALIN satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. O problema HIMMELBJ foi o único que, utilizando o parâmetro ACEITA-PERDA-PARCIAL-DE-VIABILIDADE verdadeiro, obteve viabilidade $\sqrt{\varepsilon_{\text{viab}}}$. Por isso, foi feito o reinício do método utilizando o ponto final calculado nesta chamada como ponto inicial e o parâmetro ACEITA-PERDA-PARCIAL-DE-VIABILIDADE como falso (como sugerido na Seção 2.6).

Em 125 dos 127 problemas, BETRALIN pára satisfazendo seu critério de parada relativo a sucesso. No problema STATIC3, BETRALIN pára porque o problema parece ser ilimitado. No problema HIMMELBJ, BETRALIN pára em um ponto com valor de função bem-definido por tentar calcular valor de função em vários pontos em que este não é bem-definido.

O iterando final de ALGENCAN-B não satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$ em 8 dos 127 problemas. Nestes problemas, a explicação para o que acontece é a seguinte:

- no problema QPCBOEI1, ALGENCAN-B pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas DEGENLPA, AVION2, QPNBLEND, QPCBOEI2, AGG, QPNBOEI1 e STATIC3, ALGENCAN-B pára declarando que o parâmetro de penalização se tornou muito grande.

Nos outros 119 problemas, o iterando final de ALGENCAN-B é viável com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 114 destes 119 problemas, ALGENCAN-B pára satisfazendo seu critério de parada relativo a sucesso. Nos 5 problemas restantes, a explicação é a seguinte:

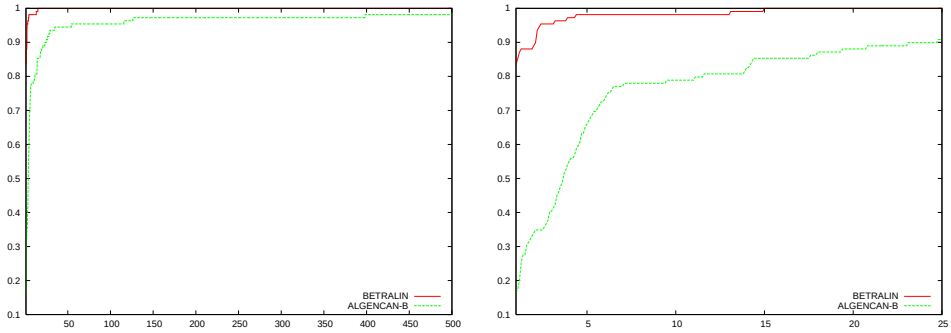
- no problema QPNSTAIR, ALGENCAN-B pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas DEGENLPB, HIMMELBJ, QPCBLEND e QPNBOEI2, ALGENCAN-B pára por atingir o número máximo de iterações permitido.

Considerando os 119 problemas nos quais BETRALIN e ALGENCAN-B obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final de ALGENCAN-B é equivalente ao obtido por BETRALIN. No entanto,

- em 5 problemas (GOFFIN, QPNBOEI2, STEENBRB, STEENBRD, STEENBRF), BETRALIN obteve valor de função menor do que ALGENCAN-B;
- em 5 problemas (BIGGSC4, HATFLDH, LIN, HS54, GMNCASE1), ALGENCAN-B obteve valor de função menor do que BETRALIN;

Nos 109 problemas restantes, tanto BETRALIN como ALGENCAN-B obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 109 problemas:

- BETRALIN realizou menos avaliações de função em 104 casos;
- ALGENCAN-B realizou menos avaliações de função em 3 casos;



(a) (b)

Figura 2.3: Curva de perfil de desempenho comparando Betralin e Algencan-B usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

- Betralin e Algencan-B realizaram o mesmo número de avaliações de função em 2 casos;
- Betralin foi mais rápido que Algencan-B (com tolerância de 10%) em 90 casos;
- Algencan-B foi mais rápido que Betralin (com tolerância de 10%) em 16 casos;
- Betralin e Algencan-B gastaram a mesma quantidade de tempo em 3 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (24 problemas), Betralin foi mais rápido que Algencan-B 14 vezes e Algencan-B foi mais rápido que Betralin 8 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 2 casos).

Na Tabela 2.2 temos um resumo da comparação entre Betralin e Algencan-B.

	Betralin	Algencan-B
Problemas viáveis	127/127	119/127
Satisfez critérios de convergência	125/127	114/119
Valor de função melhor	5/119	5/119
Menos avaliações de função	104/109	3/109
Tempo menor (10% tolerância)	90/109	16/109

Tabela 2.2: Comparação entre Betralin e Algencan-B para problemas selecionados da coleção CUTEr

Considerando os 109 problemas para os quais tanto Betralin como Algencan-B chegaram a um ponto viável com mesmo valor de função, construímos um gráfico de perfil de desempenho usando o tempo de CPU como medida de desempenho (veja Figura 2.3). Observe que Betralin mostrou desempenho muito superior a Algencan-B, como esperávamos.

Vamos agora comparar Genlin e Algencan-G. O iterando final de Genlin em todos os 127 problemas satisfazem viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 125 destes 127

problemas, GENLIN pára satisfazendo seu critério de parada relativo a sucesso. No problema STATIC3, GENLIN pára porque o problema parece ser ilimitado. No problema HIMMELBJ, GENLIN pára em um ponto com valor de função bem-definido por tentar calcular valor de função em vários pontos em que esta não é bem-definida.

O iterando final de ALGENCAN-G não satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$ em três dos 127 problemas. Nestes problemas, a explicação para o que acontece é a seguinte:

- nos problemas QPCBOEI1 e QPNBOEI1, ALGENCAN-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- no problema QPCBOEI2, ALGENCAN-G pára por atingir o número máximo de iterações permitido.

Nos outros 124 problemas, o iterando final de ALGENCAN-G é viável com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 112 destes 124 problemas, ALGENCAN-G pára satisfazendo seu critério de parada relativo a sucesso. Nos 12 problemas restantes, a explicação é a seguinte:

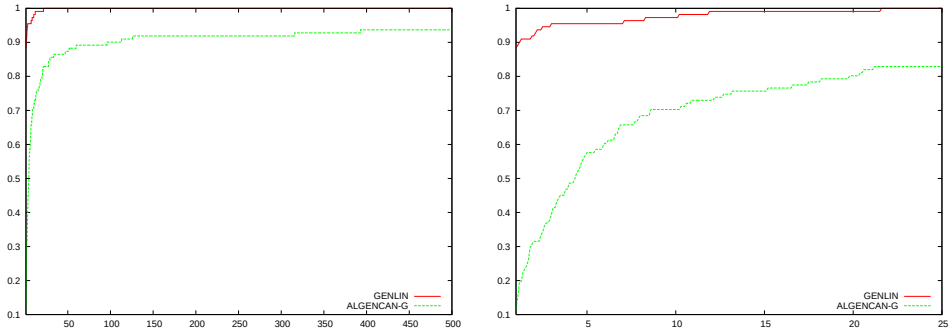
- no problema QPNSTAIR, ALGENCAN-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas HS268, S268, DEGENLPA, DEGENLPB, HIMMELBJ, AVION2, QPCBLEND, QPNBLEND, QPNBOEI2, AGG e STATIC3, ALGENCAN-G pára por atingir o número máximo de iterações permitido.

Considerando os 124 problemas nos quais GENLIN e ALGENCAN-G obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final de ALGENCAN-G é equivalente ao obtido por GENLIN. No entanto,

- em 5 problemas (PENTAGON, HIMMELBJ, STEENBRB, STEENBRD, STEENBRF), GENLIN obteve valor de função menor do que ALGENCAN-G;
- em 7 problemas (BIGGSC4, HATFLDH, LIN, HS54, HS105, QPNBOEI2, GMNCASE1), ALGENCAN-G obteve valor de função menor do que GENLIN.

Nos 111 problemas restantes, tanto GENLIN como ALGENCAN-G obtiveram pontos viáveis com valores de função equivalentes (excluindo aqui o problema STATIC3, para o qual tanto GENLIN como ALGENCAN-G param porque o problema parece ilimitado). Considerando estes 111 problemas:

- GENLIN realizou menos avaliações de função em 110 casos;
- em nenhum caso ALGENCAN-G realizou menos avaliações;
- GENLIN e ALGENCAN-G realizaram o mesmo número de avaliações de função em um caso;
- GENLIN foi mais rápido que ALGENCAN-G (com tolerância de 10%) em 94 casos;
- ALGENCAN-G foi mais rápido que GENLIN (com tolerância de 10%) em 12 casos;



(a) (b)

Figura 2.4: Curva de perfil de desempenho comparando Genlin e Algencan-G usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

- Genlin e Algencan-G gastaram a mesma quantidade de tempo em 5 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (25 problemas), Genlin foi mais rápido que Algencan-G 16 vezes e Algencan-G foi mais rápido que Genlin 5 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 4 casos).

Na Tabela 2.3 temos um resumo da comparação entre Genlin e Algencan-G.

	Genlin	Algencan-G
Problemas viáveis	127/127	124/127
Satisfez critérios de convergência	125/127	112/124
Valor de função melhor	5/124	7/124
Menos avaliações de função	110/111	0/111
Tempo menor (10% tolerância)	94/111	12/111

Tabela 2.3: Comparação entre Genlin e Algencan-G para problemas selecionados da coleção Cuter

Considerando os 111 problemas para os quais tanto Genlin como Algencan-G chegaram a um ponto viável com mesmo valor de função, construímos um gráfico de perfil de desempenho usando o tempo de CPU como medida de desempenho (veja Figura 2.4). Como esperado, Genlin apresentou índices de eficiência e robustez superiores a Algencan-G. Ou seja, trabalhar explicitamente com as restrições lineares é melhor do que penalizá-las em um método do tipo Lagrangiano aumentado.

2.7.1.4 Comparação entre Betralin e Genlin

Como visto na Seção 2.7.1.3, tanto Betralin como Genlin obtiveram pontos finais viáveis nos 127 problemas viáveis da Tabela 2.1. Usaremos, então, estes 127 problemas para comparar o desempenho de ambos os métodos.

Ambos os métodos satisfizeram o critério de parada relacionado a sucesso em 125 problemas. No problema STATIC3, ambos param porque o problema parece ser ilimitado. No problema HIMMELBJ, BETRALIN e GENLIN param em um ponto com valor de função bem-definido por tentar calcular valor de função em vários pontos em que esta não é bem-definida.

BETRALIN e GENLIN encontram valores de função equivalentes em 125 problemas. Nos dois problemas restantes (HS54 e PENTAGON), GENLIN encontra valor de função menor. Eliminamos o problema STATIC3, que ambos os métodos consideram ilimitado, e utilizamos os 124 problemas restantes na comparação entre BETRALIN e GENLIN. Considerando estes 124 problemas:

- BETRALIN realizou menos avaliações de função em 25 casos;
- GENLIN realizou menos avaliações de função em 33 casos;
- BETRALIN e GENLIN realizaram o mesmo número de avaliações de função em 66 casos;
- BETRALIN foi mais rápido que GENLIN (com tolerância de 10%) em 13 casos;
- GENLIN foi mais rápido que BETRALIN (com tolerância de 10%) em 32 casos;
- BETRALIN e GENLIN gastaram a mesma quantidade de tempo em 79 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (29 problemas), BETRALIN foi mais rápido que GENLIN 3 vezes e GENLIN foi mais rápido que BETRALIN 11 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 15 casos).

Na Tabela 2.4 temos um resumo da comparação entre BETRALIN e GENLIN.

	BETRALIN	GENLIN
Problemas viáveis	127/127	127/127
Satisfez critérios de convergência	125/127	125/127
Valor de função melhor	0/127	2/127
Menos avaliações de função	25/124	33/124
Tempo menor (10% tolerância)	13/124	32/124

Tabela 2.4: Comparação entre BETRALIN e GENLIN para problemas selecionados da coleção CUTEr

Considerando os 124 problemas para os quais tanto BETRALIN como GENLIN chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 2.5 usamos o tempo de CPU como medida de desempenho. Na Figura 2.6 usamos o número de avaliações de função como medida de desempenho.

Podemos ver que há muitos casos de empate, tanto no gasto de tempo como no número de avaliações de função usados por cada método, mostrando que BETRALIN e GENLIN têm desempenhos muito parecidos. No entanto, para este conjunto de problemas, GENLIN apresentou resultados um pouco melhores do que BETRALIN. Como mencionado anteriormente, GENCAN possui parâmetros melhor definidos do que BETRA, o que influencia neste resultado. Além

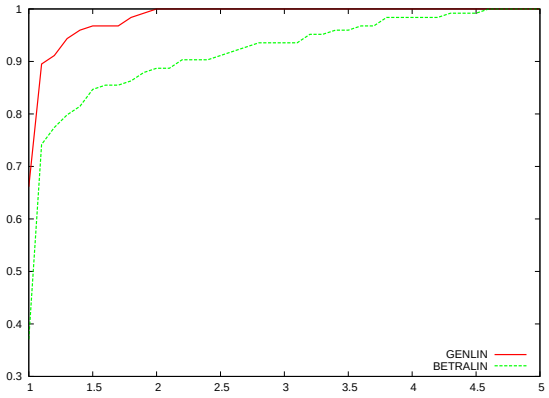


Figura 2.5: Curva de perfil de desempenho comparando Betralin e Genlin usando tempo como medida de desempenho.

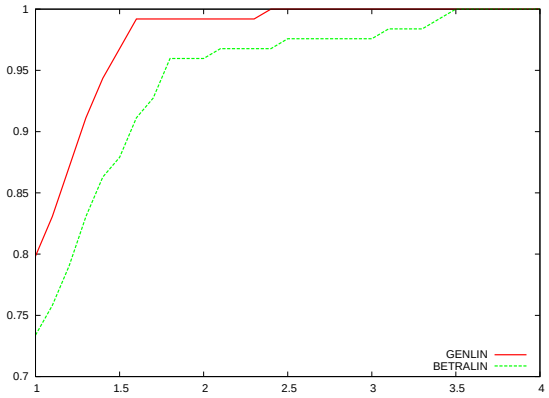


Figura 2.6: Curva de perfil de desempenho comparando Betralin e Genlin usando número de avaliações de função como medida de desempenho.

disso, o fato de GENLIN calcular apenas produtos matriz-vetor (aproveitando a estrutura da matriz Z) e BETRALIN ter de calcular o produto $Z^T\nabla^2\bar{f}Z$, também explica uma certa perda de desempenho de BETRALIN. Assim, nas comparações a seguir, feitas com outros métodos conhecidos para resolver problemas do tipo (2.1), utilizamos apenas GENLIN.

2.7.1.5 Comparação de Genlin com ve11

Nesta seção vamos comparar o desempenho de GENLIN e VE11 na resolução dos problemas da coleção CUTEr presentes na Tabela 2.1. VE11 é uma rotina da coleção HSL que corresponde ao método de Powell descrito em [47], que é, basicamente, um método de restrições ativas que utiliza atualização BFGS da aproximação da Hessiana e busca linear. Esta rotina possui apenas um parâmetro relevante ε_{VE} relacionado à tolerância, que definimos como $\varepsilon_{\text{VE}} = 10^{-8}$.

Como feito com GENLIN, executamos VE11 para resolver cada problema várias vezes, o suficiente para que o tempo total gasto fosse de, pelo menos, 1 segundo. Usamos, então, a média do tempo obtido. É importante observar que, nos casos de problemas sem função objetivo e nos casos em que há um único ponto na região viável, VE11 não conta avaliações de funções.

O iterando final de VE11 não satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$ em 16 dos 127 problemas. Nestes problemas, a explicação para o que acontece é a seguinte:

- o problema STEENBRA é o único problema para o qual VE11 pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas WATER e STATIC3, VE11 pára declarando que o ponto final é viável, mas a busca linear falha em reduzir o valor da função objetivo. Também podem ter ocorrido erros numéricos;
- nos problemas SMBANK, DALLASM, STEENBRB, STEENBRD e STEENBRF, VE11 pára declarando que o ponto é viável, mas erros de arredondamento impedem que haja maior precisão;
- nos problemas HS55, LINSPANH e SPANHYD, VE11 pára declarando que as restrições lineares de igualdade e os limitantes nas variáveis são incompatíveis;
- nos problemas AGG, GMNCASE4, DALLASS, QPCBOEI2 e QPNBOEI2, VE11 pára declarando que foi possível satisfazer as restrições lineares de igualdade e os limitantes nas variáveis, mas as restrições de desigualdade não podem ser satisfeitas.

Nos outros 111 problemas, o iterando final de VE11 é viável com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 84 destes 111 problemas, VE11 pára satisfazendo seu critério de parada relativo a sucesso. Nos 27 problemas restantes, a explicação é a seguinte:

- nos problemas HS268, S268, QCNEW, DUALC8, HS105, DUALC1, HS119, LOADBAL, AVION2, QPCBLEND, QPNBLEND, DUAL1, DUAL2, DUAL3, HYDROELS, GMNCASE1, GMNCASE2, SSEBLIN, PRIMAL1 e QPCSTAIR, VE11 pára declarando que o ponto final é viável, mas a busca linear falha em reduzir o valor da função objetivo. Também podem ter ocorrido erros numéricos;

- nos problemas HS62 e PRIMALC1, VE11 pára declarando que o ponto é viável, mas erros de arredondamento impedem que haja maior precisão;
- no problema HIMMELBJ, VE11 pára declarando que as restrições lineares de igualdade e os limitantes nas variáveis são incompatíveis;
- nos problemas HIMMELBI, QPCBOEI1, QPNBOEI1 e QPNSTAIR, VE11 pára declarando que foi possível satisfazer as restrições lineares de igualdade e os limitantes nas variáveis, mas as restrições de desigualdade não podem ser satisfeitas.

Considerando os 111 problemas nos quais GENLIN e VE11 obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final é equivalente ao obtido por GENLIN. No entanto,

- em 8 problemas (LSNNODOC, QPCBLEND, QPNBLEND, HIMMELBI, QPCBOEI1, QPNBOEI1, QPCSTAIR, QPNSTAIR), GENLIN obteve valor de função menor do que VE11;
- em três problemas (HS54, EQC, HS105), VE11 obteve valor de função menor do que GENLIN;
- em dois problemas (QCNEW, HIMMELBJ), o valor de função final fornecido por VE11 não é definido (mensagem NaN).

Nos 98 problemas restantes, tanto GENLIN como VE11 obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 98 problemas:

- GENLIN realizou menos avaliações de função em 83 casos;
- VE11 realizou menos avaliações de função em 11 casos;
- GENLIN e VE11 realizaram o mesmo número de avaliações de função em 4 casos;
- GENLIN foi mais rápido que VE11 (com tolerância de 10%) em 14 casos;
- VE11 foi mais rápido que GENLIN (com tolerância de 10%) em 79 casos;
- GENLIN e VE11 gastaram a mesma quantidade de tempo em 5 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (11 problemas), GENLIN foi mais rápido que VE11 6 vezes e VE11 foi mais rápido que GENLIN 3 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 2 casos).

Na Tabela 2.5 temos um resumo da comparação entre GENLIN e VE11. Consideramos que GENLIN encontrou melhor valor de função nos dois problemas para os quais VE11 pára em um ponto com valor de função indefinido (mensagem NaN).

Considerando os 98 problemas para os quais tanto GENLIN como VE11 chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 2.7, usamos o tempo de CPU como medida de desempenho. Na Figura 2.8, usamos o número de avaliações de função como medida de desempenho. GENLIN tem robustez muito

	GENLIN	VE11
Problemas viáveis	127/127	111/127
Satisfez critérios de convergência	125/127	84/111
Valor de função melhor	10/111	3/111
Menos avaliações de função	83/98	11/98
Tempo menor (10% tolerância)	14/98	79/98

Tabela 2.5: Comparação entre GENLIN e VE11 para problemas selecionados da coleção CUTEr

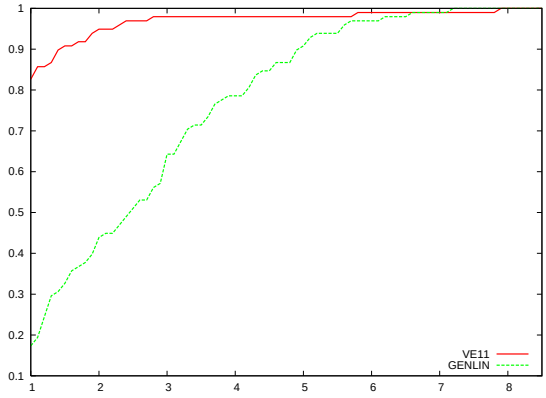


Figura 2.7: Curva de perfil de desempenho comparando GENLIN e VE11 usando tempo como medida de desempenho.

superior a VE11. Quando utilizamos o número de avaliações de função como medida de desempenho, podemos ver que GENLIN é mais eficiente. Usando o tempo gasto para resolver os problemas como medida de desempenho, VE11 é mais eficiente.

2.7.1.6 Comparação de Genlin com Minos

Nesta seção vamos comparar o desempenho de GENLIN e MINOS [41] na resolução dos problemas da coleção CUTEr presentes na Tabela 2.1. Em essência, MINOS é uma extensão do método Simplex revisado de Dantzig. Usando a terminologia associada, ele pode ser descrito como uma extensão que permite que mais de m variáveis sejam básicas (m é o número de restrições lineares), que também pode lidar com termos não-lineares usando um procedimento do tipo quase-Newton. Como MINOS trabalha apenas com restrições lineares de igualdade e restrições de caixa, ele faz uso de variáveis de folga para eliminar restrições lineares de desigualdade.

Para MINOS, usamos os parâmetros padrão, mudando apenas as tolerâncias para viabilidade e otimalidade para[1] para 10^{-8}. O tempo medido foi o utilizado para resolver cada problema uma única vez. É importante observar que, nos casos de problemas com função objetivo linear ou constante, MINOS não conta avaliações de funções.

[1] Acrescentando as linhas `Feasibility Tolerance 1.0d-08` e `Optimality Tolerance 1.0d-08` ao arquivo MINOS.SPC.

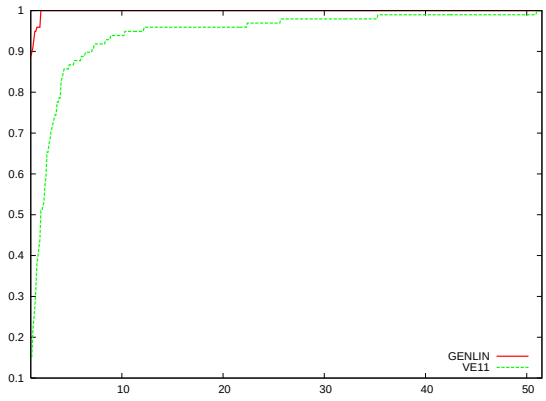


Figura 2.8: Curva de perfil de desempenho comparando GENLIN e VE11 usando número de avaliações de função como medida de desempenho.

O iterando final de MINOS não satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$ em dois dos 127 problemas. Estes problemas são HS54 e AGG. Em ambos os casos, MINOS pára declarando sucesso. Nos outros 125 problemas, o iterando final de MINOS é viável com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 119 destes 125 problemas, MINOS pára satisfazendo seu critério de parada relativo a sucesso. Nos 6 problemas restantes, a explicação é a seguinte:

- no problema STATIC3, MINOS pára declarando que o problema é ilimitado ou mal-escalado;
- no problema HS55, MINOS pára declarando que o ponto não pode ser melhorado;
- nos problemas HS118, HIMMELBJ, SPANHYD e STEENBRD, MINOS pára declarando que uma solução próxima da ótima foi encontrada.

Considerando os 125 problemas nos quais GENLIN e MINOS obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final é equivalente ao obtido por GENLIN. No entanto,

- em 6 problemas (HS41, PENTAGON, QC, HS118, QPNBLEND, STEENBRD), GENLIN obteve valor de função menor do que MINOS;
- em 6 problemas (HATFLDH, LIN, EQC, HS105, DEGENLPA, QPNBOEI1), MINOS obteve valor de função menor do que GENLIN;
- em um problema (HIMMELBJ), o valor de função final fornecido por MINOS não é definido (mensagem NaN).

Nos 111 problemas restantes, tanto GENLIN como MINOS obtiveram pontos viáveis com valores de função equivalentes (excluindo aqui o problema STATIC3, para o qual tanto GENLIN como MINOS param porque o problema parece ilimitado). Considerando estes 111 problemas:

- GENLIN realizou menos avaliações de função em 84 casos;
- MINOS realizou menos avaliações de função em 26 casos;

- Genlin e Minos realizaram o mesmo número de avaliações de função em um caso.
- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo (40 problemas), Genlin foi mais rápido que Minos (com tolerância de 10%) 17 vezes e Minos foi mais rápido que Genlin (com tolerância de 10%) 22 vezes (ambos os métodos gastaram a mesma quantidade de tempo em um caso).
- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (27 problemas), Genlin foi mais rápido que Minos 8 vezes e Minos foi mais rápido que Genlin 18 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 1 caso).

Na Tabela 2.6 temos um resumo da comparação entre Genlin e Minos. Consideramos que Genlin encontrou melhor valor de função no problema para o qual Minos pára em um ponto com valor de função indefinido (mensagem NaN). Na comparação do tempo gasto pelos métodos para resolver os problemas, consideramos apenas os problemas para os quais algum dos métodos gastou pelo menos 0.01 segundo.

	Genlin	Minos
Problemas viáveis	127/127	125/127
Satisfez critérios de convergência	125/127	119/125
Valor de função melhor	7/125	6/125
Menos avaliações de função	84/111	26/111
Tempo menor (10% tolerância)	17/40	22/40

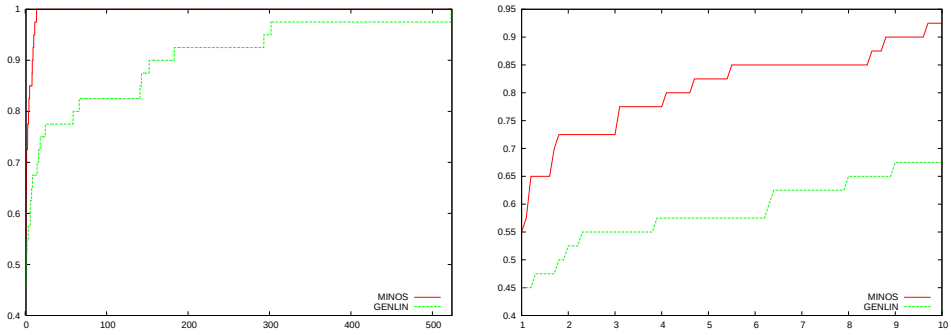
Tabela 2.6: Comparação entre Genlin e Minos para problemas selecionados da coleção CUTEr

Considerando os 111 problemas para os quais tanto Genlin como Minos chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 2.9, usamos o tempo de CPU como medida de desempenho, eliminando os problemas para os quais ambos os métodos gastaram menos de 0.01 segundo. Na Figura 2.10, usamos o número de avaliações de função como medida de desempenho. Genlin tem robustez superior a Minos. Quando utilizamos o número de avaliações de função como medida de desempenho, podemos ver que Genlin é mais eficiente. Usando o tempo gasto para resolver os problemas como medida de desempenho, Minos é mais eficiente.

2.7.1.7 Comparação de Genlin com Ipopt

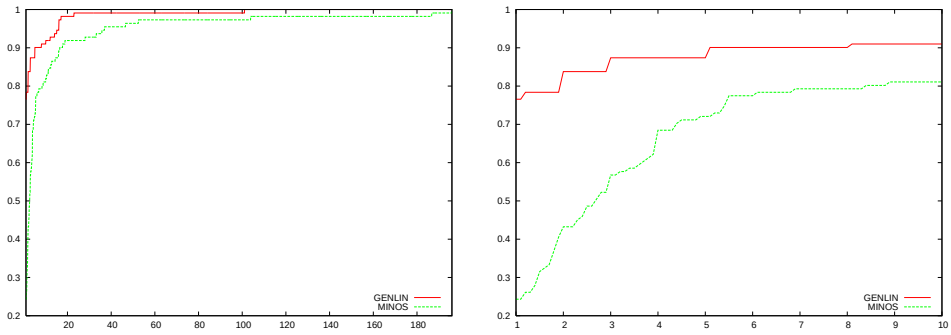
Nesta seção vamos comparar o desempenho de Genlin e Ipopt [54] na resolução dos problemas da coleção CUTEr presentes na Tabela 2.1. Ipopt é um método de pontos interiores que, a cada iteração, utiliza um algoritmo do tipo Newton para resolver os subproblemas. Ipopt usa variáveis de folga para transformar as restrições de desigualdade em restrições de igualdade. Utilizamos os parâmetros padrão para Ipopt (versão-3.3.2). O tempo medido foi o utilizado para resolver cada problema uma única vez.

O iterando final de Ipopt não satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$ em um dos 127 problemas. Este problema é himmelbj, para o qual Ipopt pára declarando que



(a) (b)

Figura 2.9: Curva de perfil de desempenho comparando GENLIN e MINOS usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).



(a) (b)

Figura 2.10: Curva de perfil de desempenho comparando GENLIN e MINOS usando número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

a restauração falhou. Nos outros 126 problemas, o iterando final de IPOPT é viável com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 121 destes 126 problemas, IPOPT pára satisfazendo seu critério de parada relativo a sucesso. Nos 5 problemas restantes, a explicação é a seguinte:

- no problema STATIC3, IPOPT pára declarando que o problema pode ser ilimitado;
- nos problemas EQC e DALLASM, IPOPT pára declarando que o problema foi resolvido até um nível aceitável;
- no problema EQCNEW, IPOPT pára declarando que a restauração falhou;
- no problema LIN, IPOPT pára declarando que a direção de busca se tornou muito pequena.

Considerando os 126 problemas nos quais GENLIN e IPOPT obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final é equivalente ao obtido por GENLIN. No entanto,

- em 12 problemas (HS44, EXPFITA, EXPFITB, EXPFITC, HS268, S268, HS55, PENTAGON, EQC, MAKELA4, GOFFIN, PRIMAL1), GENLIN obteve valor de função menor do que IPOPT;
- em 17 problemas (BIGGSC4, HATFLDH, HS54, QCNEW, HS105, DUALC1, DEGENLPA, DEGENLPB, QPCBLEND, QPNBLEND, QPNBOEI2, GMNCASE1, GMNCASE2, GMNCASE3, PRIMALC5, QPNBOEI1, STEENBRD), IPOPT obteve valor de função menor do que GENLIN;
- em um problema (LIN), o valor de função final fornecido por IPOPT não é definido (mensagem NaN).

Nos 95 problemas restantes, tanto GENLIN como IPOPT obtiveram pontos viáveis com valores de função equivalentes (excluindo aqui o problema STATIC3, para o qual tanto GENLIN como IPOPT param porque o problema parece ilimitado). Considerando estes 95 problemas:

- GENLIN realizou menos avaliações de função em 71 casos;
- IPOPT realizou menos avaliações de função em 15 casos;
- GENLIN e IPOPT realizaram o mesmo número de avaliações de função em 9 casos.
- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo (54 problemas), GENLIN foi mais rápido que IPOPT (com tolerância de 10%) 38 vezes e IPOPT foi mais rápido que GENLIN (com tolerância de 10%) 16 vezes.
- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (26 problemas), GENLIN foi mais rápido que IPOPT 12 vezes e IPOPT foi mais rápido que GENLIN 14 vezes.

Na Tabela 2.7 temos um resumo da comparação entre GENLIN e IPOPT. Consideramos que GENLIN encontrou melhor valor de função no problema para o qual IPOPT pára em um ponto com valor de função indefinido (mensagem NaN). Na comparação do tempo gasto pelos

	GENLIN	IPOPT
Problemas viáveis	127/127	126/127
Satisfez critérios de convergência	125/127	121/126
Valor de função melhor	13/126	17/126
Menos avaliações de função	71/95	15/95
Tempo menor (10% tolerância)	38/54	16/54

Tabela 2.7: Comparação entre GENLIN e IPOPT para problemas selecionados da coleção CUTEr

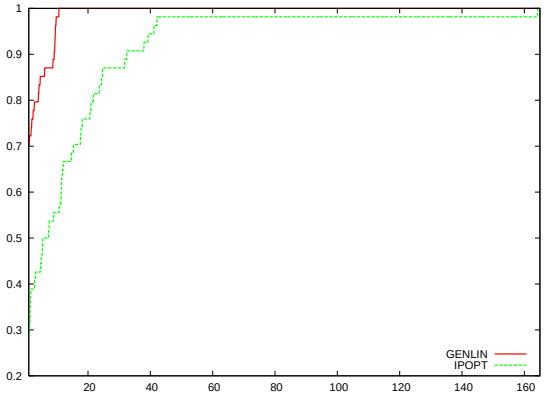


Figura 2.11: Curva de perfil de desempenho comparando GENLIN e IPOPT usando tempo como medida de desempenho.

métodos para resolver os problemas, consideramos apenas os problemas para os quais algum dos métodos gastou pelo menos 0.01 segundo.

Considerando os 95 problemas para os quais tanto GENLIN como IPOPT chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 2.11, usamos o tempo de CPU como medida de desempenho, eliminando os problemas para os quais ambos os métodos gastaram menos de 0.01 segundo. Na Figura 2.12, usamos o número de avaliações de função como medida de desempenho. Note que GENLIN resolve mais problemas do que IPOPT. No entanto, IPOPT obtém melhores valores de função mais vezes. Em termos de eficiência, tanto utilizando o número de avaliações de função como utilizando o tempo como medida de desempenho, podemos ver que GENLIN é mais eficiente do que IPOPT.

2.7.1.8 Comparação de Genlin com Lancelot B

Nesta seção vamos comparar o desempenho de GENLIN e LANCELOT B [18, 19, 20] na resolução dos problemas da coleção CUTEr presentes na Tabela 2.1. Usamos os parâmetros padrão de LANCELOT B, mudando apenas as tolerâncias para viabilidade e otimalidade[2] para 10^{-8}. O tempo medido foi o utilizado para resolver cada problema uma única vez.

[2] Acrescentamos as linhas `primal-accuracy-required 1.0d-08` e `dual-accuracy-required 1.0d-08` ao arquivo RUNLANB.SPC.

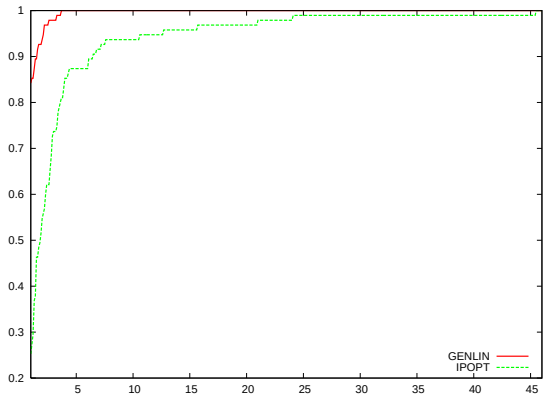


Figura 2.12: Curva de perfil de desempenho comparando GENLIN e IPOPT usando número de avaliações de função como medida de desempenho.

Como LANCELOT B transforma problemas sem função objetivo em problemas de minimização irrestrita, eliminamos os 4 problemas da Tabela 2.1 com esta característica. São eles: BOOTH, HIMMELBA, ZANGWIL3 e RES.

O iterando final de LANCELOT B não satisfaz viabilidade com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$ em 9 dos 123 problemas restantes. Estes problemas são HIMMELBJ, AVION2, QPNBOEI2, AGG, PRIMALC1, QPCBOEI1, QPNBOEI1, STATIC3 e STEENBRD. Em todos os casos, LANCELOT B pára por atingir o número máximo de iterações permitido.

Nos outros 114 problemas, o iterando final de LANCELOT B é viável com tolerância $\varepsilon_{\text{viab}} = 10^{-8}$. Em 88 destes 114 problemas, LANCELOT B pára satisfazendo seu critério de parada relativo a sucesso. Nos 26 problemas restantes (SIPOW1, SIPOW2, HS62, OET1, TFI3, OET3, SIPOW3, SIPOW4, EXPFITC, HS86, HS54, DUALC2, DUALC5, DUALC8, DUALC1, DEGENLPA, DEGENLPB, KSIP, LOADBAL, QPCBLEND, QPNBLEND, QPCBOEI2, SSEBLIN, PRIMALC2, QPCSTAIR, QPNSTAIR), LANCELOT B pára declarando que o passo é muito pequeno e não gera mudança na função objetivo.

Considerando os 114 problemas nos quais GENLIN e LANCELOT B obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final é equivalente ao obtido por GENLIN. No entanto,

- em 2 problemas (HS44NEW, HS54), GENLIN obteve valor de função menor do que LANCELOT B;
- em 4 problemas (BIGGSC4, HATFLDH, LIN, QPCBLEND), LANCELOT B obteve valor de função menor do que GENLIN.

Nos 108 problemas restantes, tanto GENLIN como LANCELOT B obtiveram pontos viáveis com valores de função equivalentes. Como não dispomos do número de avaliações de função utilizados por LANCELOT B, analisaremos apenas o tempo gasto. Considerando estes 108 problemas:

- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo

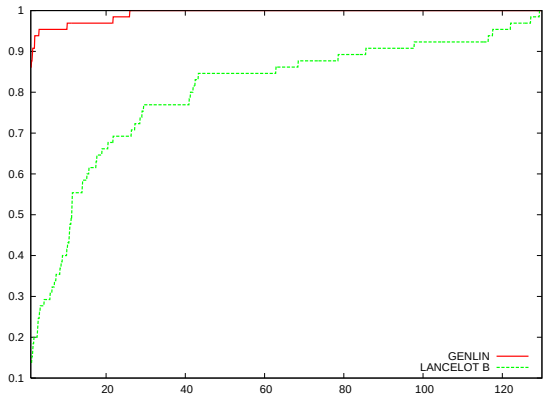


Figura 2.13: Curva de perfil de desempenho comparando GENLIN e LANCELOT B usando tempo como medida de desempenho.

(65 problemas), GENLIN foi mais rápido que LANCELOT B (com tolerância de 10%) 56 vezes e LANCELOT B foi mais rápido que GENLIN (com tolerância de 10%) 9 vezes.

- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (34 problemas), GENLIN foi mais rápido que LANCELOT B 26 vezes e LANCELOT B foi mais rápido que GENLIN 8 vezes.

Na Tabela 2.8 temos um resumo da comparação entre GENLIN e LANCELOT B. Na comparação do tempo gasto pelos métodos para resolver os problemas, consideramos apenas os problemas para os quais algum dos métodos gastou pelo menos 0.01 segundo.

	GENLIN	LANCELOT B
Problemas viáveis	123/123	114/123
Satisfez critérios de convergência	121/123	88/114
Valor de função melhor	2/114	4/114
Tempo menor (10% tolerância)	56/65	9/65

Tabela 2.8: Comparação entre GENLIN e LANCELOT B para problemas selecionados da coleção CUTEr

Considerando os 108 problemas para os quais tanto GENLIN como LANCELOT B chegaram a um ponto viável com mesmo valor de função, construímos um gráfico de perfil de desempenho. Usamos o tempo de CPU como medida de desempenho, eliminando os problemas para os quais ambos os métodos gastaram menos de 0.01 segundo (veja Figura 2.13). Note que GENLIN é mais robusto e mais eficiente do que LANCELOT B.

2.7.2 Problema de estimação de parâmetros em um problema de óptica

Um filme fino é uma fina camada de material de espessura normalmente inferior a um mícron. Este é depositado sobre um substrato transparente, de espessura muito maior. Como medir a espessura de filmes finos é um processo caro, estamos interessados em estimar a espessura e

algumas constantes ópticas (índice de refração e coeficiente de absorção) de um filme. Já que filmes finos são utilizados na produção de semicondutores, revestimentos de diversos materiais, biotecnologia e na geração e conservação de energia (painéis solares), é grande a importância da resolução deste problema.

A idéia é utilizar alguns dados facilmente medidos e o comportamento físico teórico conhecido do sistema e modelar o problema de estimar a espessura e as constantes ópticas como um problema de programação não-linear (veja [5] e [16]). Como veremos a seguir, este problema possui apenas restrições lineares e de caixa. Daí a idéia de utilizar BETRALIN e GENLIN para resolvê-lo.

A transmitância T de um filme fino absorvente depositado sobre um substrato grosso transparente é dada por (veja [51, 52])

$$T = \frac{Ax}{B - Cx + Dx^2},$$

onde

$$\begin{aligned} A &= 16s(n^2+\kappa^2), \\ B &= [(n+1)^2+\kappa^2][(n+1)(n+s^2)+\kappa^2], \\ C &= [(n^2-1+\kappa^2)(n^2-s^2+\kappa^2) - 2\kappa^2(s^2+1)]2\cos\varphi \\ &\quad -\kappa[2(n^2-s^2+\kappa^2)+(s^2+1)(n^2-1+\kappa^2)]2\mathrm{sen}\varphi, \\ D &= [(n-1)^2+\kappa^2][(n-1)(n-s^2)+\kappa^2], \\ \varphi &= 4\pi nd/\lambda, \quad x = \exp(-\alpha d), \quad \alpha = 4\pi\kappa/\lambda. \end{aligned}$$

Nas expressões acima, λ é o comprimento de onda, $s = s(\lambda)$ é o índice de refração do substrato transparente (considerado conhecido), $n = n(\lambda)$ é o índice de refração do filme, $\kappa = \kappa(\lambda)$ é o coeficiente de atenuação do filme, $\alpha = \alpha(\lambda)$ é o coeficiente de absorção do filme e d é a espessura do filme.

Um conjunto de dados experimentais $(\lambda_i, T^{\mathrm{obs}}(\lambda_i))$, $\lambda_{\min} \le \lambda_i < \lambda_{i+1} \le \lambda_{\max}$, para $i = 1, \ldots, N$, é dado e queremos estimar d, $n(\lambda)$ e $\kappa(\lambda)$. Este problema é altamente indeterminado. Mesmo com d conhecido, queremos que valha

$$T[\lambda_i, s(\lambda_i), d, n(\lambda_i), \kappa(\lambda_i)] = T^{\mathrm{obs}}(\lambda_i),\ i = 1, \ldots, N.$$

Ou seja, temos uma única equação para cada duas incógnitas ($n(\lambda_i)$ e $\kappa(\lambda_i)$). O conjunto de pontos que satisfazem esta equação, para um dado d, é infinito. No entanto, algumas restrições físicas conhecidas reduzem o espaço de possíveis soluções. Na vizinhança do pico de absorção fundamental, essas restrições são:

R1. $n(\lambda) \ge 1$ e $\kappa(\lambda) \ge 0$ para todo $\lambda \in [\lambda_{\min}, \lambda_{\max}]$;

R2. $n(\lambda)$ e $\kappa(\lambda)$ são funções decrescentes em λ;

R3. $n(\lambda)$ é convexa;

R4. existe $\lambda_{\mathrm{infl}} \in [\lambda_{\min}, \lambda_{\max}]$ tal que $\kappa(\lambda)$ é convexa se $\lambda \ge \lambda_{\mathrm{infl}}$ e côncava se $\lambda \le \lambda_{\mathrm{infl}}$.

Utilizando os comprimentos de onda $\lambda_i,\ i = 1, \ldots, N$, eqüidistantes no intervalo $[\lambda_{\min}, \lambda_{\max}]$, obtemos uma discretização das funções acima. Note que $\lambda_i = \lambda_{\min} + (i-1)\frac{\lambda_{\max}-\lambda_{\min}}{N-1}$. Denotaremos por $n_i = n(\lambda_i)$, $\kappa_i = \kappa(\lambda_i)$, $s_i = s(\lambda_i)$ e $T_i^{\text{obs}} = T^{\text{obs}}(\lambda_i)$.

A restrição R2 pode ser escrita como

$$n_{i+1} \leq n_i,\ i = 1, \ldots, N-1,$$

$$\kappa_{i+1} \leq \kappa_i,\ i = 1, \ldots, N-1.$$

A restrição R3 pode ser escrita como $n''(\lambda) \geq 0$, ou

$$n_i \leq n_{i-1} + \frac{n_{i+1} - n_{i-1}}{\lambda_{i+1} - \lambda_{i-1}}(\lambda_i - \lambda_{i-1}), \Rightarrow$$

$$n_i \leq \frac{1}{2}(n_{i+1} + n_{i-1}),\ i = 2, \ldots, N-1.$$

A restrição R4 pode ser escrita como $\kappa''(\lambda) \geq 0$, para $\lambda \geq \lambda_{\text{infl}}$ e $\kappa''(\lambda) \leq 0$, para $\lambda \leq \lambda_{\text{infl}}$, ou seja,

$$\kappa_i \geq \kappa_{i-1} + \frac{\kappa_{i+1} - \kappa_{i-1}}{\lambda_{i+1} - \lambda_{i-1}}(\lambda_i - \lambda_{i-1}) \Rightarrow$$

$$\kappa_i \geq \frac{1}{2}(\kappa_{i+1} + \kappa_{i-1}),\ \lambda_{i+1} \leq \lambda_{\text{infl}},$$

$$\kappa_i \leq \kappa_{i-1} + \frac{\kappa_{i+1} - \kappa_{i-1}}{\lambda_{i+1} - \lambda_{i-1}}(\lambda_i - \lambda_{i-1}) \Rightarrow$$

$$\kappa_i \leq \frac{1}{2}(\kappa_{i+1} + \kappa_{i-1}),\ \lambda_{i-1} \geq \lambda_{\text{infl}}.$$

Para calcular os valores de d, $n(\lambda)$ e $\kappa(\lambda)$, resolvemos o seguinte problema de programação não-linear:

$$\begin{array}{lllll}
\text{Minimizar} & & \sum_{i=1}^{N}[T_i^{\text{obs}} - T(\lambda_i, s_i, d, n_i, \kappa_i)]^2 & & \\
\text{sujeita a} & n_{i+1} & \leq n_i, & i = 1, \ldots, N-1, & \\
& \kappa_{i+1} & \leq \kappa_i, & i = 1, \ldots, N-1, & \\
& n_i & \leq \frac{1}{2}(n_{i+1} + n_{i-1}), & i = 2, \ldots, N-1, & \\
& \kappa_i & \geq \frac{1}{2}(\kappa_{i+1} + \kappa_{i-1}), & \lambda_{i+1} \leq \lambda_{\text{infl}}, & \\
& \kappa_i & \leq \frac{1}{2}(\kappa_{i+1} + \kappa_{i-1}), & \lambda_{i-1} \leq \lambda_{\text{infl}}, & \\
& n_i & \geq 1, & i = 1, \ldots, N, & \\
& \kappa_i & \geq 0, & i = 1, \ldots, N. &
\end{array} \tag{2.14}$$

Note que, fixado d, temos $2N$ variáveis (n_i e κ_i, para $i = 1, \ldots, N$) e $4N-6$ restrições lineares.

Por saber que, na região de interesse, a função κ é convexa, fixamos λ_{infl} em $\lambda_{\min}$. Dados N, $\lambda_{\min}$, $\lambda_{\max}$ e as observações T_i^{obs}, $i = 1, \ldots, N$, fixamos o valor de d. Utilizando uma estimativa inicial para n e κ, usamos Betralin (Genlin) para resolver a instância do problema (2.14).

Como em [5], dado um intervalo $[d_{\min}, d_{\max}]$ no qual sabe-se estar a espessura verdadeira, utilizamos como estimativa para espessura do filme os valores no intervalo $[d_{\min}, d_{\max}]$, espaçados de 10 em 10. Ou seja, $d_{\min}, d_{\min} + 10, \ldots, d_{\max}$.

Utilizamos seis estimativas iniciais de n. Todas elas são funções lineares decrescentes, que vão do ponto (x_1, y_1) ao ponto (x_2, y_2), sendo os pares $[(x_1, y_1); (x_2, y_2)]$ os seguintes: $[(\lambda_{\min}, 3); (\lambda_{\max}, 2)]$, $[(\lambda_{\min}, 4); (\lambda_{\max}, 2)]$, $[(\lambda_{\min}, 5); (\lambda_{\max}, 2)]$, $[(\lambda_{\min}, 4); (\lambda_{\max}, 3)]$, $[(\lambda_{\min}, 5); (\lambda_{\max}, 3)]$, $[(\lambda_{\min}, 5); (\lambda_{\max}, 4)]$.

Para as estimativas iniciais de κ, utilizamos a função linear por partes que vale 0.1 em $\lambda_{\min}$, 0.01 em $\lambda_{\min} + 0.2(\lambda_{\max} - \lambda_{\min})$, e 10^{-10} em $\lambda_{\max}$.

Para resolver o problema (2.14) utilizamos cada combinação das estimativas de n, κ e d citadas acima e utilizamos BETRALIN (GENLIN) para resolver as instâncias do problema. Dentre todas as combinações usadas, ficamos com a solução $\bar{n}$ e $\bar{\kappa}$ que fornece menor valor da função objetivo. Chamemos $\bar{d}$ o valor de d utilizado na obtenção desta solução. Definimos então o intervalo $[\bar{d} - 10, \bar{d} + 10]$. Fixamos d como os pontos deste intervalo espaçados um a um, $\bar{n}$ e $\bar{\kappa}$ como ponto inicial e resolvemos estas instâncias do problema (2.14) utilizando BETRALIN (GENLIN). Dentre as soluções obtidas para cada instância, consideramos como solução do problema (2.14) a espessura d^* e a solução n^* e κ^* que fornecem o menor valor de função objetivo.

Para verificar se este método de resolução do problema (2.14) é confiável, utilizamos transmitância gerada por computador de filmes *gedanken*, para os quais temos os resultados verdadeiros para comparar com a solução fornecida pelo método. Os conjuntos de filme e substrato utilizados nos experimentos são:

- Filme A: este filme gerado por computador simula um filme fino de Silício amorfo hidrogenado (*a-Si:H*), depositado sobre um substrato de vidro, com espessura $d^{\text{real}} = 100$ nm. O intervalo de comprimentos de onda usado vai de 550 nm a 1530 nm;
- Filme B: este filme, como o Filme A, simula um filme fino de Silício amorfo hidrogenado (*a-Si:H*) depositado sobre um substrato de vidro. Sua espessura é $d^{\text{real}} = 600$ nm e o intervalo de comprimentos de onda usado vai de 620 nm a 1610 nm;
- Filme C: simula um filme fino de Germânio amorfo hidrogenado (*a-Ge:H*), depositado sobre um substrato cristalino de Silício, com espessura $d^{\text{real}} = 100$ nm e intervalo de comprimentos de onda de 1250 nm a 2537 nm;
- Filme D: este filme, como o Filme C, simula um filme fino de Germânio amorfo hidrogenado (*a-Ge:H*), depositado sobre um substrato cristalino de Silício. Sua espessura é $d^{\text{real}} = 600$ nm e o intervalo de comprimentos de onda vai de 1250 nm a 2537 nm;
- Filme E: simula um filme de óxido de metal, com espessura $d^{\text{real}} = 80$, depositado sobre vidro. O intervalo de comprimentos de onda usado vai de 360 nm a 657 nm.

O índice de refração $s(\lambda)$ dos substratos de vidro e Silício usados nos experimentos são dados, respectivamente, por

$$s_{\text{vidro}}(\lambda) = \sqrt{1 + \frac{1}{\left(0.7568 - \frac{7930}{\lambda^2}\right)}},$$

$$s_{\text{Si}}(\lambda) = 3.71382 - 8.69123 \times 10^{-5}\lambda - 2.47125 \times 10^{-8}\lambda^2 + 1.04677 \times 10^{-11}\lambda^3.$$

O índice de refração n^{real} e o coeficiente de absorção α^{real} de *a-Si:H* são dados por

$$n^{\text{real}}(\lambda) = \sqrt{1 + \frac{1}{\left(0.09195 - \frac{12600}{\lambda^2}\right)}},$$

$$\ln(\alpha^{\text{real}}(E)) = \begin{cases} 6.5944 \times 10^{-6} \exp(9.0846E) - 16.102, & 0.60 < E < 1.40, \\ 20E - 41.9, & 1.40 < E < 1.75, \\ \sqrt{59.56E - 102.1} - 8.391, & 1.75 < E < 2.29. \end{cases}$$

O índice de refração n^{real} e o coeficiente de absorção α^{real} de *a-Ge:H* são dados por

$$n^{\text{real}}(\lambda) = \sqrt{1 + \frac{1}{\left(0.065 - \frac{15000}{\lambda^2}\right)}},$$

$$\ln(\alpha^{\text{real}}(E)) = \begin{cases} 6.5944 \times 10^{-6} \exp(13.629E) - 16.102, & 0.48 < E < 0.93, \\ 30E - 41.9, & 0.93 < E < 1.17, \\ \sqrt{89.34E - 102.1} - 8.391, & 1.17 < E < 1.50. \end{cases}$$

O índice de refração n^{real} e o coeficiente de absorção α^{real} do óxido de metal são dados por

$$n^{\text{real}}(\lambda) = \sqrt{1 + \frac{1}{\left(0.3 - \frac{10000}{\lambda^2}\right)}},$$

$$\ln(\alpha^{\text{real}}(E)) = \ 6.5944 \times 10^{-6} \exp(4.0846E) - 11.02, \quad 0.50 < E < 3.50.$$

Nas expressões acima, o comprimento de onda λ é dado em nanômetros (nm), a energia do fóton $E = 1240/\lambda$ é dada em eV e o coeficiente de absorção α é dado em nm^{-1}. Note que κ pode ser obtido a partir de α.

Para todos os experimentos, usamos $N = 100$. Os valores de λ_i utilizados são os 100 pontos do intervalo fechado $[\lambda_{\min}, \lambda_{\max}]$ igualmente espaçados. Para cada conjunto de filme e substrato descrito acima, calculamos os valores verdadeiros s^{real}, n^{real} e κ^{real} utilizando cada um desses λ_i. A partir destes valores, calculamos a transmitância verdadeira dos filmes T^{real}. Arredondamos os valores de T^{real} para quatro casas decimais, para levar em conta erros de precisão que poderiam ser obtidos em uma medição real. Precisamos de um intervalo inicial para a espessura d. Utilizamos $d^{\min} = 0.5d^{\text{real}}$ e $d^{\max} = 1.5d^{\text{real}}$.

Tanto em Betralin como em Genlin, utilizamos precisão $\varepsilon_{\text{viab}} = 10^{-8}$ e $\varepsilon_{\text{otim}} = 10^{-4}$ para o critério de convergência. Ou seja, mantemos a viabilidade das caixas exatamente e das restrições lineares com precisão 10^{-8} (norma infinito das restrições lineares é, no máximo, 10^{-8}) a cada iteração e o gradiente projetado no ponto final deve ser menor ou igual a 10^{-4}. Na primeira parte do método, quando os valores estimados da espessura são espaçados de 10 em 10, o número máximo de iterações e de avaliações de função permitidos são 1000 e 5000, respectivamente. Na segunda parte, quando os valores estimados de espessura são espaçados de um em um, os números máximos de iterações e avaliações de função são 2500 e 12500, respectivamente. Os demais parâmetros são os mesmos mencionados na Seção 2.6.

As Figuras 2.14 a 2.22 mostram os resultados obtidos por BETRALIN e GENLIN na solução do problema (2.14) para os conjuntos de filme e substrato Filme A a Filme E. Na Tabela 2.9 temos os valores das espessuras, em nm, recuperados pelos dois métodos.

		BETRALIN			GENLIN		
Instância	Esp verdadeira	Espessura	Erro quadrático	Tempo	Espessura	Erro quadrático	Tempo
Filme A	100	100	4.46E−08	327.68	100	5.25E−07	250.68
Filme B	600	600	2.45E−06	2235.33	600	1.93E−07	1635.60
Filme C	100	101	2.83E−07	172.86	105	1.15E−07	119.46
Filme D	600	600	1.29E−06	924.26	600	1.69E−07	841.28
Filme E	80	81	6.00E−08	314.58	80	1.26E−07	239.99

Tabela 2.9: Espessuras (em nanômetros) verdadeiras e recuperadas, erro quadrático e tempo (em segundos) gasto por BETRALIN e GENLIN para resolver as instâncias Filme A a Filme E.

Os valores de espessura recuperados por BETRALIN e GENLIN estão muito próximos dos valores reais e similares aos obtidos em [5]. Note que o tempo gasto por GENLIN para resolver o problema é menor do que o gasto por BETRALIN. Com relação às constantes ópticas recuperadas, no caso do Filme C (Figura 2.18), ambos os métodos obtiveram uma aproximação ruim para o coeficiente de absorção, o que é esperado, dado o alto grau de indeterminação do modelo para este filme. Note que, para o Filme D (Figura 2.20), a recuperação do coeficiente de absorção obtida por GENLIN foi um pouco melhor do que a obtida por BETRALIN. No restante dos casos, os resultados obtidos por ambos os métodos foram excelentes.

2.8 Conclusões

Implementamos, em Fortran 77, dois métodos para resolver o problema (2.1) e os chamamos de BETRALIN e GENLIN. Utilizamos os 133 problemas da coleção CUTER com até 500 variáveis, apenas restrições lineares e de caixa, de 1 a 2000 restrições lineares para verificar o desempenhos destes novos métodos.

Comparamos o desempenho dos métodos com ALGENCAN-B e ALGENCAN-G. BETRALIN e GENLIN, que trabalham explicitamente com as restrições lineares, se mostraram muito mais eficientes e robustos do que ALGENCAN-B e ALGENCAN-G, respectivamente. Isso indica, como esperávamos, que trabalhar explicitamente com as restrições lineares é melhor do que penalizá-las.

Comparamos o desempenho de BETRALIN com o desempenho de GENLIN. BETRALIN se mostrou um pouco menos robusto do que GENLIN, já que este obtém valor de função melhor em dois problemas. Apesar de ambos os métodos apresentarem índices de eficiência muito parecidos, GENLIN se mostrou um pouco mais eficiente. Isso provavelmente se deve ao fato de GENCAN possuir parâmetros melhor definidos do que BETRA, e ao fato de GENLIN calcular apenas produtos matriz-vetor (aproveitando a estrutura da matriz Z) e BETRALIN ter de calcular o produto $Z^T \nabla^2 \bar{f} Z$.

Comparamos também o desempenho de GENLIN com VE11, MINOS, IPOPT e LANCELOT B. Com relação a VE11, GENLIN se mostrou muito mais robusto. Utilizando o número de avaliações de função como medida de desempenho, GENLIN é mais eficiente. Utilizando

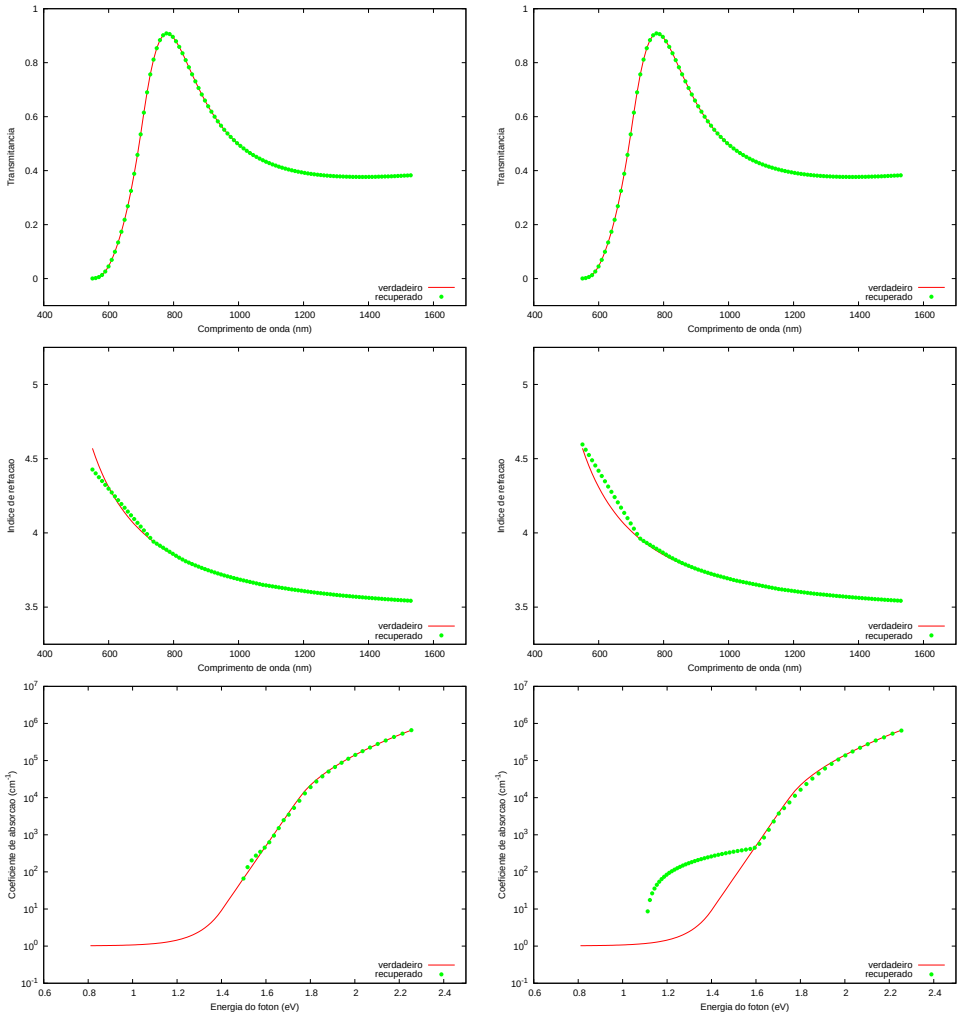


Figura 2.14: Valores reais e recuperados por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) para transmitância, índice de reflexão e coeficiente de absorção do Filme A.

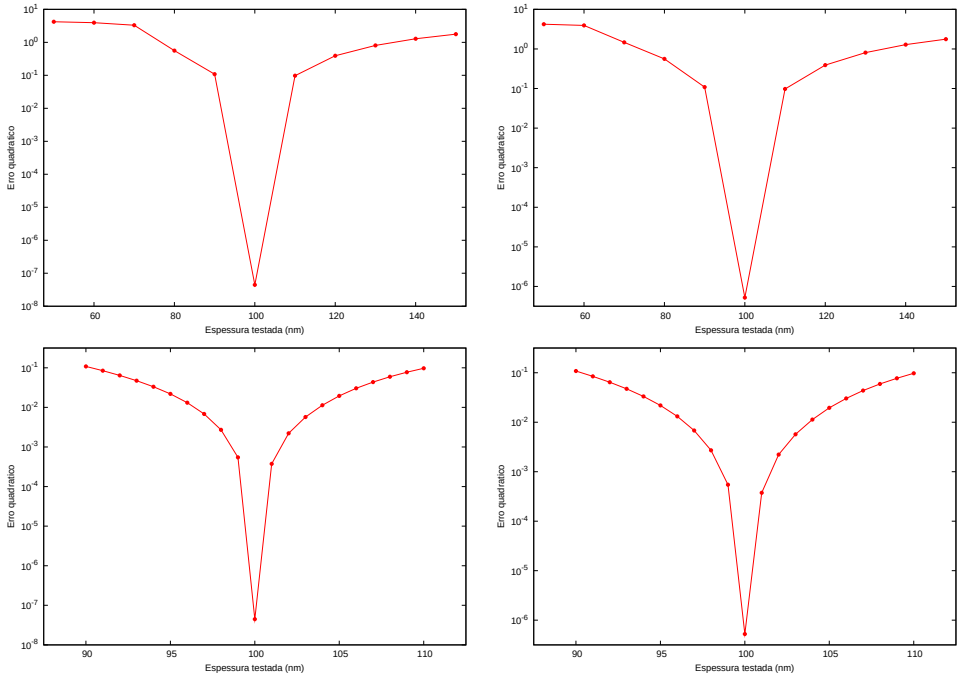


Figura 2.15: Erro quadrático obtido por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) durante o processo de otimização em relação aos valores testados para espessura do Filme A.

tempo, VE11 é mais eficiente. O mesmo acontece quando comparamos GENLIN e MINOS, com a diferença que a robustez de GENLIN com relação a MINOS não é tão grande como em relação a VE11.

Comparando GENLIN com IPOPT, temos que GENLIN resolve mais problemas, porém IPOPT consegue melhor valor de função mais vezes. Em termos de eficiência, GENLIN é mais rápido e utiliza menor número de avaliações de função do que IPOPT.

Quando comparamos GENLIN e LANCELOT B, temos que GENLIN é mais robusto e eficiente do que LANCELOT B, considerando o tempo gasto para resolver cada problema como medida de eficiência.

Um fato interessante a ser notado é que, quando comparado com todos estes métodos conhecidos, GENLIN sempre utiliza um número menor de avaliações de função. No caso de VE11 e MINOS, isso se deve ao fato de ambos utilizarem método do tipo quase-Newton, enquanto GENLIN utiliza Newton truncado. Já no caso de IPOPT, isso aparentemente se deve à maneira com que GENLIN lida com as restrições lineares.

Utilizamos os novos métodos para resolver o problema de recuperar constantes ópticas de filmes finos. Os resultados obtidos por BETRALIN e GENLIN foram muito bons. As constantes ópticas e espessuras recuperadas foram muito próximas dos valores verdadeiros. Para este problema, GENLIN levou menos tempo para resolver cada instância do que BETRALIN.

Agora que já possuímos uma implementação eficiente e competitiva para resolver pro-

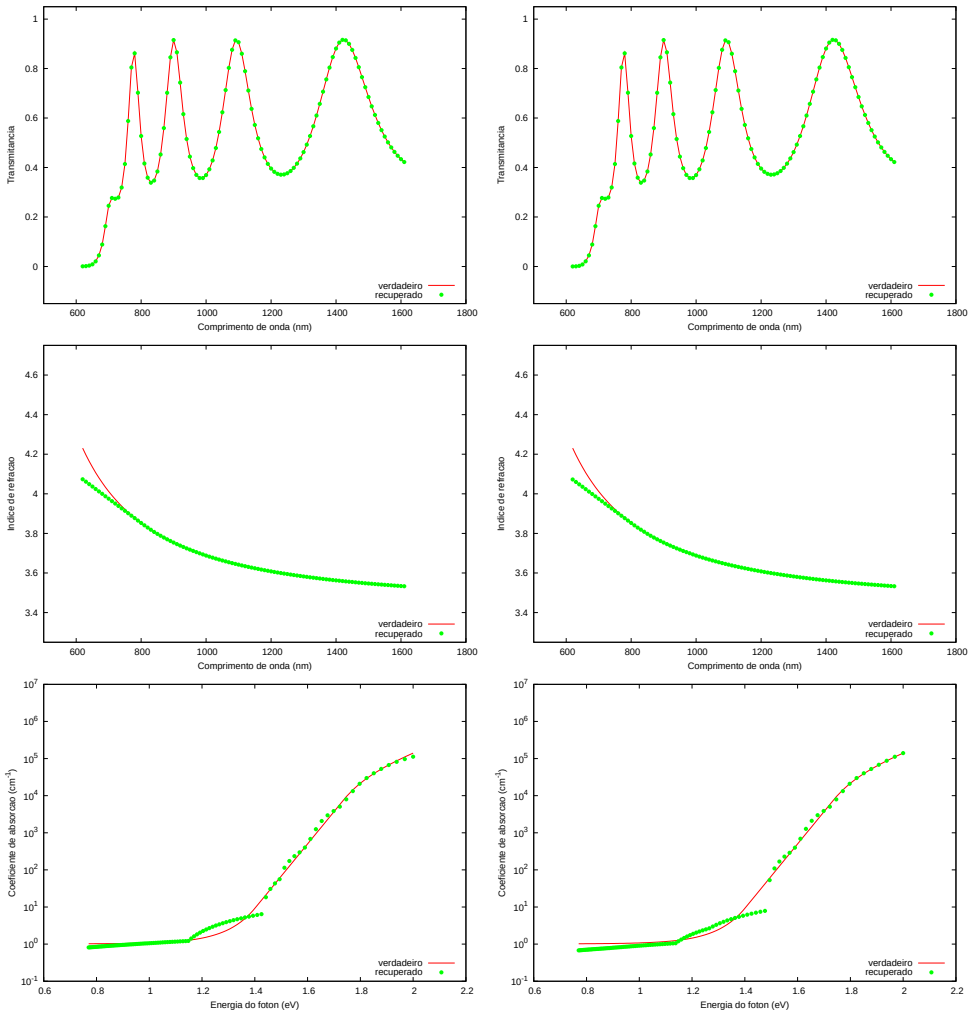


Figura 2.16: Valores reais e recuperados por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) para transmitância, índice de reflexão e coeficiente de absorção do Filme B.

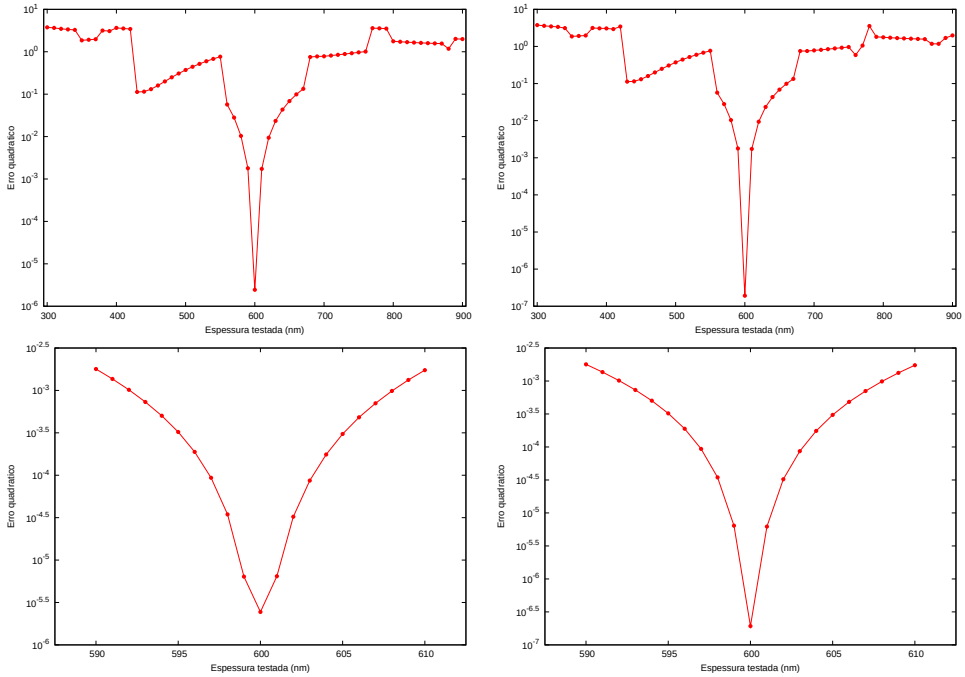


Figura 2.17: Erro quadrático obtido por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) durante o processo de otimização em relação aos valores testados para espessura do Filme B.

blemas com restrições lineares, podemos desenvolver uma implementação de ALGENCAN que utilize BETRALIN ou GENLIN para resolver os subproblemas que aparecem a cada iteração e, assim, penalizar somente as restrições não-lineares.

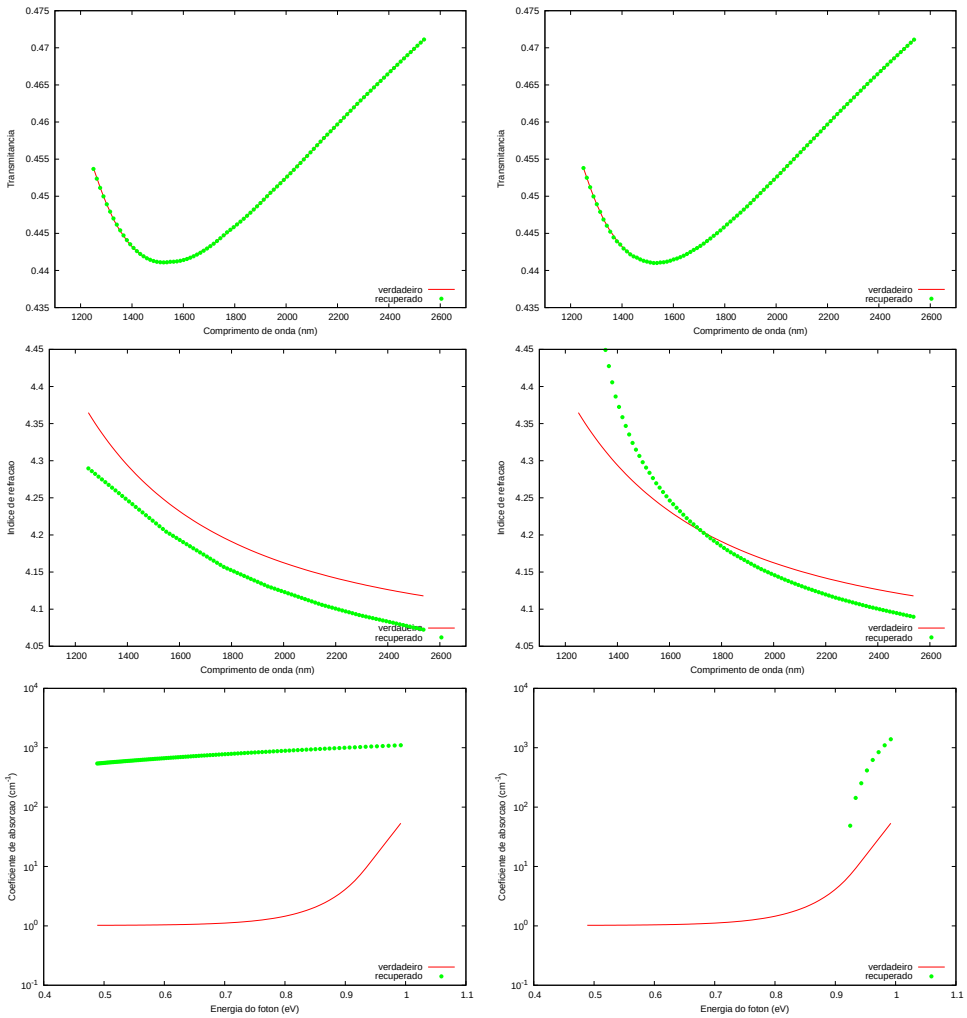


Figura 2.18: Valores reais e recuperados por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) para transmitância, índice de reflexão e coeficiente de absorção do Filme C.

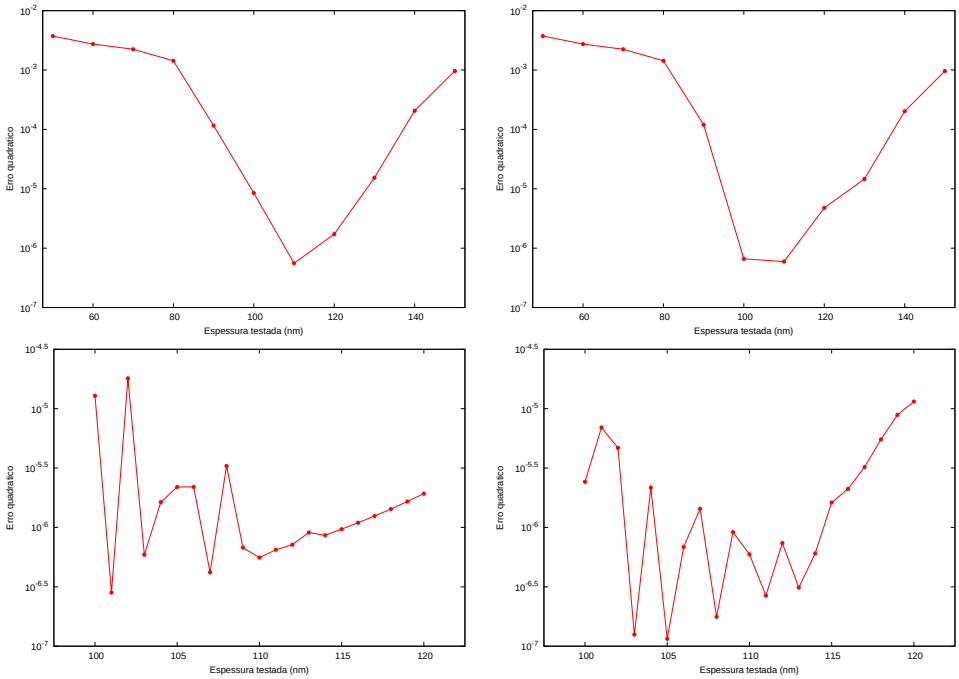


Figura 2.19: Erro quadrático obtido por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) durante o processo de otimização em relação aos valores testados para espessura do Filme C.

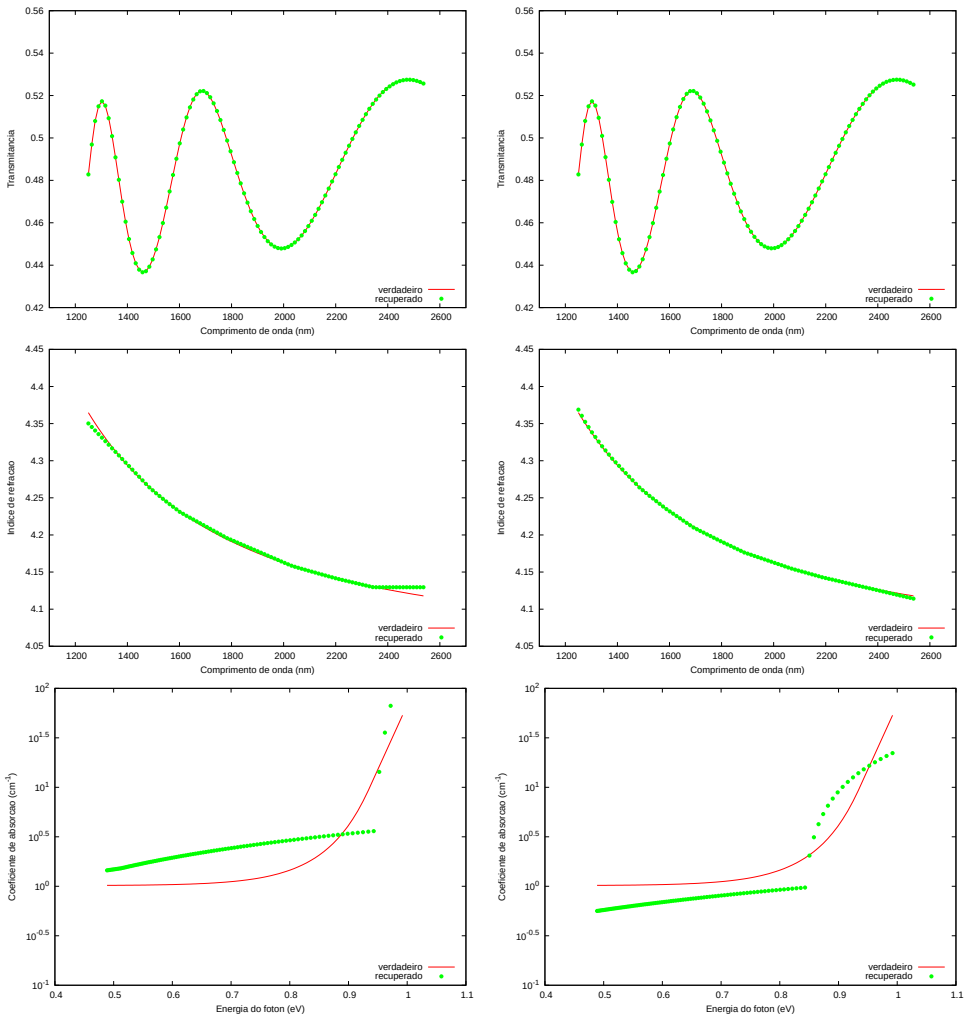


Figura 2.20: Valores reais e recuperados por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) para transmitância, índice de reflexão e coeficiente de absorção do Filme D.

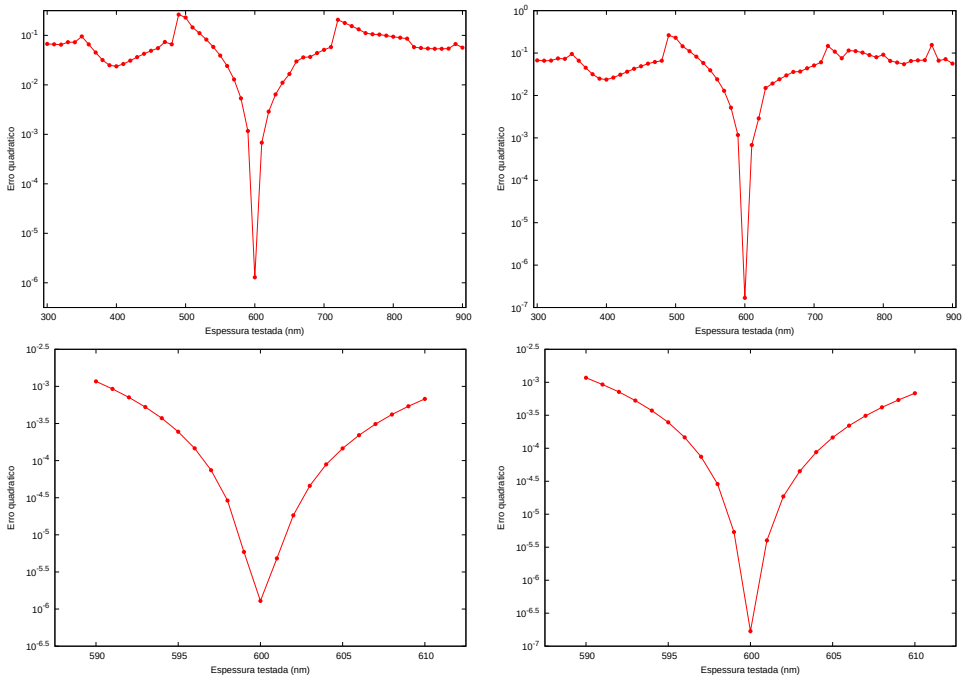


Figura 2.21: Erro quadrático obtido por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) durante o processo de otimização em relação aos valores testados para espessura do Filme D.

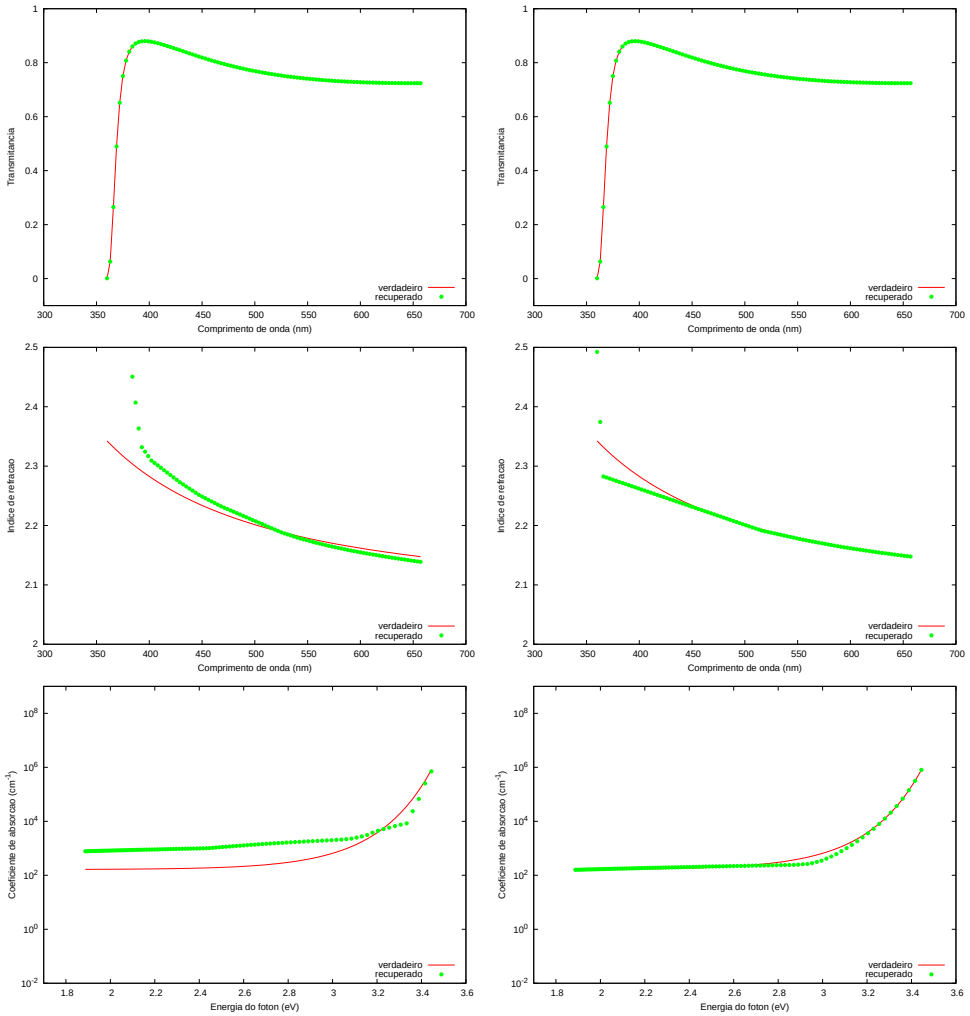


Figura 2.22: Valores reais e recuperados por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) para transmitância, índice de reflexão e coeficiente de absorção do Filme E.

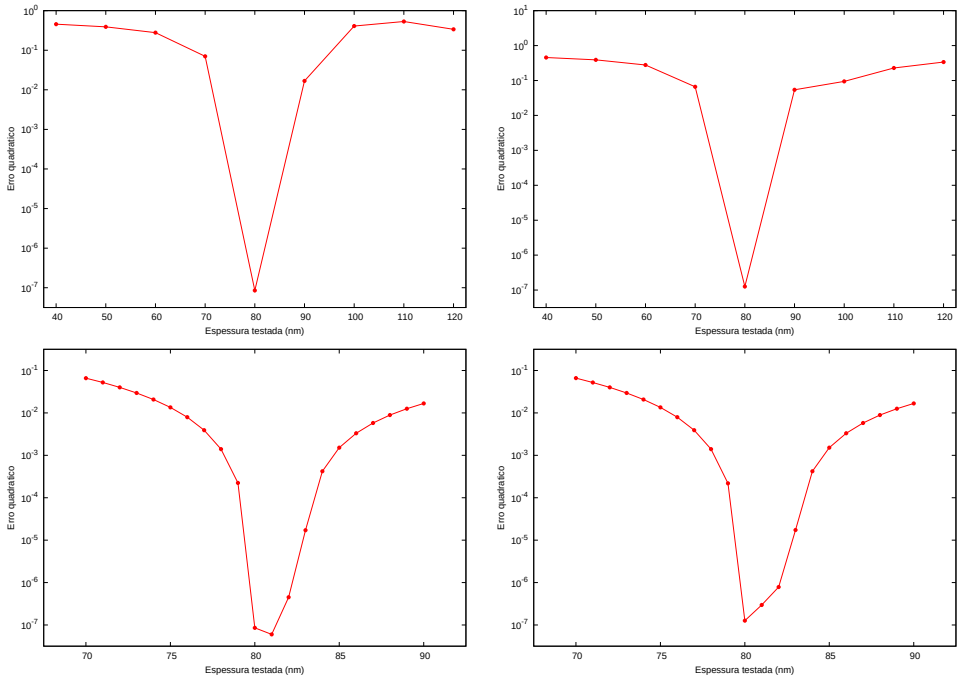


Figura 2.23: Erro quadrático obtido por BETRALIN (gráficos à esquerda) e GENLIN (gráficos à direita) durante o processo de otimização em relação aos valores testados para espessura do Filme E.

Capítulo 3

Lagrangiano aumentado com restrições lineares no nível inferior

Estamos agora interessados em resolver o seguinte problema:

$$\begin{array}{ll} \text{Minimizar} & f(x) \\ \text{sujeita a} & x \in \Omega_1 = \{x \in I\!R^n \mid h(x) = 0,\ g(x) \leq 0\} \text{ e} \\ & x \in \Omega_2 = \{x \in I\!R^n \mid Ax = b,\ Cx \leq d,\ \ell \leq x \leq u\}, \end{array} \tag{3.1}$$

onde $h : I\!R^n \to I\!R^{m_1}, g : I\!R^n \to I\!R^{p_1}, f : I\!R^n \to I\!R$ são suaves, $x \in I\!R^n$, $A \in I\!R^{m_2 \times n}$, $b \in I\!R^{m_2}$, $C \in I\!R^{p_2 \times n}$, $d \in I\!R^{p_2}$, $\ell \in I\!R^n$, $u \in I\!R^n$ e $\ell \leq u$.

Como vimos no Capítulo 1, podemos usar um método de Lagrangiano aumentado para resolver o problema (3.1) penalizando as restrições em Ω_1 e, a cada iteração, resolver um subproblema que trabalha explicitamente com as restrições em Ω_2, dado que tenhamos um método eficiente para resolver estes subproblemas.

No Capítulo 2, apresentamos um método eficiente para minimização com restrições lineares e de caixa. Implementamos, então, em Fortran 77, duas versões do método de Lagrangiano aumentado, baseadas na implementação de Algencan, que penalizam somente as restrições não-lineares. Chamamos de Albetralin o método de Lagrangiano aumentado que utiliza Betralin como subalgoritmo para resolver os subproblemas com restrições lineares e de caixa que aparecem a cada iteração. Chamamos Algenlin o método que utiliza Genlin para resolver os subproblemas.

Na implementação dos dois métodos, mantiveram-se todos os detalhes e parâmetros de Betralin e Genlin apresentados no Capítulo 2. A implementação do Lagrangiano aumentado é a mesma utilizada pelas versões de Algencan, apresentadas na Seção 1.2, trocando-se apenas o algoritmo interno para resolver o subproblema de cada iteração. Utilizamos os parâmetros padrão de Algencan referentes ao Lagrangiano aumentado.

Note que, como Albetralin e Algenlin utilizam Betralin e Genlin, respectivamente, para resolver os subproblemas, a teoria de convergência apresentada na Seção 2.5 garante que valem os Teoremas 1.1 e 1.2. Ou seja, temos provada a convergência de Albetralin e

Algenlin.

Na Seção 3.1 apresentamos os resultados numéricos da comparação entre os novos métodos usando problemas da coleção CUTEr [15] e comparamos seus desempenhos com Algencan-B, Algencan-G, Minos [41], Ipopt [54] e Lancelot B [18, 19, 20], conhecidos métodos para resolver (3.1). Na Seção 3.2 apresentamos as conclusões obtidas.

3.1 Resultados numéricos

De forma análoga a Algencan, os critérios de convergência de Albetralin e Algenlin são $\|g_P(x_k, \delta^k)\|_\infty \leq \varepsilon_1$ e $\max\{\|h(x_k)\|_\infty, \|\sigma_k\|_\infty\} \leq \varepsilon_2$, onde $g_P(x_k, \delta^k) = P_{\Omega_2(x_k,\delta^k)}(x_k - \nabla L_{\rho_k}(x_k, \lambda_k, \mu_k)) - x_k$ e $[\sigma(x, \mu, \rho)]_i = \max\{[g(x)]_i, \mu_i/\rho\}$. Utilizamos $\varepsilon_1 = \varepsilon_2 = 10^{-8}$. Os parâmetros de Betralin e Genlin são os mesmos descritos no capítulo 2.

Como na Seção 2.7.1, denotamos por n_{lim} o número de limitantes nas variáveis do problema. Consideramos que variáveis com limitantes inferior e superior contribuem com dois limitantes. Denotamos por m_2 o número de restrições lineares de igualdade, p_2 o número de restrições lineares de desigualdade, m_1 o número de restrições não-lineares de igualdade, p_1 o número de restrições não-lineares de desigualdade, e n o número de variáveis verdadeiras.

Selecionamos para nossos experimentos todos os problemas da coleção CUTEr [15] com, no máximo, 500 variáveis, com $1 \leq m_2 + p_2 \leq 2000$, e número ilimitado de restrições não-lineares, totalizando 84 problemas. As principais características destes problemas estão na Tabela 3.1. Alguns problemas da coleção CUTEr possuem variáveis nas quais os limitantes inferior e superior coincidem. Na Tabela 3.1, $n = n_1(n_2)$ significa que o problema possui n_1 variáveis verdadeiras e n_2 variáveis fixas. Neste caso, n_{lim} é o número de limitantes das variáveis verdadeiras.

3.1.1 Comparação de Albetralin e Algenlin com Algencan-B e Algencan-G

Estamos interessados em saber se, na presença de restrições lineares e não-lineares, vale a pena trabalhar explicitamente com as restrições lineares e penalizar as não-lineares (Albetralin e Algenlin) ou se é melhor trabalhar explicitamente apenas com as restrições de caixa e penalizar as demais restrições (Algencan-B e Algencan-G). Para tanto, vamos comparar o desempenho de Albetralin com Algencan-B e Algenlin com Algencan-G na resolução dos problemas da coleção CUTEr presentes na Tabela 3.1.

Em primeiro lugar, vamos comparar o desempenho de Albetralin e Algencan-B. Em todos os problemas, o iterando final de Albetralin satisfaz viabilidade das restrições lineares com tolerância 10^{-8}. No entanto, em 11 dos 84 problemas Albetralin não satisfaz viabilidade das restrições não-lineares com tolerância 10^{-8}. Nestes problemas, a explicação para o que acontece é a seguinte:

- nos problemas vanderm3, yorknet e net3, Albetralin pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

Problema	n	n_{lim}	m_2	p_2	m_1	p_1	Problema	n	n_{lim}	m_2	p_2	m_1	p_1
CUBENE	2 (0)	0	1	0	1	0	DEMBO7	16 (0)	32	0	1	0	19
RSNBRNE	2 (0)	0	1	0	1	0	RK23	17 (0)	6	4	0	7	0
SINVALNE	2 (0)	0	1	0	1	0	SYNTHES3	17 (0)	34	2	17	0	4
HS14	2 (0)	0	1	0	0	1	ERRINBAR	18 (0)	14	0	1	8	0
HIMMELP6	2 (0)	4	0	2	0	3	TENBARS1	18 (0)	14	0	1	8	0
HS23	2 (0)	4	0	1	0	4	TENBARS4	18 (0)	10	0	1	8	0
HS22	2 (0)	0	0	1	0	1	HIMMELBK	24 (0)	24	2	0	12	0
ZECEVIC4	2 (0)	4	0	1	0	1	MRIBASIS	24 (12)	48	1	43	8	3
HIMMELBE	3 (0)	0	2	0	1	0	LAUNCH	25 (0)	50	6	12	3	7
RECIPE	3 (0)	0	1	0	2	0	OPTCNTRL	29 (3)	30	10	0	10	0
BT4	3 (0)	0	1	0	1	0	OPTPRLOC	30 (0)	60	0	5	0	25
BT5	3 (0)	0	1	0	1	0	MESH	33 (8)	48	4	24	20	0
HS63	3 (0)	3	1	0	1	0	NET1	43 (5)	46	21	16	17	3
HS32	3 (0)	3	1	0	0	1	BATCH	48 (0)	94	12	60	0	1
CONGIGMZ	3 (0)	0	0	3	0	2	CHNRSBNE	50 (0)	0	49	0	49	0
MINMAXRB	3 (0)	0	0	2	0	2	PRODPL0	60 (0)	60	20	5	0	4
DEMYMALO	3 (0)	0	0	2	0	1	PRODPL1	60 (0)	60	20	5	0	4
GIGOMEZ1	3 (0)	0	0	2	0	1	CORE1	65 (0)	121	20	15	21	3
MAKELA1	3 (0)	0	0	1	0	1	SWOPF	83 (0)	20	43	0	35	14
MIFFLIN1	3 (0)	0	0	1	0	1	FEEDLOC	87 (3)	174	4	214	15	74
HS73	4 (0)	4	1	1	0	1	LAKES	90 (0)	18	60	0	18	0
HS42	4 (0)	0	1	0	1	0	GROUPING	100 (0)	200	25	0	100	0
FLETCHER	4 (0)	1	0	3	1	0	VANDERM1	100 (0)	0	0	99	100	0
HS74	4 (0)	8	0	2	3	0	VANDERM2	100 (0)	0	0	99	100	0
HS75	4 (0)	8	0	2	3	0	VANDERM3	100 (0)	0	0	99	100	0
BT11	5 (0)	0	1	0	2	0	VANDERM4	100 (0)	0	0	99	100	0
HS85	5 (0)	10	0	1	0	20	A4X12	115 (15)	0	1	192	0	192
LEWISPOL	6 (0)	12	3	0	6	0	NET2	133 (11)	136	64	32	59	5
TRIGGER	6 (1)	0	3	0	3	0	TRIMLOSS	142 (0)	264	20	51	0	4
SYNTHES1	6 (0)	12	0	4	0	2	LEAKNET	156 (0)	82	73	0	80	0
HEART8	8 (0)	0	2	0	6	0	CORE2	157 (0)	273	48	26	60	0
HS106	8 (0)	16	0	3	0	3	SSEBNLN	192 (2)	360	48	24	24	0
HS109	9 (0)	16	0	2	6	2	NGONE	197 (3)	198	0	98	0	4950
HS114	10 (0)	20	1	4	2	4	POLYGON	198 (2)	396	0	99	0	4950
HS113	10 (0)	0	0	3	0	5	BROWNALE	200 (0)	0	199	0	1	0
SYNTHES2	11 (0)	20	1	11	0	3	YORKNET	312 (0)	288	136	0	120	0
TRUSPYR2	11 (0)	8	0	8	3	0	C-RELOAD	342 (0)	600	26	0	174	84
TRUSPYR1	11 (0)	8	0	1	3	0	TWIRISM1	343 (0)	686	50	5	174	84
HS116	13 (0)	26	0	5	0	10	HAIFAL	343 (0)	0	0	18	0	8940
CONCON	15 (0)	5	7	0	4	0	HADAMARD	401 (0)	1	0	800	210	0
MCONCON	15 (0)	5	7	0	4	0	NET3	427 (37)	413	195	110	199	17
NYSTROM5	15 (3)	0	2	0	18	0	ARWDHNE	500 (0)	0	499	0	499	0

Tabela 3.1: Problemas com restrições lineares e não-lineares selecionados da coleção CUTEr.

- nos problemas FLETCHER, HS85, MESH, HADAMARD e ARWHDNE, ALBETRALIN pára declarando que há falta de progresso na dimuição da inviabilidade;

- nos problemas LEWISPOL e A4X12, ALBETRALIN pára declarando que o parâmetro de penalização se tornou muito grande;

- no problema VANDERM4, ALBETRALIN pára por não conseguir avaliar a função objetivo na projeção do ponto inicial.

Nos outros 73 problemas, o iterando final de ALBETRALIN é viável com tolerância 10^{-8}. Considerando estes 73 problemas, ALBETRALIN não pára satisfazendo seu critério de parada relativo a sucesso apenas no problema HS116, pois pára por atingir o número máximo de iterações.

O iterando final de ALGENCAN-B não satisfaz viabilidade com tolerância 10^{-8} em 9 dos 84 problemas. Nestes problemas, a explicação para o que acontece é a seguinte:

- nos problemas VANDERM4, A4X12, YORKNET, NET3 e ARWHDNE, ALGENCAN-B pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

- nos problemas LEWISPOL, HS106 e MESH, ALGENCAN-B pára declarando que o parâmetro de penalização se tornou muito grande;

- no problema HS75, ALGENCAN-B pára por atingir o número máximo de iterações.

Nos outros 75 problemas, o iterando final de ALGENCAN-B é viável com tolerância 10^{-8}. Em 67 destes 75 problemas, ALGENCAN-B pára satisfazendo seu critério de parada relativo a sucesso. Nos 8 problemas restantes, a explicação é a seguinte:

- no problema POLYGON, ALGENCAN-B pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);

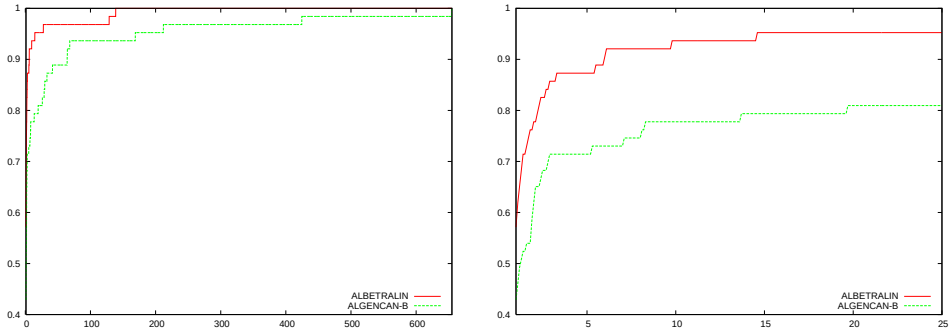
- nos problemas HS116, CONCON, MCONCON, BATCH, LAKES, TRIMLOSS e LEAKNET, ALGENCAN-B pára por atingir o número máximo de iterações permitido.

Considerando os 71 problemas nos quais ALBETRALIN e ALGENCAN-B obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final de ALGENCAN-B é equivalente ao obtido por ALBETRALIN. No entanto,

- em 4 problemas (HIMMELBK, GROUPING, TRIMLOSS, POLYGON), ALBETRALIN obteve valor de função menor do que ALGENCAN-B;

- em 4 problemas (HS116, NGONE, C-RELOAD, TWIRISM1), ALGENCAN-B obteve valor de função menor do que ALBETRALIN.

Nos 63 problemas restantes, tanto ALBETRALIN como ALGENCAN-B obtiveram pontos viáveis com valores de função equivalentes (utilizando o critério (1.13)). Considerando estes 63 problemas:



(a) (b)

Figura 3.1: Curva de perfil de desempenho comparando ALBETRALIN e ALGENCAN-B usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

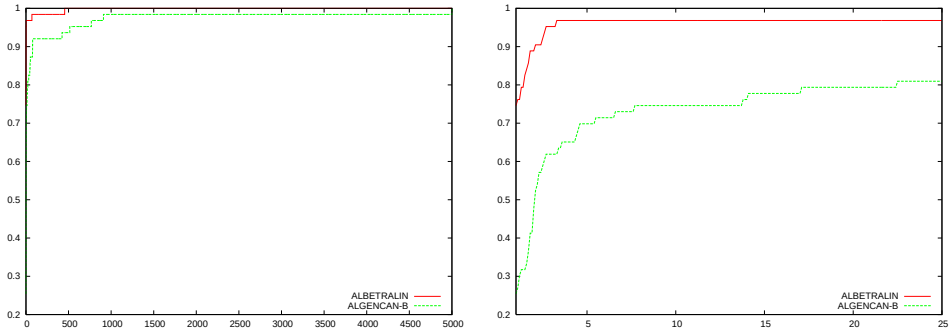
- ALBETRALIN realizou menos avaliações de função em 47 casos;
- ALGENCAN-B realizou menos avaliações de função em 16 casos;
- ALBETRALIN foi mais rápido que ALGENCAN-B (com tolerância de 10%) em 34 casos;
- ALGENCAN-B foi mais rápido que ALBETRALIN (com tolerância de 10%) em 24 casos;
- ALBETRALIN e ALGENCAN-B gastaram a mesma quantidade de tempo em 5 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (26 problemas), ALBETRALIN foi mais rápido que ALGENCAN-B 18 vezes e ALGENCAN-B foi mais rápido que ALBETRALIN 7 vezes (ambos os métodos gastaram a mesma quantidade de tempo em um caso).

Na Tabela 3.2 temos um resumo da comparação entre ALBETRALIN e ALGENCAN-B.

	ALBETRALIN	ALGENCAN-B
Problemas viáveis	73/84	75/84
Satisfez critérios de convergência	72/73	67/75
Valor de função melhor	4/71	4/71
Menos avaliações de função	47/63	16/63
Tempo menor (10% tolerância)	34/63	24/63

Tabela 3.2: Comparação entre ALBETRALIN e ALGENCAN-B para problemas selecionados da coleção CUTEr

Considerando os 63 problemas para os quais tanto ALBETRALIN como ALGENCAN-B chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho (veja Apêndice A). Na Figura 3.1, usamos o tempo de CPU como medida de desempenho. Na Figura 3.2, usamos o número de avaliações de função como medida de desempenho.



(a) (b)

Figura 3.2: Curva de perfil de desempenho comparando ALBETRALIN e ALGENCAN-B usando número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

Vamos agora comparar o desempenho de ALGENLIN e ALGENCAN-G. Em todos os problemas, o iterando final de ALGENLIN satisfaz viabilidade das restrições lineares com tolerância 10^{-8}. No entanto, em 10 dos 84 problemas ALGENLIN não satisfaz viabilidade das restrições não-lineares com tolerância 10^{-8}. Nestes problemas, a explicação para o que acontece é a seguinte:

- nos problemas A4X12, YORKNET e NET3, ALGENLIN pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas FLETCHER, HS85, MESH, HADAMARD e ARWHDNE, ALGENLIN pára declarando que há falta de progresso na diminuição da inviabilidade;
- no problema LEWISPOL, ALGENLIN pára declarando que o parâmetro de penalização se tornou muito grande;
- no problema VANDERM4, ALGENLIN pára por não conseguir avaliar a função objetivo na projeção do ponto inicial.

Nos outros 74 problemas, o iterando final de ALGENLIN é viável com tolerância 10^{-8}. Em 72 destes 74 problemas ALGENLIN pára satisfazendo seu critério de parada relativo a sucesso. Os outros dois problemas são HS116 (para o qual ALGENLIN pára por atingir o número máximo de iterações internas) e HAIFAL (para o qual ALGENLIN pára por atingir o tempo limite de 10 minutos imposto nos experimentos).

O iterando final de ALGENCAN-G não satisfaz viabilidade com tolerância 10^{-8} em 9 dos 84 problemas. Nestes problemas, a explicação para o que acontece é a seguinte:

- nos problemas A4X12, POLYGON e HADAMARD, ALGENCAN-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas LEWISPOL, YORKNET e ARWHDNE, ALGENCAN-G pára declarando que o parâmetro de penalização se tornou muito grande;

- nos problemas TRIGGER, MESH e VANDERM4, ALGENCAN-G pára declarando que há falta de progresso na diminuição da inviabilidade.

Nos outros 75 problemas, o iterando final de ALGENCAN-G é viável com tolerância 10^{-8}. Em 66 destes 75 problemas, ALGENCAN-G pára satisfazendo seu critério de parada relativo a sucesso. Nos 9 problemas restantes, a explicação é a seguinte:

- no problema NET3, ALGENCAN-G pára devido ao limite de tempo de CPU imposto na execução de cada par problema/método (10 minutos);
- nos problemas HS75, HS106, HS116, CONCON, MCONCON, BATCH, LAKES e LEAKNET, ALGENCAN-G pára por atingir o número máximo de iterações permitido.

Considerando os 72 problemas nos quais ALGENLIN e ALGENCAN-G obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final de ALGENCAN-G é equivalente ao obtido por ALGENLIN. No entanto,

- em 3 problemas (HS106, GROUPING, TWIRISM1), ALGENLIN obteve valor de função menor do que ALGENCAN-G;
- em 3 problemas (NGONE, C-RELOAD, HAIFAL), ALGENCAN-G obteve valor de função menor do que ALGENLIN.

Nos 66 problemas restantes, tanto ALGENLIN como ALGENCAN-G obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 66 problemas:

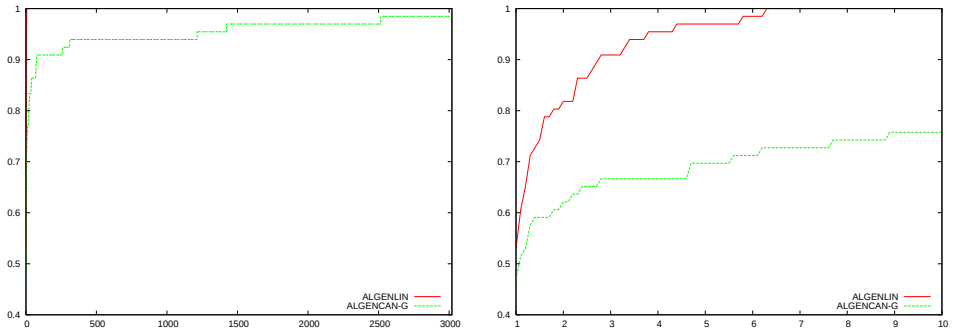
- ALGENLIN realizou menos avaliações de função em 52 casos;
- ALGENCAN-G realizou menos avaliações de função em 12 casos;
- ALGENLIN e ALGENCAN-G gastaram o mesmo número de avaliações de função em 2 casos;
- ALGENLIN foi mais rápido que ALGENCAN-G (com tolerância de 10%) em 32 casos;
- ALGENCAN-G foi mais rápido que ALGENLIN (com tolerância de 10%) em 26 casos;
- ALGENLIN e ALGENCAN-G gastaram a mesma quantidade de tempo em 8 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (24 problemas), ALGENLIN foi mais rápido que ALGENCAN-G 17 vezes e ALGENCAN-G foi mais rápido que ALGENLIN 6 vezes (ambos os métodos gastaram a mesma quantidade de tempo em um caso).

Na Tabela 3.3 temos um resumo da comparação entre ALGENLIN e ALGENCAN-G.

Considerando os 66 problemas para os quais tanto ALGENLIN como ALGENCAN-G chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 3.3, usamos o tempo de CPU como medida de desempenho. Na Figura 3.4, usamos o número de avaliações de função como medida de desempenho.

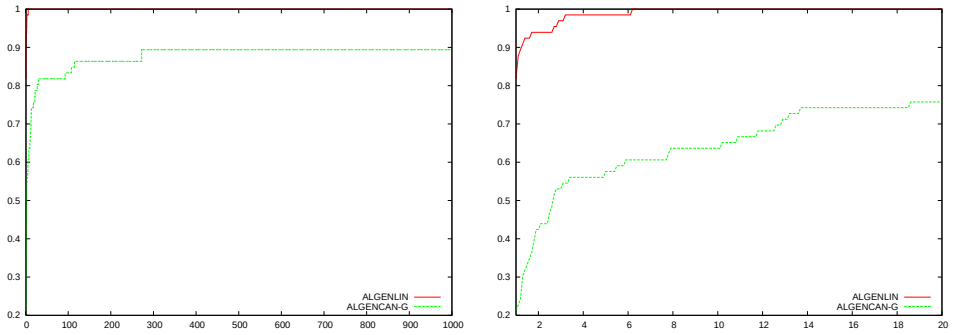
	Algenlin	Algencan-G
Problemas viáveis	74/84	75/84
Satisfez critérios de convergência	72/74	66/75
Valor de função melhor	3/72	3/72
Menos avaliações de função	52/66	12/66
Tempo menor (10% tolerância)	32/66	26/66

Tabela 3.3: Comparação entre Algenlin e Algencan-G para problemas selecionados da coleção Cuter



(a) (b)

Figura 3.3: Curva de perfil de desempenho comparando Algenlin e Algencan-G usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).



(a) (b)

Figura 3.4: Curva de perfil de desempenho comparando Algenlin e Algencan-G usando número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

Note que, apesar de ALBETRALIN resolver dois problemas a menos que ALGENCAN-B, ele gasta um pouco menos de tempo e muito menos avaliações de função para resolver os problemas. O mesmo acontece com ALGENLIN, que resolve um problema a menos que ALGENCAN-G, mas também gasta menos tempo e avaliações de função.

3.1.2 Comparação entre Albetralin e Algenlin

Como visto na Seção 3.1.1, tanto ALBETRALIN como ALGENLIN obtiveram pontos finais viáveis em 73 dos 84 problemas da Tabela 3.1. ALGENLIN encontrou ponto final viável também para o problema VANDERM3, para o qual ALBETRALIN pára em um ponto inviável. Consideremos, então, os 73 problemas para os quais ambos os métodos encontraram ponto final viável. Nestes 73 problemas, ALBETRALIN pára satisfazendo seu critério de parada relativo a sucesso em 72 problemas, enquanto ALGENLIN satisfaz seu critério de sucesso em 71 problemas. No problema HS116, tanto ALBETRALIN como ALGENLIN param por atingir o número máximo de iterações. No problema HAIFAL, ALGENLIN pára por atingir o tempo limite de 10 minutos imposto nos experimentos.

ALBETRALIN e ALGENLIN encontram valores de função equivalentes em 68 dos 73 problemas. ALBETRALIN encontra valor de função melhor para o problema HAIFAL. ALGENLIN encontra valor de função melhor para os problemas HS116, POLYGON, C-RELOAD e TWIRISM1. Considerando os 68 problemas para os quais ambos os métodos encontraram pontos finais viáveis e com valor de função equivalentes:

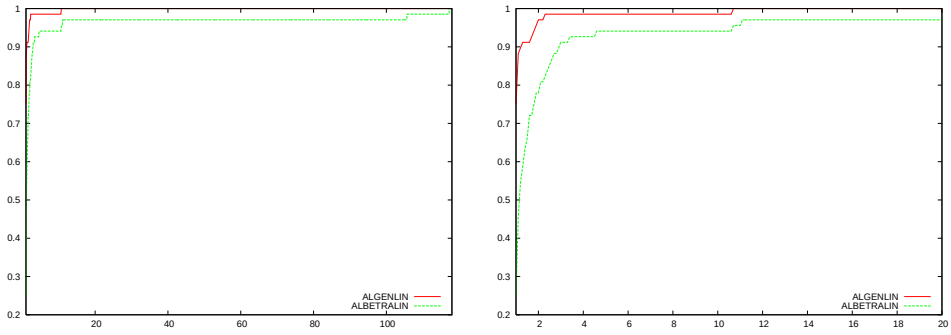
- ALBETRALIN realizou menos avaliações de função em 31 casos;
- ALGENLIN realizou menos avaliações de função em 27 casos;
- ALBETRALIN e ALGENLIN realizaram o mesmo número de avaliações de função em 10 casos;
- ALBETRALIN foi mais rápido que ALGENLIN (com tolerância de 10%) em 8 casos;
- ALGENLIN foi mais rápido que ALBETRALIN (com tolerância de 10%) em 37 casos;
- ALBETRALIN e ALGENLIN gastaram a mesma quantidade de tempo em 23 casos.
- Restringindo os três itens acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (23 problemas), ALBETRALIN foi mais rápido que ALGENLIN 3 vezes e ALGENLIN foi mais rápido que ALBETRALIN 16 vezes (ambos os métodos gastaram a mesma quantidade de tempo em 4 casos).

Na Tabela 3.4 temos um resumo da comparação entre ALBETRALIN e ALGENLIN.

Considerando os 68 problemas para os quais tanto ALBETRALIN como ALGENLIN chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 3.5, usamos o tempo de CPU como medida de desempenho. Na Figura 3.6, usamos o número de avaliações de função como medida de desempenho.

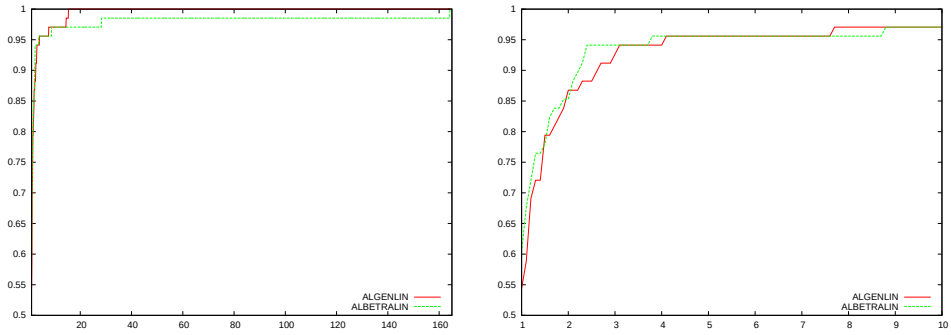
	Albetralin	Algenlin
Problemas viáveis	73/84	74/84
Satisfez critérios de convergência	72/73	72/74
Valor de função melhor	1/73	4/73
Menos avaliações de função	31/68	27/68
Tempo menor (10% tolerância)	8/68	37/68

Tabela 3.4: Comparação entre Albetralin e Algenlin para problemas selecionados da coleção Cuter



(a) (b)

Figura 3.5: Curva de perfil de desempenho comparando Albetralin e Algenlin usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).



(a) (b)

Figura 3.6: Curva de perfil de desempenho comparando Albetralin e Algenlin usando número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

Como aconteceu na comparação entre BETRALIN e GENLIN, podemos ver que há muitos casos de empate no número de avaliações de função usados por cada método. Já no caso do tempo gasto por cada método, ALGENLIN mostra resultados superiores a ALBETRALIN para este conjunto de problemas. Por isso, nas comparações a seguir, feitas com outros métodos conhecidos para resolver problemas do tipo (3.1), utilizamos apenas ALGENLIN.

3.1.3 Comparação de Algenlin com Minos

Nesta seção vamos comparar o desempenho de ALGENLIN e MINOS [41] na resolução dos problemas da coleção CUTEr presentes na Tabela 3.1. Para MINOS, usamos os parâmetros padrão, mudando apenas as tolerâncias para viabilidade e otimalidade[1] para 10^{-8}. O tempo medido foi o utilizado para resolver cada problema uma única vez. É importante observar que, nos casos de problemas com função objetivo linear ou constante, MINOS não conta avaliações de funções.

O iterando final de MINOS não satisfaz viabilidade com tolerância 10^{-8} em 16 dos 84 problemas. Nestes 16 problemas a explicação é a seguinte:

- nos problemas LEWISPOL, NYSTROM5, VANDERM1, VANDERM2, VANDERM3, C-RELOAD e TWIRISM1, MINOS pára declarando que o problema é inviável;
- nos problemas MESH e VANDERM4, MINOS pára declarando que o problema é ilimitado ou mal-escalado;
- nos problemas A4X12 e ARWHDNE, MINOS pára por atingir o número máximo de iterações;
- no problema RECIPE, MINOS pára por não haver mudança no ponto;
- nos problemas FLETCHER e HADAMARD, MINOS pára declarando que o ponto não pode ser melhorado;
- no problema TENBARS1, MINOS pára porque a base permaneceu singular depois de várias tentativas;
- no problema FEEDLOC, MINOS pára por erro no sistema.

Nos 68 problemas restantes, o iterando final de MINOS é viável com tolerância 10^{-8}. Em todos estes problemas, MINOS pára satisfazendo seu critério de parada relativo a sucesso.

Considerando os 65 problemas nos quais tanto ALGENLIN como MINOS obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função no ponto final obtido por ambos os métodos é equivalente. No entanto,

- em dois problemas (GROUPING, POLYGON), ALGENLIN obteve valor de função menor do que MINOS;

[1] Acrescentando as linhas `Feasibility Tolerance 1.0d-08`, `Optimality Tolerance 1.0d-08` e `Row Tolerance 1.0d-08` ao arquivo MINOS.SPC.

- em dois problemas (HS116, HAIFAL), MINOS obteve valor de função menor do que ALGENLIN.

Nos 61 problemas restantes, tanto ALGENLIN como MINOS obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 61 problemas:

- ALGENLIN realizou menos avaliações de função em 11 casos;
- MINOS realizou menos avaliações de função em 48 casos;
- ALGENLIN e MINOS realizaram o mesmo número de avaliações de função em 2 casos.
- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo (25 problemas), ALGENLIN foi mais rápido que MINOS (com tolerância de 10%) 3 vezes e MINOS foi mais rápido que ALGENLIN (com tolerância de 10%) 22 vezes.
- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (14 problemas), ALGENLIN foi mais rápido que MINOS 1 vez e MINOS foi mais rápido que ALGENLIN 13 vezes.

Na Tabela 3.5 temos um resumo da comparação entre ALGENLIN e MINOS. Na comparação do tempo gasto pelos métodos para resolver os problemas, consideramos apenas os problemas para os quais algum dos métodos gastou pelo menos 0.01 segundo.

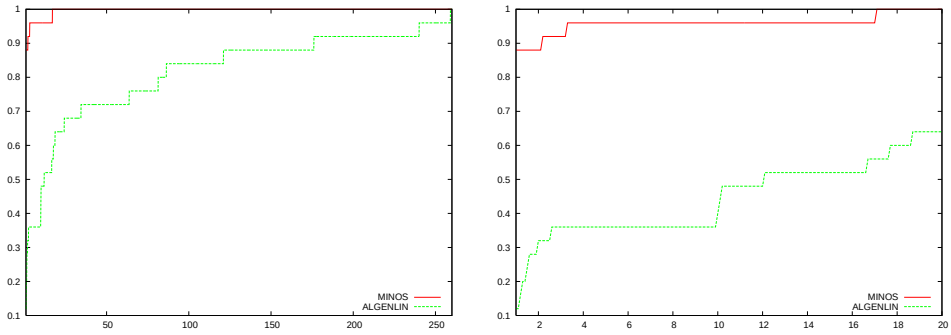
	ALGENLIN	MINOS
Problemas viáveis	74/84	68/84
Satisfez critérios de convergência	72/74	68/68
Valor de função melhor	2/65	2/65
Menos avaliações de função	11/61	48/61
Tempo menor (10% tolerância)	3/25	22/25

Tabela 3.5: Comparação entre ALGENLIN e MINOS para problemas selecionados da coleção CUTEr

Considerando os 61 problemas para os quais tanto ALGENLIN como MINOS chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 3.7, usamos o tempo de CPU como medida de desempenho, eliminando os problemas para os quais ambos os métodos gastaram menos de 0.01 segundo. Na Figura 3.8, usamos o número de avaliações de função como medida de desempenho. Note que ALGENLIN resolve mais problemas do que MINOS, porém é menos eficiente.

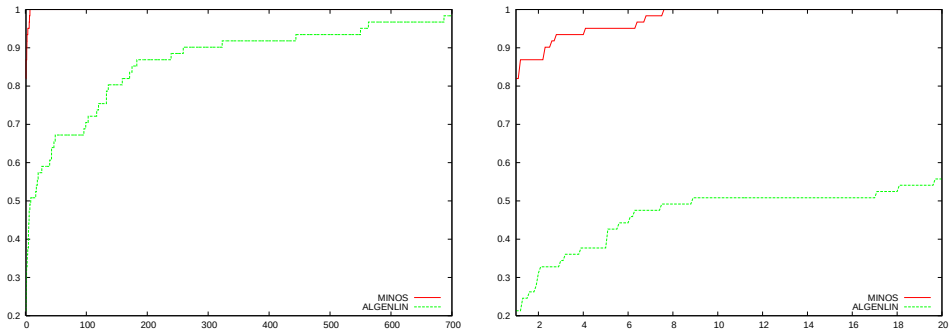
3.1.4 Comparação de Algenlin com Ipopt

Nesta seção vamos comparar o desempenho de ALGENLIN e IPOPT [54] na resolução dos problemas da coleção CUTEr presentes na Tabela 3.1. Utilizamos os parâmetros padrão para IPOPT (versão 3.3.2). O tempo medido foi o utilizado para resolver cada problema uma única vez.



(a) (b)

Figura 3.7: Curva de perfil de desempenho comparando ALGENLIN e MINOS usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).



(a) (b)

Figura 3.8: Curva de perfil de desempenho comparando ALGENLIN e MINOS usando número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

Um ponto a ser observado é que, sob hipóteses razoáveis, métodos de pontos interiores Newtonianos (como IPOPT) convergem quadraticamente (ou, ao menos, superlinearmente), enquanto métodos práticos do tipo Lagrangiano aumentado (como ALGENCAN) geralmente convergem linearmente. Portanto, se ambos os métodos convergem a um mesmo ponto e a precisão pedida é restrita o suficiente, um método de pontos interiores Newtoniano precisará de menos tempo computacional do que um método do tipo Lagrangiano aumentado, independentemente do custo de cada iteração. Como estamos usando precisão 10^{-8}, podemos esperar um melhor desempenho de IPOPT.

O iterando final de IPOPT não satisfaz viabilidade com tolerância 10^{-8} em 6 dos 84 problemas. A explicação para cada problema é a seguinte:

- no problema MESH, IPOPT pára declarando que o problema pode ser ilimitado;
- nos problemas VANDERM1, VANDERM2 e VANDERM3, IPOPT pára declarando que o problema foi resolvido até um nível aceitável;
- no problema VANDERM4, IPOPT pára declarando que a restauração falhou;
- no problema TRIGGER, IPOPT pára declarando sucesso.

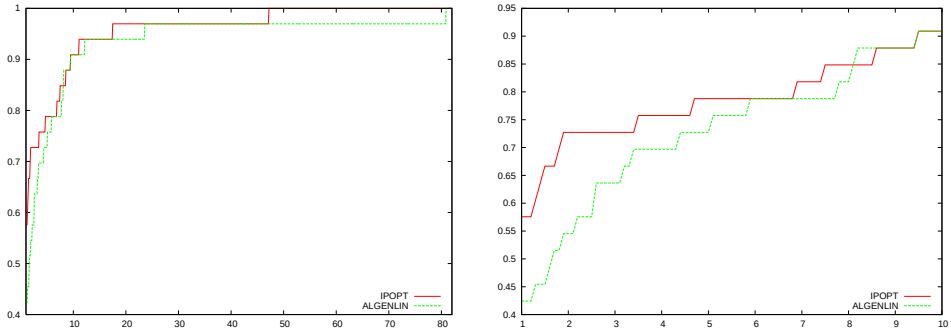
Em 5 problemas (LEWISPOL, NYSTROM5, CHNRSBNE, GROUPING, ARWHDNE) IPOPT pára no ponto inicial declarando que o problema possui pouco grau de liberdade, sem informar nenhum dado sobre o problema (valor das restrições ou da função objetivo). Nos outros 73 problemas, o iterando final de IPOPT é viável com tolerância 10^{-8}. Em todos estes problemas, IPOPT pára satisfazendo seu critério de parada relativo a sucesso.

Considerando os 67 problemas nos quais ALGENLIN e IPOPT obtiveram ponto final viável, observamos que, na maior parte dos casos, o valor de função de ambos os métodos no ponto final são equivalentes. No entanto,

- em 4 problemas (BT4, LEAKNET, POLYGON, TWIRISM1), ALGENLIN obteve valor de função menor do que IPOPT;
- em 5 problemas (SYNTHES2, HS116, NGONE, C-RELOAD, HAIFAL), IPOPT obteve valor de função menor do que ALGENLIN.

Nos 58 problemas restantes, tanto ALGENLIN como IPOPT obtiveram pontos viáveis com valores de função equivalentes. Considerando estes 58 problemas:

- ALGENLIN realizou menos avaliações de função em 8 casos;
- IPOPT realizou menos avaliações de função em 50 casos.
- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo (33 problemas), ALGENLIN foi mais rápido que IPOPT (com tolerância de 10%) 14 vezes e IPOPT foi mais rápido que ALGENLIN (com tolerância de 10%) 19 vezes.
- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (10 problemas), ALGENLIN foi mais rápido que IPOPT 1 vez e IPOPT foi mais rápido que ALGENLIN 9 vezes.



(a) (b)

Figura 3.9: Curva de perfil de desempenho comparando ALGENLIN e IPOPT usando tempo como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

Na Tabela 3.6 temos um resumo da comparação entre ALGENLIN e IPOPT. Consideramos como inviáveis os 5 problemas para os quais IPOPT pára no ponto inicial sem dar informação alguma sobre o problema. Na comparação do tempo gasto pelos métodos para resolver os problemas, consideramos apenas os problemas para os quais algum dos métodos gastou pelo menos 0.01 segundo.

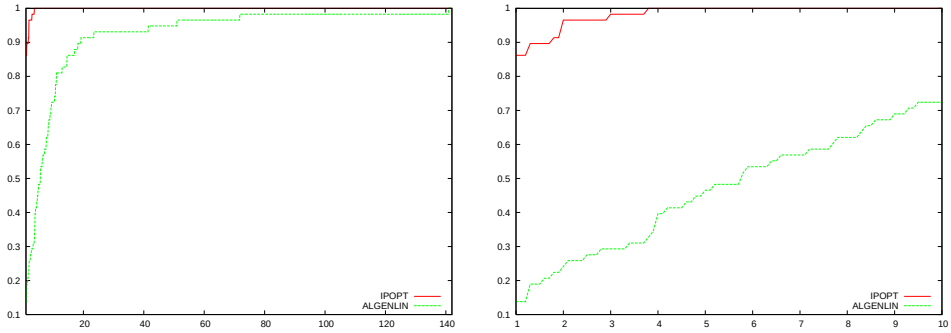
	ALGENLIN	IPOPT
Problemas viáveis	74/84	73/84
Satisfez critérios de convergência	72/74	73/73
Valor de função melhor	4/67	5/67
Menos avaliações de função	8/58	50/58
Tempo menor (10% tolerância)	14/33	19/33

Tabela 3.6: Comparação entre ALGENLIN e IPOPT para problemas selecionados da coleção CUTEr

Considerando os 58 problemas para os quais tanto ALGENLIN como IPOPT chegaram a um ponto viável com mesmo valor de função, construímos dois gráficos de perfil de desempenho. Na Figura 3.9, usamos o tempo de CPU como medida de desempenho, eliminando os problemas para os quais ambos os métodos gastaram menos de 0.01 segundo. Na Figura 3.10, usamos o número de avaliações de função como medida de desempenho. Veja que ALGENLIN resolve mais problemas do que IPOPT, mas IPOPT obtém melhores valores de função mais vezes do que ALGENLIN. Em termos de eficiência, IPOPT também apresenta melhores resultados. Como mencionado anteriormente, este comportamento já era esperado.

3.1.5 Comparação de Algenlin com Lancelot B

Nesta seção vamos comparar o desempenho de ALGENLIN e LANCELOT B [18, 19, 20] na resolução dos problemas da coleção CUTEr presentes na Tabela 3.1. Usamos os parâmetros



(a) (b)

Figura 3.10: Curva de perfil de desempenho comparando ALGENLIN e IPOPT usando número de avaliações de função como medida de desempenho. A Figura (b) destaca a região mais à esquerda da Figura (a).

padrão de LANCELOT B, mudando apenas as tolerâncias para viabilidade e otimalidade[2] para 10^{-8}. O tempo medido foi o utilizado para resolver cada problema uma única vez.

Como LANCELOT B transforma problemas sem função objetivo em problemas de minimização irrestrita, eliminamos os 15 problemas da Tabela 3.1 com esta característica. São eles: CUBENE, RSNBRNE, SINVALNE, HIMMELBE, RECIPE, TRIGGER, HEART8, NYSTROM5, CHNRSBNE, VANDERM1, VANDERM2, VANDERM3, VANDERM4, BROWNALE e ARWHDNE.

O iterando final de LANCELOT B não satisfaz viabilidade com tolerância 10^{-8} em 19 dos 69 problemas restantes. Nestes problemas, a explicação para o que acontece com o método é a seguinte:

- nos problemas LEWISPOL e GROUPING, LANCELOT B pára declarando que o problema parece inviável;
- nos problemas HS106, HS109, TRUSPYR2, HS116, CONCON, DEMBO7, ERRINBAR, TENBARS4, LAUNCH, MESH, FEEDLOC, LAKES, A4X12, LEAKNET, NGONE, YORKNET e NET3, LANCELOT B pára por atingir o número máximo de iterações permitido.

LANCELOT B pára a resolução do problema HAIFAL por atingir o tempo limite de CPU (10 minutos) imposto nos experimentos. Como, neste caso, nenhuma informação sobre o valor da função e das restrições no ponto final, este problema também foi eliminado das comparações com ALGENLIN.

Nos outros 49 problemas, o iterando final de LANCELOT B é viável com tolerância 10^{-8}. Em 33 destes 49 problemas, LANCELOT B pára satisfazendo seu critério de parada relativo a sucesso. Nos 16 problemas restantes (HS74, HS75, HS85, HS114, TRUSPYR1, MCONCON, TENBARS1, MRIBASIS, NET1, BATCH, PRODPL0, PRODPL1, CORE1, NET2, SSEBNLN, C-RELOAD), LANCELOT B pára declarando que o passo é muito pequeno e não gera mudança na função objetivo.

[2] Acrescentamos as linhas `primal-accuracy-required 1.0d-08` e `dual-accuracy-required 1.0d-08` ao arquivo RUNLANB.SPC.

Considerando os 46 problemas nos quais ALGENLIN e LANCELOT B obtiveram ponto final viável, observamos que, na maior parte dos casos, os valores de função no ponto final de ambos os métodos são equivalentes. No entanto, em 4 problemas (TENBARS1, POLYGON, C-RELOAD, TWIRISM1), LANCELOT B obteve valor de função menor que ALGENLIN.

Nos 42 problemas restantes, tanto ALGENLIN como LANCELOT B obtiveram pontos viáveis com valores de função equivalentes. Como não dispomos do número de avaliações de função utilizados por LANCELOT B, analisaremos apenas o tempo gasto. Considerando estes 42 problemas:

- Considerando os problemas nos quais um dos métodos gastou pelo menos 0.01 segundo (23 problemas), ALGENLIN foi mais rápido que LANCELOT B (com tolerância de 10%) 16 vezes e LANCELOT B foi mais rápido que ALGENLIN (com tolerância de 10%) 7 vezes.

- Restringindo o item acima a problemas nos quais um dos métodos gastou pelo menos 0.1 segundo (11 problemas), ALGENLIN foi mais rápido que LANCELOT B 7 vezes e LANCELOT B foi mais rápido que ALGENLIN 4 vezes.

Na Tabela 3.7 temos um resumo da comparação entre ALGENLIN e LANCELOT B. Na comparação do tempo gasto pelos métodos para resolver os problemas, consideramos apenas os problemas para os quais algum dos métodos gastou pelo menos 0.01 segundo.

	ALGENLIN	LANCELOT B
Problemas viáveis	61/69	50/69
Satisfez critérios de convergência	59/61	33/49
Valor de função melhor	0/46	4/46
Tempo menor (10% tolerância)	16/23	7/23

Tabela 3.7: Comparação entre ALGENLIN e LANCELOT B para problemas selecionados da coleção CUTEr

Considerando os 42 problemas para os quais tanto ALGENLIN como LANCELOT B chegaram a um ponto viável com mesmo valor de função, construímos um gráfico de perfil de desempenho. Usamos o tempo de CPU como medida de desempenho, eliminando os problemas para os quais ambos os métodos gastaram menos de 0.01 segundo (veja Figura 3.11). Note que ALGENLIN é mais eficiente e mais robusto do que LANCELOT B.

3.2 Conclusões

Implementamos, em Fortran 77, ALBETRALIN e ALGENLIN, dois métodos para resolver o problema (3.1) e os testamos nos 84 problemas da coleção CUTEr com até 500 variáveis, de 1 a 2000 restrições lineares e número ilimitado de restrições não-lineares.

Comparamos o desempenho de ALBETRALIN com ALGENCAN-B. O novo método, que trabalha explicitamente com as restrições lineares e de caixa e penaliza as restrições não-lineares, resolveu dois problemas a menos. No entanto, mostrou-se um pouco mais rápido e realizou

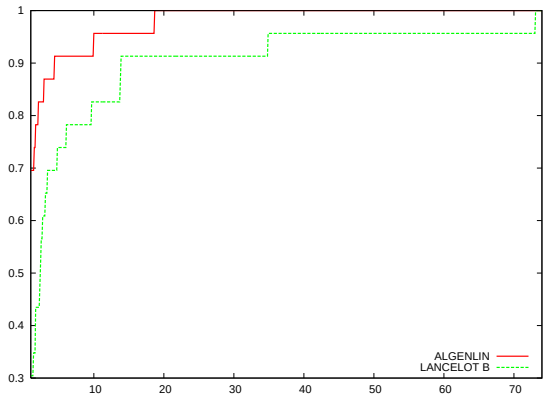


Figura 3.11: Curva de perfil de desempenho comparando ALGENLIN e LANCELOT B usando tempo como medida de desempenho.

muito menos avaliações de função do que ALGENCAN-B. O mesmo acontece quando comparamos ALGENLIN com ALGENCAN-G. ALGENLIN resolveu um problema a menos, mas foi, também, um pouco mais rápido e realizou menos avaliações de função do que ALGENCAN-G. Esperamos que, desenvolvendo uma versão esparsa de GENLIN para ser usado em ALGENLIN, ou pelo menos usar a projeção esparsa para GENLIN e BETRALIN, melhore os resultados obtidos pelos novos métodos.

Comparamos os desempenhos de ALBETRALIN e ALGENLIN. Neste conjunto de problemas, ALGENLIN se mostrou mais robusto e um pouco mais eficiente do que ALBETRALIN, como aconteceu na comparação entre BETRALIN e GENLIN.

Comparamos também o desempenho de ALGENLIN com MINOS, IPOPT e LANCELOT B. Quando comparado a MINOS, temos que ALGENLIN resolve mais problemas, mas é mais lento e gasta maior número de avaliações de função. O mesmo acontece na comparação com IPOPT. No entanto, este último obtém melhores valores de função mais vezes do que ALGENLIN. Quando comparado com LANCELOT B, ALGENLIN resolve mais problemas e é mais rápido.

É importante observar que métodos de pontos interiores Newtonianos (como IPOPT) convergem quadraticamente (ou, ao menos, superlinearmente), enquanto métodos práticos do tipo Lagrangiano aumentado (como ALGENCAN) geralmente convergem linearmente. Como pedimos precisão 10^{-8}, podemos esperar um melhor desempenho de IPOPT.

Estes resultados nos levam a buscar melhoria na eficiência de ALGENLIN, o que pode ser obtido utilizando-se a técnica de aceleração, apresentada em [7]. Além disso, como já mencionado, uma versão esparsa de GENLIN pode trazer melhores resultados.

Conclusões e trabalhos futuros

Neste trabalho, introduzimos Betra-esparso, uma versão de Betra [3] para problemas de grande porte. Utilizamos Betra (denso ou esparso) na resolução dos subproblemas que surgem a cada iteração de Algencan (um método de Lagrangiano aumentado). Para decidir qual algoritmo deve ser utilizado para resolver cada subproblema, desenvolvemos regras que escolhem o subalgoritmo de acordo com as características do subproblema. Como esperado, os experimentos numéricos mostraram que usar Betra como subalgoritmo de Algencan é a melhor alternativa para subproblemas pequenos, Betra-esparso é a melhor alternativa para subproblemas de grande porte com decomposição esparsa da Hessiana e Gencan é a melhor alternativa para os outros tipos de subproblemas. Além disso, o uso da escolha automática do algoritmo interno apresentou resultados muito bons, algumas vezes ainda melhores do que a aplicação de um único algoritmo interno. Quando comparadas com Lancelot B para resolver problemas com restrições gerais, as versões de Algencan obtiveram resultados compatíveis na resolução de problemas de pequeno porte, porém há que se melhorar o desempenho de Algencan para problemas de grande porte.

Paralelamente ao desenvolvimento deste trabalho, foram feitos avanços em Algencan, como o uso da técnica de aceleração introduzida em [7]. Outros detalhes algorítmicos menores, porém de grande impacto, como escalamento da função objetivo e das restrições e adição de variáveis de folga (em casos nos quais isso se mostra adequado), foram incorporados à nova versão de Algencan. Experimentos numéricos mostram que esta nova versão de Algencan é mais robusta e eficiente do que Lancelot B e comparável com Ipopt. Note que estes resultados foram obtidos por uma versão que não utiliza Betra, Betra-esparso nem Betralin (ou Genlin). Acreditamos que, com a integração destes subalgoritmos à versão mais recente de Algencan, obteremos resultados mais do que satisfatórios quando comparados a Lancelot B, Ipopt ou outros métodos para resolução de problemas de programação não-linear.

Introduzimos, também, dois métodos de restrições ativas para resolver problemas com restrições lineares: Betralin e Genlin. Os experimentos numéricos mostraram que, como esperado, trabalhar explicitamente com restrições lineares é muito mais eficiente e eficaz do que penalizá-las (como é feito no método de Lagrangiano aumentado Algencan). Quando comparado a outros métodos conhecidos para resolver problemas com restrições lineares, os novos métodos se mostraram competitivos. A eficiência de Betralin (e Genlin) está baseada na idéia, desenvolvida neste trabalho, de projeções parciais nas restrições, adotada tanto no critério de convergência como no método de Gradiente Espectral Projetado Parcial.

Por fim, incorporamos Betralin e Genlin ao arcabouço de Lagrangianos aumentados. Embora ambos os métodos tenham se mostrado extremamente eficientes para resolver problemas apenas com restrições lineares, os novos métodos de Lagrangiano aumentado (Albetralin e Algenlin) que penalizam somente as restrições não-lineares não apresentaram grande ganho com relação a Algencan (que penaliza também as restrições lineares) quando usados para resolver problemas com restrições lineares e não-lineares. Analisando os problemas para os quais Albetralin (ou Algenlin) apresenta resultados piores do que Algencan, notamos que o número de iterações de Lagrangiano aumentado gasto por ambos os métodos é parecido, devido a alguma restrição não-linear que precisa de muitas iterações para ser satisfeita. Como cada iteração de minimização com restrições lineares é mais custosa do que uma iteração de minimização com restrições de caixa, Albetralin (ou Algenlin) leva mais tempo para resolver o problema. Isto explica, ao menos parcialmente, o motivo pelo qual o ganho obtido na resolução de problemas apenas com restrições lineares não se transfere de forma imediata ao método de Lagrangiano aumentado.

Algumas linhas para trabalhos futuros seguem:

1. Para os algoritmos "irrestritos" (usados como subalgoritmos dos métodos apresentados nos Capítulos 1 e 2), implementaremos uma mistura dos algoritmos "irrestritos" de Betra e Gencan. A idéia é utilizar uma direção tão boa como a calculada pelo algoritmo "irrestrito" com regiões de confiança, porém utilizando menos decomposições de matriz. Com isso, esperamos melhorar o desempenho tanto dos métodos de minimização em caixa como dos métodos para minimização com restrições lineares e, conseqüentemente, melhorar os métodos de Lagrangiano aumentado.

2. Os métodos para minimização com restrições lineares introduzidos são métodos densos. O fato das matrizes serem densas aparece na fatoração utilizada para caminhar no subespaço afim e no algoritmo de projeção. Isto limita a aplicabilidade dos métodos a problemas de pequeno e médio porte. Pretendemos estudar alternativas para desenvolver uma versão esparsa dos métodos introduzidos neste trabalho e, assim, estender a aplicabilidade destes métodos a problemas de grande porte.

Neste trabalho, exploramos os algoritmos desenvolvidos para minimização com restrições lineares no contexto de Lagrangianos aumentados. No entanto, algoritmos para minimização com restrições lineares aparecem como subalgoritmos de outras estratégias para resolver problemas de programação não-linear. Métodos de Restauração Inexata são uma alternativa interessante para resolver problemas gerais de programação não-linear (veja [33, 35, 36]). Estes métodos consistem em duas fases: na primeira se busca melhorar a viabilidade. Na segunda se busca melhorar a otimalidade, minimizando o Lagrangiano da função objetivo sujeito a uma linearização das restrições. Até agora só existem implementações de métodos do tipo Restauração Inexata para resolver problemas específicos. Pretendemos implementar um método utilizando essa técnica que seja capaz de resolver problemas gerais, usando, na segunda fase, os novos métodos desenvolvidos para resolver problemas com restrições lineares.

Apêndice A

Gráfico de perfil de desempenho

Depois de definido um conjunto de problemas aos quais serão aplicados diferentes métodos de programação não-linear, estamos interessados em comparar, basicamente, o número de avaliação de função ou o tempo gasto por cada método e a qualidade da solução encontrada por cada um deles.

Para visualizar melhor a comparação do desempenho dos métodos, utilizamos uma medida de desempenho (tipicamente o número de avaliações de função ou o tempo de CPU gasto pelos métodos para resolver os problemas) e construímos um gráfico do perfil de desempenho (*performance profile*, veja [22]). Neste gráfico, usando, por exempolo, o tempo como medida de desempenho, se um ponto da curva correspondente ao método M vale I no eixo das abscissas e J no eixo das ordenadas, significa que o método M resolve $(J \times 100)\%$ dos problemas em um tempo menor ou igual a I vezes o tempo mais rápido. Os pontos mais significativos do gráfico são o mais à esquerda e o mais à direita. À esquerda medimos a *eficiência* dos métodos (verificamos a porcentagem de problemas para os quais cada método é o mais rápido dentre todos os comparados) e à direita medimos a *robustez* (verificamos a porcentagem de problemas que cada método foi capaz de resolver).

Para a construção dos gráficos de perfil de desempenho leva-se em conta não somente o tempo de CPU gasto por cada método, mas também a solução encontrada por cada um deles. Se, para o conjunto de problemas usado, algum método fornece um ponto inviável como solução, consideramos que este método não encontrou uma solução. Quando todos os métodos encontram valores de função equivalentes, comparamos o tempo gasto por cada um para encontrá-la. Quando encontram soluções diferentes, consideramos que aquele que chegou a um ponto com valor de função maior não encontrou solução.

Um outro ponto importante a ser observado é a medição do tempo gasto pelos métodos. Quando o tempo gasto por um método para resolver um problema é muito próximo de zero, a medição do tempo pode conter um erro grande. Uma idéia para que isso seja evitado é, quando possível, executar o par problema/método várias vezes, com o mesmo ponto inicial, medindo o tempo antes e depois das várias execuções, até que o tempo total gasto por este par seja pelo menos um segundo. Neste caso, usamos a média de tempo das execuções na construção do gráfico.

Referências Bibliográficas

[1] 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, pp. 1286–1309, 2007.

[2] R. Andreani, E. G. Birgin, J. M. Martínez e M. L. Schuverdt. Augmented Lagrangian methods under the Constant Positive Linear Dependence constraint qualification. *Mathematical Programming* 111, pp. 5–32, 2008.

[3] M. Andretta, E. G. Birgin e J. M. Martínez. Practical active-set Euclidian trust-region method with spectral projected gradients for bound-constrained minimization. *Optimization* 54, pp. 305–325, 2005.

[4] E. G. Birgin, R. Castillo e J. M. Martínez, Numerical comparison of Augmented Lagrangian algorithms for nonconvex problems. *Computational Optimization and Applications* 31, pp. 31–56, 2005.

[5] E. G. Birgin, I. Chambouleyron e J. M. Martínez. Estimation of the optical constants and the thickness of thin films using unconstrained minimization. *Journal of Computational Physics* 151, pp. 862–880, 1999.

[6] E. G. Birgin, I. Chambouleyron e J. M. Martínez. Optimization problems in the estimation of parameters of thin films and the elimination of the influence of the substrate. *Journal of Computational and Applied Mathematics* 152, pp. 35–50, 2003.

[7] E. G. Birgin e J. M. Martínez. Improving ultimate convergence of an Augmented Lagrangian method. *Optimization Methods and Software* 23, pp. 177–195, 2008.

[8] E. G. Birgin e J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Computational Optimization and Applications* 23, pp. 101–125, 2002.

[9] E. G. Birgin, J. M. Martínez, W. F. Mascarenhas e D. P. Ronconi. Method of Sentinels for Packing Items within Arbitrary Convex Regions. *Journal of the Operational Research Society* 57, pp. 735–746, 2006.

[10] E. G. Birgin, J. M. Martínez, F. H. Nishihara e D. P. Ronconi. Orthogonal packing of rectangular items within arbitrary convex regions by nonlinear optimization. *Computers & Operations Research* 33, pp. 3535–3548, 2006.

[11] E. G. Birgin, J. M. Martínez e M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization* 10, pp. 1196–1211, 2000.

[12] E. G. Birgin, J. M. Martínez e M. Raydan. Algorithm 813: SPG - software for convex-constrained optimization. *ACM Transactions on Mathematical Software* 27, pp. 340–349, 2001.

[13] E. G. Birgin, J. M. Martínez e M. Raydan, Inexact Spectral Projected Gradient methods on convex sets. *IMA Journal on Numerical Analysis* 23, pp. 539–559, 2003.

[14] E. G. Birgin e F. N. C. Sobral. Minimizing the object dimensions in circle and sphere packing problems. *Computers & Operations Research* 35, pp. 2357–2375, 2008.

[15] I. Bongartz, A. R. Conn, N. I. M. Gould e Ph. L. Toint. CUTE: constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software* 21, pp. 123–160, 1995.

[16] I. Chambouleyron, J. M. Martínez, A. C. Moretti e C. Mulato. Retrieval of optical constants and thickness of thin films from transmition spectra. *Applied Optics* 36, n. 31, pp. 8238–8247, 1997.

[17] A. K. Cline, C. B. Moler, G. W. Stewart e J. H. Wilkinson. An estimate for the condition number of a matrix. *SIAM Journal on Numerical Analysis* 16, pp. 368–375, 1979.

[18] 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, pp. 545–572, 1991.

[19] A. R. Conn, N. I. M. Gould e Ph. L. Toint, *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*, Springer-Verlag, Berlin, 1992.

[20] A. R. Conn, N. I. M. Gould e Ph. L. Toint, *Trust-Region Methods*, MPS/SIAM series on Optimization, SIAM, Philadelphia, 2000.

[21] J. H. Conway e N. J. C. Sloane. Sphere Packings, Lattices and Groups. Springer-Verlag, New York, 1988.

[22] E. D. Dolan e J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming* 91, pp. 201–213, 2002.

[23] I. S. Duff, A. M. Erisman e J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1997.

[24] I. S. Duff e J. K. Reid. The Multifrontal Solution of Indefinite Sparse Symmetric Linear. *ACM Transactions on Mathematical Software (TOMS)* 9, n.3, pp.302–325, 1983.

[25] P. E. Gill, G. H. Golub, W. Murray e M. A. Saunders, Methods for modifying matrix factorizations. *Mathematics of Computation* 28, pp. 505–535, 1974.

[26] P. E. Gill, W. Murray e M. H. Wright, *Practical Optimization*, Academic Press, 1982.

[27] D. Goldfarb e A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming* 27 (1), pp. 1–33, 1983.

[28] G. H. Golub e C. F. van Loan, *Matrix Computations*, Johns Hopkins, 1996.

[29] R. Fletcher, *Practical Methods of Optimization*, Wiley, 1997.

[30] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications* 4, pp. 303–320, 1969.

[31] J. Horák. Constrained mountain pass algorithm for the numerical solution of semilinear elliptic problems. *Numerische Mathematik* 98, pp. 251–276, 2004.

[32] N. Krejic, J. M. Martínez, M. Mello e E. A. Pilotta. Validation of an Augmented Lagrangian algorithm with a Gauss-Newton Hessian approximation using a set of Hard-Spheres problems. *Computational Optimization and Applications* 16, pp. 247–263, 2000.

[33] J. M. Martínez. Inexact restoration method with Lagrangian tangent decrease and new merit function for nonlinear programming. *Journal of Optimization Theory and Applications* 111, pp. 39–58, 2001.

[34] M. Martínez e S. A. Santos, *Métodos Computacionais de Otimização*, Sociedade Brasileira de Matemática, 1995.

[35] J. M. Martínez e E. A. Pilotta. Inexact restoration methods for nonlinear programming: advances and perspectives. Em *Optimization and Control with applications*, editado por L. Q. Qi, K. L. Teo e X. Q. Yang. Springer, pp. 271–292, 2005.

[36] J. M. Martínez e E. A. Pilotta. Inexact restoration algorithms for constrained optimization. *Journal of Optimization Theory and Applications* 104, pp. 135–163, 2000.

[37] J. J. Moré e T. S. Munson. Computing mountain passes and transition states. *Mathematical Programming* 100, pp. 151–182, 2004.

[38] J. J. Moré e D. C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing* 4, pp. 553–572, 1983.

[39] C. Mulato, I. Chambouleyron, E. G. Birgin e J. M. Martínez. Determination of thickness and optical constants of amorphous silicon films from transmittance data. *Applied Physics Letters* 77, pp. 2133–2135, 2000.

[40] B. A. Murtagh e R. W. H. Sargent. A constrained minimization method with quadratic convergence. Em *Optimization*, editado por R. Fletcher. Academic Press, pp. 215–246, 1969.

[41] B. A. Murtagh e M. A. Saunders, Large-Scale linearly constrained optimization. *Mathematical Programming* 14, pp. 41–72, 1978.

[42] J. Nocedal e S. J. Wright, *Numerical Optimization*, Springer, 1999.

[43] M. J. D. Powell. A method for nonlinear constraints in minimization problems. Em *Optimization*, editado por R. Fletcher. Academic Press, New York, NY, pp. 283–298, 1969.

[44] M. J. D. Powell. ZQPCVX, a FORTRAN subroutine for convex programming. Report DAMTP/1983/NA17, University of Cambridge, England, 1983.

[45] M. J. D. Powell. On the quadratic-programming algorithm of Goldfarb and Idnani. *Mathematical Programming Study* 25, pp. 46–61, 1985.

[46] M. J. D. Powell. Log barrier methods for semi-infinite programming calculations. Em *Advances on Computer Mathematics and Its Applications*, editado por E. A. Lipitakis. World Scientific, pp. 1–2, 1993.

[47] M. J. D. Powell. A tolerant algorithm for linearly constrained optimization calculations. *Mathematical Programming B* 45, pp. 547–566, 1989.

[48] R. T. Rockafellar. The multiplier method of Hostenes and Powell applied to convex programming. *Journal of Optimization Theory and Applications* 12, pp. 555–562, 1973.

[49] R. W. Sargent e B. A. Murtagh. Projection methods for nonlinear programming. *Mathematical Programming* 4, pp. 245–268, 1973.

[50] K. Schittkowski, *QL: a Fortran code for convex quadratic programming*. User's Guide, version 2.1, 2004.

[51] R. Swanepoel. Determination of the thickness and optical constants of amorphous silicon. *Journal of Physics E* 16, pp. 1214–1222, 1983.

[52] R. Swanepoel. Determination of surface roughness and optical constants of inhomogeneous amorphous silicon films. *Journal of Physics E* 17, pp. 896–903, 1985.

[53] M. J. Todd e E. A. Yildirim. On Khachiyan's algorithm for the computation of minimum volume enclosing ellipsoids. *Discrete Applied Mathematics* 155, pp. 1731–1744, 2007.

[54] 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, pp. 25–57, 2006.

[55] Tango Project: Trustable Algorithms for Nonlinear General Optimization, disponível em `www.ime.usp.br/~egbirgin/tango`.

[56] La Cumparsita: the Tango Project collection of nonlinear programming test problems, disponível em `www.ime.usp.br/~egbirgin/collection`.