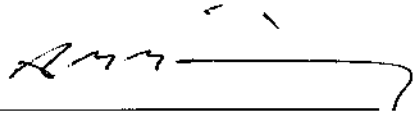
DIFERENCIAÇÃO COMPUTACIONAL E APLICAÇÕES

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Ernesto Julián Goldberg Birgin e aprovada pela comissão julgadora.

Campinas, 20 de agosto de 1998.



Prof. Dr. José Mario Martínez
Orientador

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica, UNICAMP, como requisito parcial para obtenção do Título de DOUTOR em Matemática Aplicada.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Birgin, Ernesto Julián Goldberg

B533d Diferenciação computacional e aplicações / Ernesto Julián Goldberg Birgin -- Campinas, [S.P. :s.n.], 1998.

Orientador : José Mario Martínez Perez

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Cálculo diferencial. 2. Programação não-linear. 3. Teoria do controle. 4. Filmes finos - Propriedades óticas . I. Martínez Perez, José Mario. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

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

Diferenciação Computacional e Aplicações*

Autor: ERNESTO JULIÁN GOLDBERG BIRGIN
Orientador: PROF. DR. JOSÉ MARIO MARTÍNEZ

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas, como parte dos pré-requisitos para obtenção do Título de Doutor em Matemática Aplicada.

Agosto de 1998

*Pesquisa financiada pela FAPESP, processo 95/2452-6.

Tese de Doutorado defendida e aprovada em 20 de agosto de 1998

Pela Banca Examinadora composta pelos Profs. Drs.

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

Prof (a). Dr (a). ANTONIO CARLOS MORETTI

Prof (a). Dr (a). JOSÉ LUIZ BOLDRINI

Prof (a). Dr (a). OLEG BURDAKOV

Prof (a). Dr (a). IVAN EMILIO CHAMBOULEYRON

Índice

Capítulo 1

Introdução

Consideremos o problema de

$$\text{Minimizar } f(x) = \prod_{i=1}^{n} x_i$$

e suponhamos que queremos aplicar um algoritmo que utilize informação de primeira ordem para gerar direções de descida. Existem pelo menos três formas para calcular o vetor gradiente

$$\nabla f = \begin{bmatrix} \prod_{i \neq 1} x_i \\ \prod_{i \neq 2} x_i \\ \vdots \\ \prod_{i \neq n} x_i \end{bmatrix}.$$

A primeira delas é calcular f e depois calcular $[\nabla f]_i = f(x)/x_i$, para $i = 1, \ldots, n$. Embora seja simples, se calculado desta forma, $[\nabla f]_i$ está indefinido para $x_i = 0$. Assim, descartaremos esta alternativa. Uma outra possibilidade é calcular $[\nabla f]_i = \prod_{j \neq i} x_j$. Esta alternativa está bem definida mas tem outro inconveniente. Calcular todas as derivadas parciais de f requer de $n(n-2)$ produtos enquanto o cálculo de f precisa de $n-1$. Desta forma, a complexidade de calcular o gradiente é $O(n)$ multiplicado pela complexidade de avaliar a função. Finalmente, a seguinte subrotina ilustra a terceira alternativa para o cálculo de f e ∇f.

Subrotina Produto:

```
esquerda[1]= 1
Para i= 2, ..., n
      Fazer esquerda[i] ← esquerda[i-1] * x_{i-1}.
fimpara
f= esquerda[n] * x_n
direita[n]= 1
Para i= n-1, ..., 1
      Fazer direita[i] ← direita[i+1] * x_{i+1}.
fimpara
Para i= n, ..., 1
      Fazer [∇f]_i → esquerda[i] * direita[i].
fimpara
Retornar f e ∇f.
```

Observemos que ∇f é calculado corretamente para todos os valores de x. Além disso, só são necessários $3n-1$ produtos para calcular a função e o gradiente. É assim que uma das principais técnicas da **Diferenciação Computacional**, o modo reverso, calcularia o gradiente de f.

Este exemplo ilustra também o principal resultado da diferenciação automática: **dada uma função escalar $f : \mathbb{R}^n \to \mathbb{R}$ é possível calcular ∇f em tempo proporcional ao tempo de avaliar a função f.**

Diferenciação automática (DA) é o processo através do qual é possível calcular, de forma automática, as derivadas de uma função. A diferenciação automática baseia-se no fato de que toda função, independentemente de sua complexidade, pode ser representada como uma seqüência de operações e funções elementares. Assim, aplicando repetidamente a regra da cadeia às operações e funções elementares, é possível calcular as derivadas exatas de uma função de forma mecânica e automática.

A bibliografia disponível a respeito de diferenciação automática é enorme. As referências feitas ao longo deste trabalho não são exaustivas. As duas referências básicas para um estado da arte da diferenciação automática são os *Proceedings* do *First SIAM Workshop on Automatic Differentiation* (Janeiro de 1991) [65] e do *Second International Workshop on Computational Differentiation* (Fevereiro de 1996) [8]. Há vários grupos trabalhando em diferenciação automática na atualidade. Um deles encontra-se no *Argonne National Laboratory*. Todos os relatorios técnicos

que aparecem citados neste trabalho podem ser encontrados na *home page* do *Computational Differentiation Project*, *i.e.*, `http://www.mcs.anl.gov/autodiff/`. No momento em que estamos acabando de escrever este trabalho, Andreas Griewank e Bruce Christianson estão escrevendo um livro intitulado *"Evaluating Derivatives: Principles and Techniques of Computational Differentiation"* que pode ser encontrado em `http://www.math.tu-dresden.de/wir/staff/griewank/`, a *home page* de A. Griewank, junto com outros dos seus trabalhos.

Como apontado por Bischof e Griewank em [15], quanto mais os pesquisadores trabalham no desenvolvimento de ferramentas de Diferenciação Automática, mais evidente resulta que o processo de calcular derivadas eficientemente não é exatamente "automático". Assim, o termo "Diferenciação Automática" utilizado até pouco tempo atrás deixou de ser adequado. Muitos trabalhos estão sendo desenvolvidos no sentido de aperfeiçoar as técnicas existentes e aplicá-las eficientemente a problemas de diversos tipos. Assim, o termo por eles escolhido para se referir a estas técnicas foi Diferenciação Computacional (observemos a mudança no nome do primeiro para o segundo *workshop*). Neste trabalho, utilizamos o termo **Diferenciação Automática (DA)** para referir-nos as técnicas básicas de diferenciação, os modos direto e reverso (diferentes formas de aplicar a regra da cadeia). Com o termo **Diferenciação Computacional (DC)** descrevemos o conjunto de técnicas que, associadas com as idéias básicas, são utilizadas para calcular derivadas de forma automática e eficiente. Finalmente, em alguns casos, utilizamos o termo **Fast Automatic Differentiation (FAD)**, introduzido por Masao Iri em [75, 76], para uma técnica especial de diferenciação automática, o modo reverso.

Este trabalho esta dividido em duas partes. Na primeira, Capítulos 2 e 3, introduzimos as noções básicas da difererenciação automática e realizamos algumas simulações numéricas para conferir as propriedades teóricas das diferentes técnicas. A segunda parte, Capítulos 4, 5, 6 e 7, está dedicada à aplicação das técnicas de diferenciação computacional em problemas de otimização. Em particular, nos dedicamos à resolução de um problema de estimação de parâmetros em ótica e a problemas de controle ótimo. Com esse objetivo, desenvolvemos dois novos métodos de otimização.

O **Método de Gradientes Espectrais Projetados (SPG)** foi desenvolvido para a minimização de uma função diferenciável em conjuntos fechados e convexos. Este método combina o Método de Gradientes Projetados [6, 57, 80] com dois ingredientes recentemente desenvolvidos em otimização. Primeiro, estendemos as estratégias típicas de globalização associadas com esses métodos aos esquemas de busca linear não-monótonos desenvolvidos por Grippo, Lampariello e Lucidi [70] para métodos do tipo Newton. Segundo, associamos o comprimento espectral do passo,

introduzido por Barzilai e Borwein [5] e analisado por Raydan [100]. Esta escolha do comprimento do passo tem um baixo custo computacional e aumenta surpreendentemente a velocidade de convergência do método do gradiente.

Os resultados obtidos com o SPG e com o Método de Gradientes Espectrais (SGM) [101], nos sugeriram que as idéias de gradientes espectrais e gradientes conjugados poderiam ser combinadas obtendo, assim, algoritmos ainda mais eficientes. O **Método de Gradientes Conjugados Espectrais (SCG)** combina o Método de Gradientes Conjugados clássico [73] com o comprimento de passo espectral para minimização irrestrita.

Calculando o gradiente da função objetivo pelo modo reverso de diferenciação automática, utilizamos o SGM e o SCG para resolver um problema de estimação de constantes óticas. Finalmente, utilizando uma estratégia mista dos modos diretos e reversos, aplicamos o SPG para a resolução de problemas de controle ótimo integrados por métodos na família Runge-Kutta.

No Capítulo 2, descrevemos as principais técnicas de diferenciação automática e introduzimos uma nova ferramenta de DC: ADIC++. No Capítulo 3, utilizamos esta ferramenta para o cálculo de Jacobianos esparsos na resolução de sistemas de equações não-lineares pelo método de Newton. No Capítulo 4, utilizamos o Método de Gradientes Espectrais para, resolvendo um problema de minimização irrestrita, estimar a espessura, o índice de refração e o coeficiente de atenuação de um filme fino. No Capítulo 5, introduzimos o Método de Gradientes Espectrais Projetados para minimização em conjuntos convexos. No Capítulo 6, utilizamos este método e uma estratégia mista dos modos reverso e direto para resolver problemas de controle ótimo. No Capítulo 7, introduzimos o Método de Gradientes Conjugados Espectrais e o aplicamos ao problema de ótica descrito no Capítulo 4. Finalmente, no Capítulo 8, apresentamos algumas conclusões.

Capítulo 2

Diferenciação Computacional

2.1 Introdução

Muitos métodos numéricos utilizados em otimização e na resolução de sistemas de equações não-lineares requerem o cálculo das derivadas de uma função $f : \mathbb{R}^n \to \mathbb{R}$. Tanto a exatidão como o custo computacional do cálculo são de relevante importância para a robustez e a velocidade do método numérico.

Existem pelo menos quatro formas de calcular derivadas. A primeira destas formas é à **mão**, com o inconveniente de que, dependendo do tamanho e a complexidade da função, este método pode se tornar inviável e muito susceptível a erro nos cálculos. Porém, talvez ainda seja o método que, "dependendo das mãos", gere o código mais eficiente. Pelo método de **aproximação por diferenças finitas**, as derivadas parciais da f são aproximadas, entre outras formas, por diferenças avançadas, *i.e.*,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + h_i\, e_i) - f(x)}{h_i} \tag{2.1}$$

ou por diferenças centrais, *i.e.*,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + h_i\, e_i) - f(x - h_i\, e_i)}{h_i}, \tag{2.2}$$

onde $0 < h_i \in \mathbb{R}$ é o passo de discretização e $\{e_1, e_2, \ldots, e_n\}$ é a base canônica de $\mathbb{R}^n$. As desvantagens de utilizar este método são: (i) o número de avaliações de função necessário, (ii) a dificuldade para a eleição dos parâmetros de diferenciação h_i e (iii) a possível falta de precisão no cálculo. A respeito da escolha dos parâmetros de diferenciação, h_i "grandes" incorrerão em erros inaceitáveis e h_i "pequenos" causarão erros de truncamento. Em [3, 12] destaca-se como

vantagem do método de diferenças divididas que a função f transforma-se numa "caixa preta". Porém numa implementação minuciosa deveria ser aproveitado o cálculo de $f(x)$ para os cálculos de $f(x + h\, e_i)$ e $f(x - h\, e_i)$ $(i = 1, \ldots, n)$. Consideremos a função escalar

$$f(x_1, x_2) = x_1 + g(x_2)$$

e suponhamos que é avaliada no ponto $(\bar{x}_1, \bar{x}_2)$. Para calcular $\partial f / \partial \bar{x}_1$ utilizando (2.1), precisaremos avaliar

$$f(\bar{x}_1 + h_1, \bar{x}_2) = \bar{x}_1 + h_1 + g(\bar{x}_2).$$

Assim, poderíamos utilizar o valor de $g(\bar{x}_2)$ calculado anteriormente e adicionar-lhe $\bar{x}_1 + h_1$. Se a função g é uma função custosa, este tipo de aproveitamento pode ser muito vantajoso. Algumas técnicas que utilizam estas idéias serão discutidas mais adiante neste capítulo. Outra alternativa é a utilização da **diferenciação simbólica**, implementada em pacotes tais como Maple, Reduce e Macsyma. Dada uma seqüência de caracteres descrevendo a definição de uma função, os pacotes de manipulação simbólica fornecem expressões algébricas para as derivadas. Suas principais desvantagens são: (i) a dificuldade de aproveitar cálculos comuns e (ii) as limitações impostas pela quantidade de recursos necessários. A **diferenciação automática (DA)** baseia-se no fato de que toda função, independentemente de sua complexidade, pode ser representada como uma seqüência de **operações elementares**, tais como $+$, $-$, $\times$ e $/$, e **funções elementares** (sin, cos, $\sqrt{\cdot}$, etc.). Assim, aplicando repetidamente a regra da cadeia às operações e funções elementares é possível calcular as derivadas exatas de uma função de forma mecânica e automática.

Na Seção 2.2, descrevemos as principais técnicas de diferenciação automática: os modos direto e reverso. Na Seção 2.3, apresentamos uma classificação das ferramentas de DC. A Seção 2.4 introduz o pacote ADIC++, uma nova ferramenta de DC. Na Seção 2.5, mostramos resultados numéricos comparando as diferentes técnicas de diferenciação.

2.2 Diferenciação automática: acumulação direta e inversa

Existem essencialmente duas técnicas para diferenciar automaticamente funções: **diferenciação automática com acumulação direta** (em inglês: *forward, contravariant* ou *bottom-up mode*) e **diferenciação automática com acumulação inversa** (em inglês: *reverse, backward, covariant* ou *top-down mode*). Ilustraremos ambos métodos como o seguinte exemplo. Consideremos a função

$$f(x_1, x_2) = \sin(x_1) + x_1 x_2. \tag{2.3}$$

Daqui em diante, dada uma função $f : \mathbb{R}^n \to \mathbb{R}^m$, $J(x)$ denotará a matriz Jacobiana. Se $m = 1$, denotaremos o vetor gradiente por ∇f, *i.e*,

$$\nabla f(x) = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix} \quad \text{e} \quad J(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \nabla f_2(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix}.$$

As subrotinas da Figura 2.1 calculam f e ∇f pelos métodos de diferenciação automática com acumulação direta e acumulação inversa, respectivamente.

Rotina `fg_direto`(x_1, x_2)
$x_3 = \sin(x_1)$
$\nabla x_3 = [\cos(x_1), 0]^T$
$x_4 = x_1 x_2$
$\nabla x_4 = [x_2, x_1]^T$
$x_5 = x_3 + x_4$
$\nabla x_5 = \nabla x_3 + \nabla x_4$
Retornar $x_5, \nabla x_5$

Rotina `fg_reverso`(x_1, x_2)
$x_3 = \sin(x_1)$
$x_4 = x_1 x_2$
$x_5 = x_3 + x_4$
$\partial x_5/\partial x_5 = 1$
$\partial x_5/\partial x_3 = 1$
$\partial x_5/\partial x_4 = 1$
$\partial x_5/\partial x_1 = \partial x_5/\partial x_3 \cos(x_1)$
$\partial x_5/\partial x_1 = \partial x_5/\partial x_1 + \partial x_5/\partial x_4 \, x_2$
$\partial x_5/\partial x_2 = \partial x_5/\partial x_4 \, x_1$
Retornar $x_5, \partial x_5/\partial x_1, \partial x_5/\partial x_2$

Figura 2.1: Cálculo de f e ∇f pelos métodos de diferenciação automática com acumulação direta e inversa.

Denotemos as p **variáveis dependentes** (ou expressões intermediárias) que aparecem no cálculo da f por

$$x_i = \begin{cases} \Phi_i(x_{i_1}, x_{i_2}), \text{ se } x_i \text{ depende de dois parâmetros, ou} \\ \Phi_i(x_{i_1}), \text{ se } x_i \text{ depende de um só parâmetro,} \end{cases}$$

para $i = n+1, n+2, \ldots, n+p$. Cada $\Phi_i(\cdot)$ representa uma operação ou função elementar para a qual resulta trivial calcular as derivadas parciais com respeito aos seus parâmetros. Notemos que, em particular, temos $f = x_{n+p} = \Phi_{n+p}(\cdot)$. Por simplicidade, e só por um instante, suponhamos

que todas as funções e operações elementares têm dois parâmetros. Assim, podemos representar de forma algorítmica o modo direto como:

Subrotina 2.1: Modo direto

Para $i = 1, 2, \ldots, n$

    Iniciar $\nabla x_i = e_i$.

FimPara

Para $i = n+1, n+2, \ldots, n+p$

    Calcular $x_i = \Phi(x_{i_1}, x_{i_2})$.

    Calcular $\nabla x_i = \partial x_i / \partial x_{i_1} \nabla x_{i_1} + \partial x_i / \partial x_{i_2} \nabla x_{i_2}$.

FimPara

Retornar x_{n+p} e ∇x_{n+p}.

Analogamente, podemos representar de forma algorítmica o modo reverso como:

Subrotina 2.2: Modo reverso

Para $i = n+1, n+2, \ldots, n+p$

    Calcular $x_i = \Phi(x_{i_1}, x_{i_2})$.

FimPara

Iniciar $\partial x_{n+p} / \partial x_{n+p} = 1$.

Para $i = n+p-1, n+p-2, \ldots, n+1$

    Iniciar $\partial x_{n+p} / \partial x_i = 0$.

FimPara

Para $i = n+p, n+p-1, \ldots, n+1$

    Calcular $\partial x_{n+p} / \partial x_{i_1} = \partial x_{n+p} / \partial x_{i_1} + \partial x_{n+p} / \partial x_i \; \partial x_i / \partial x_{i_1}$.

    Calcular $\partial x_{n+p} / \partial x_{i_2} = \partial x_{n+p} / \partial x_{i_2} + \partial x_{n+p} / \partial x_i \; \partial x_i / \partial x_{i_2}$.

FimPara

Retornar x_{n+p} e $\partial x_{n+p} / \partial x_i$ para $i = 1, \ldots, n$.

No modo direto, as variáveis dependentes x_i ($i = n+1, \ldots, n+p$) são calculados junto aos seus respectivos gradientes (veja o ciclo principal da Subrotina 2.1). O cálculo de cada gradiente ∇x_i só precisa dos valores de $x_i = \Phi(x_{i_1}, x_{i_2})$, x_{i_1}, ∇x_{i_1}, x_{i_2} e ∇x_{i_2}. Por isso dizemos que no modo direto os cálculos são efetuados "localmente". O cálculo do gradiente de uma variável dependente

consiste, basicamente, na combinação linear dos vetores gradientes dos seus parâmetros. Assim, a tarefa mais custosa do modo direto é o cálculo destas combinações lineares. Se o gradiente tem poucas entradas não-nulas, há a possibilidade de trabalhar com vetores esparsos ou de calcular as derivadas parciais não-nulas em forma independente. A primeira destas opções tem a desvantagem de trabalhar com estruturas dinâmicas e a segunda requer que uma subrotina seja executada uma vez para cada derivada parcial não-nula.

No modo reverso, as variáveis dependentes e as derivadas da função f com respeito a estas (também chamadas de **valores adjuntos**) calculam-se na ordem inversa (veja os ciclos da Subrotina 2.2). Por esse motivo, é preciso guardar um registro das "dependências" que permita voltar pelo caminho andado. Assim, o modo reverso, requer memória suficiente para decorar o caminho de volta e armazenar os valores das variáveis dependentes que serão utilizados no cálculo das derivadas. Um ponto importante a respeito é que tanto o "caminho" como os valores das variáveis dependentes serão requeridos exatamente na ordem inversa e seqüencialmente. Este detalhe facilitará as tentativas por resolver tal problema.

Resumindo, podemos dizer que o modo direto apresenta um problema "temporal" enquanto o modo reverso apresenta um problema "espacial". Se o número de variáveis dependentes é pequeno, o modo reverso é preferível. Porém, se a expressão da função f é "complicada", *i.e.*, com muitas variáveis dependentes, e o número de variáveis independentes é pequeno, então o modo direto é o mais indicado. Ainda neste capítulo, comentaremos algumas tentativas para resolver os problemas apresentados.

Os modos direto e reverso também podem ser explicados considerando o **grafo computacional** associado a uma função. A noção de grafo computacional ou grafo de Kantorovich foi introduzida por Kantorovich em [79]. As Figuras 2.2 e 2.3, que mostram o cálculo do gradiente da função (2.3) pelos modos direto e reverso respectivamente, são auto-explicativas. Veja [75, 76] para uma descrição clara do formalismo dos grafos computacionais e sua relação com os modos direto e reverso de diferenciação automática.

No modo direto, o grafo é percorrido uma única vez enquanto no modo reverso é percorrido duas. Esta representação das técnicas de diferenciação automática não deve ser confundida com o seguinte. Uma classe importante de ferramentas de diferenciação automática baseia-se em, dado um algoritmo para a avaliação da função f, "transformá-lo" num algoritmo que calcule a função e suas derivadas. Nesse tipo de ferramentas, o modo direto não precisa de nenhuma estrutura de dados para efetuar a transformação. Porém, o modo reverso precisa sim de algum tipo de armazenamento para guardar a informação que lhe permita "desfazer o caminho". Essa

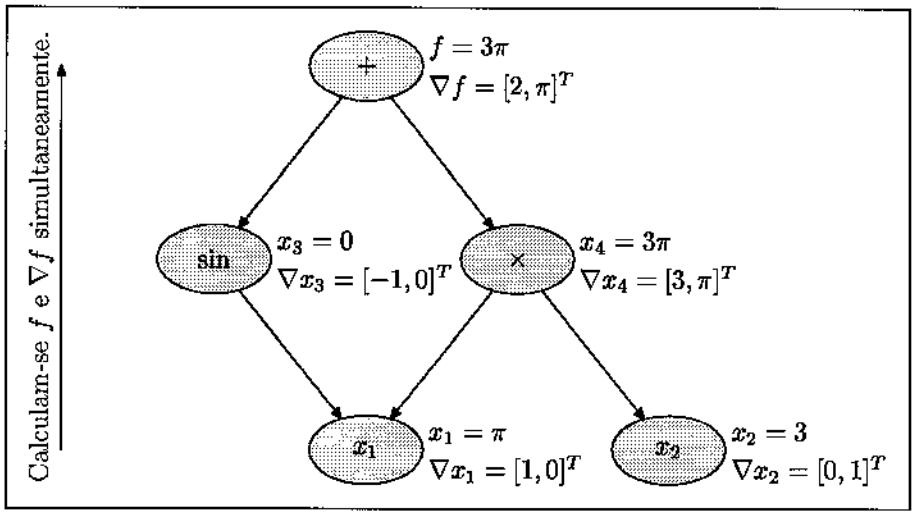


Figura 2.2: Modo direto

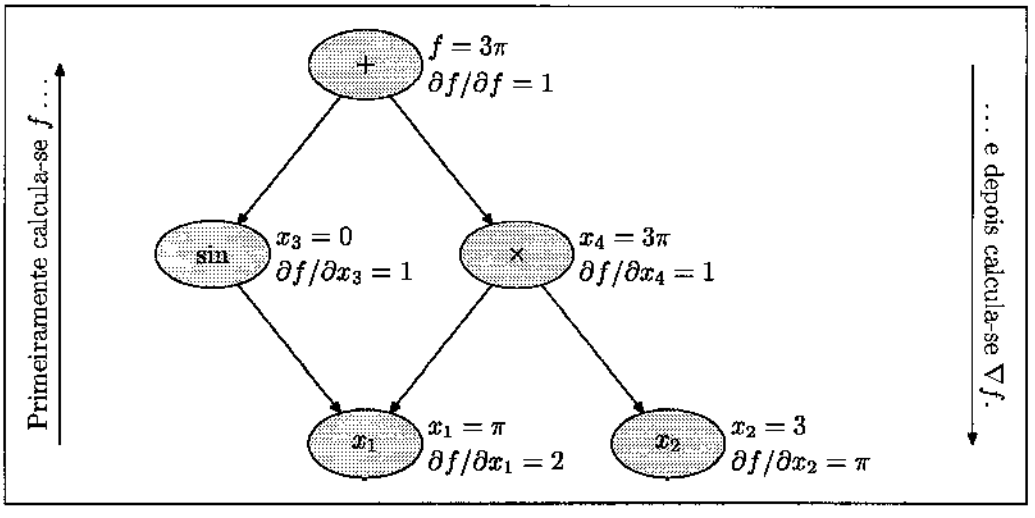


Figura 2.3: Modo reverso

estrutura pode ser um grafo computacional (ou alguma estrutura similar), como em JAKEF [74] ou um arquivo seqüencial (chamado de *tape*), como em ADOL-C [69].

Lembrando que denotamos por p ao número de variáveis dependentes (proporcional ao custo de avaliar f) e por n ao número de variáveis independentes, vejamos as complexidades espacial e temporal das técnicas de diferenciação automática apresentadas. O modo direto que não leva em consideração a esparsidade do gradiente tem complexidade temporal $O(np)$ e complexidade espacial $O(n)$. Já uma implementação que considere esparsidade tem complexidades espacial

e temporal $O(n_{\neq 0}p)$ e $O(n_{\neq 0})$, respectivamente, onde $n_{\neq 0}$ representa o número de derivadas parciais não-nulas (conhecido *a priori*). Finalmente, o modo reverso tem complexidades espacial e temporal $O(p)$ (independentes de n).

As complexidades apresentadas derivam de problemas inerentes à definição das diferentes técnicas. Porém muitos trabalhos foram e estão sendo dedicados à aplicação eficiente dos métodos de diferenciação automática. Em [13, 17, 18, 19], Bischof *et al.* apresentam SparsLinC (*Sparse Linear Combination*), uma biblioteca de métodos que possibilitam o tratamento transparente de vetores e matrizes esparsas. Os esforços nessa direção apontam, principalmente, a diminuir o custo da combinação linear de vetores (o maior problema do modo direto). Nos trabalhos acima citados, são apresentados alguns resultados numéricos combinando ADIFOR (*Automatic Differentiation in Fortran*) [11], uma ferramenta para a diferenciação de funções escritas em Fortran, e SparsLinC [17]. Uma outra idéia baseada na compactação de Jacobianos estruturalmente ortogonais é apresentada em [3]. Este último enfoque será brevemente descrito e comparado com o anterior no Capítulo 3. Em [67], Griewank mostra como a complexidade espacial do modo reverso pode ser reduzida a $O(log(p))$. As técnicas por ele utilizadas diminuem a complexidade espacial compensando tal ganho com o aumento da complexidade temporal. Esse equilíbrio entre ambas complexidades não é muito simples de medir e, como o próprio Griewank comenta, ele "tentará ser tão realista e independente da plataforma computacional quanto seja possível" para medir o crescimento da complexidade temporal. Diferentes técnicas para a eliminação de nós do grafo computacional e a detecção de estruturas independentes são apresentadas em [66] e [20], respectivamente. Um resumo de outras idéias e sua aplicação a um problema inverso que envolve a solução de equações diferenciais parciais é apresentado em [30]. Finalmente, limitantes para as complexidades de calcular gradientes, Jacobianos e Hessianas pelos modos direto e reverso, utilizando o conceito de funções parcialmente separáveis [63, 36, 95], é apresentado por Griewank em [68].

2.3 Ferramentas de diferenciação computacional: uma classificação.

As ferramentas de DA fornecem suporte para produzir os valores das derivadas de uma função a partir de alguma representação da função. Cada ferramenta de DA assume que essa função está representada em alguma linguagem fonte, que pode ser simbólica ou de programação. As ferramentas de DA auxiliam-se na transformação da linguagem fonte a uma forma que possa ser facilmente utilizada para gerar os valores das derivadas. Essa transformação pode ser explícita

ou implícita. A classificação apresentada em [78], e resumida nesta seção, baseia-se em como, quando e onde é feita esta transformação.

A mais básica das classes está composta pelas ferramentas de DA **elementares**. Estas ferramentas baseiam-se na premissa de que todas as funções podem ser decompostas numa seqüência de operações elementares. O valor da derivada de uma operação elementar depende só dos valores dos argumentos da operação e de suas derivadas. Portanto, o calculo de derivadas de operações elementares pode ser feito localmente e com custo fixo. As ferramentas elementares são freqüentemente implementadas como um conjunto de procedimentos, um para cada operação. Cada procedimento toma como entrada os argumentos da operação e os valores das derivadas dos argumentos, e devolve o resultado da operação e suas derivadas. A transformação da linguagem fonte é feita manualmente decompondo cada função numa seqüência de chamadas a procedimentos para cada operação elementar.

Baseada no conceito pioneiro das ferramentas elementares, a classe das ferramentas **extensivas** fornece extensões para diferenciação automática às linguagens de programação convencionais. Estas ferramentas automatizam a decomposição em seqüências de operações elementares. Geralmente, diferenciadores automáticos nesta classe utilizam pré-compiladores para transformar as extensões criadas para a linguagem original.

As ferramentas **integradoras** levam a idéia das ferramentas extensivas um passo mais adiante. As ferramentas integradoras têm a habilidade de integrar dentro do ambiente da linguagem a diferenciação automática de funções. Transformações explícitas da linguagem fonte não são necessárias. A transformação é feita implicitamente pelo compilador ou intérprete da linguagem. A diferença entre as ferramentas extensivas e integradoras é que as últimas são linguagens diferentes e não extensões menores de uma linguagem existente.

As ferramentas **operacionais** têm origem nas idéias das ferramentas elementares e extensivas. A diferenciação automática é proporcionada mediante a definição de novos tipos de dados para os quais a diferenciação pode ser feita com simplicidade. Nestas ferramentas a transformação necessária para produzir derivadas é feita implicitamente quando a linguagem de programação interpreta os novos tipos de dados.

A última classe de ferramentas de DA é a das ferramentas **simbólicas**. As ferramentas nesta classe processam uma representação simbólica da função em tempo de execução ou produzem uma função diferenciada que é traduzida numa linguagem de programação *standard*.

A noção de que funções complicadas podem ser decompostas numa seqüência de operações elementares é a base das ferramentas de diferenciação automática. É nesta idéia que se baseia a

classe das ferramentas elementares. O fato de como as funções são mais naturalmente representadas diferencia as outras classes. A classe simbólica baseia-se na idéia de que as funções deveriam ser representadas algebricamente e não como algoritmos. As classes extensiva, integradora e operacional baseiam-se na idéia oposta e se diferenciam entre elas no modo em que a diferenciação automática é proporcionada. A classe operacional se baseia na idéia de que a diferenciação automática deve ser proporcionada por um novo tipo de dados numa linguagem com sobrecarga de operadores. A classe extensiva baseia-se na idéia de estender uma linguagem de programação para incluir diferenciação automática. Finalmente, a classe integradora baseia-se na idéia de que a diferenciação automática deve ser proporcionada por uma linguagem de programação. A relação entre as diferentes classes é mostrada na Figura 2.4 [78].

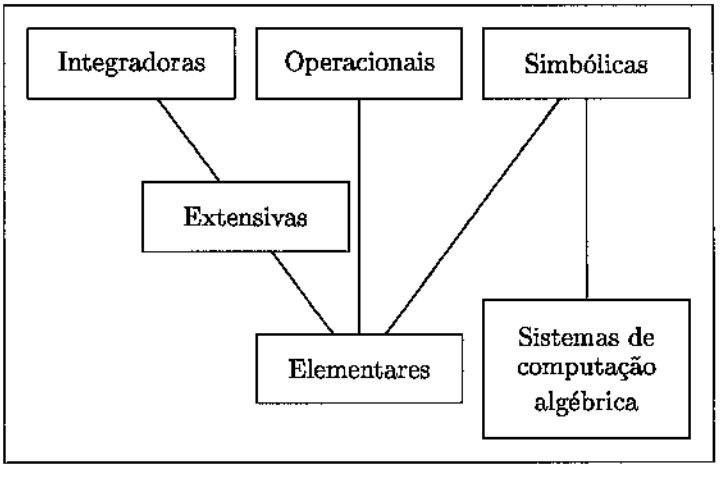


Figura 2.4: Relação entre as diferentes classes de ferramentas de DA.

2.4 Uma ferramenta de diferenciação computacional

Nesta seção descrevemos o desenvolvimento de uma ferramenta de diferenciação computacional. O pacote ADIC++ é uma ferramenta de diferenciação computacional implementada em C++ que fornece uma base simples e eficiente para o cálculo de derivadas. Embora não seja muito simples de classificar, podemos dizer que ADIC++ pertence à classe das ferramentas de DC simbólicas. Além de calcular derivadas pelos modos direto e reverso, ADIC++ também calcula derivadas por diferenciação simbólica e diferenças finitas. Estas técnicas foram incorporadas com o intuito de testar e comparar os procedimentos de diferenciação automática descritos. As implementações das técnicas de diferenças finitas e diferenciação simbólica têm incorporados me-

canismos para a detecção e aproveitamento de expressões comuns. Gostaríamos de ressaltar que ADIC++ é uma ferramenta teste, pois a implementação conjunta de todas estas técnicas numa só ferramenta fez com que os procedimentos e as estruturas de dados utilizadas não sejam as mais adequadas para cada um dos casos.

As operações elementares reconhecidas são: adição (+), subtração (−), multiplicação (∗), divisão (/); e as funções elementares são: potenciação (`pow(·,·)`), radiciação (`sqrt(·)`), seno (`sin(·)`), co-seno (`cos(·)`), tangente (`tg(·)`), logaritmo neperiano (`ln(·)`), logaritmo base dez (`log(·)`) e exponenciação (`exp(·)`).

Como apontado em [1], as ferramentas de diferenciação automática ADIFOR [21] e ADIC [22], para a diferenciação de funções implementadas em Fortran 77 e C respectivamente, implementam algoritmos relativamente simples para a propagação das derivadas. Isto deve-se a que a maior parte do tempo de desenvolvimento das ferramentas citadas foi concentrado na interpretação e manipulação de um grande número de primitivas e semânticas das linguagens fonte nas quais as funções são representadas. Assim, na primeira versão de nossa ferramenta, criamos uma linguagem simples para a representação de funções. A Figura 2.5 mostra a representação da função

$$f(x_1, x_2, x_3) = (x_1^2 + x_2, x_2 + x_2 x_3, \ln(x_2)). \tag{2.4}$$

```
n= 3
m= 3
f1= pow(x1,2) + x2
f2= x2 + x2 * x3
f3= ln(x2)
```

Figura 2.5: Arquivo de entrada da função (2.4).

O primeiro passo de ADIC++ é interpretar o arquivo de entrada e criar as estruturas de dados para a representação interna da função. A cada função escalar $f_i : \mathbb{R}^n \to \mathbb{R}$, para $i = 1, \ldots, m$, é associado um grafo computacional como se descreve em [75, 76]. Para cada variável x_i $(i = 1, \ldots, n)$ é criada uma lista, que chamaremos de **referências**(x_i), com as funções escalares nas quais aparece a variável x_i. Na Figura 2.6 mostramos as estruturas criadas para a função (2.4). Notemos que, utilizando as listas de referências, é simples conhecer as derivadas parciais "estruturalmente não-nulas".

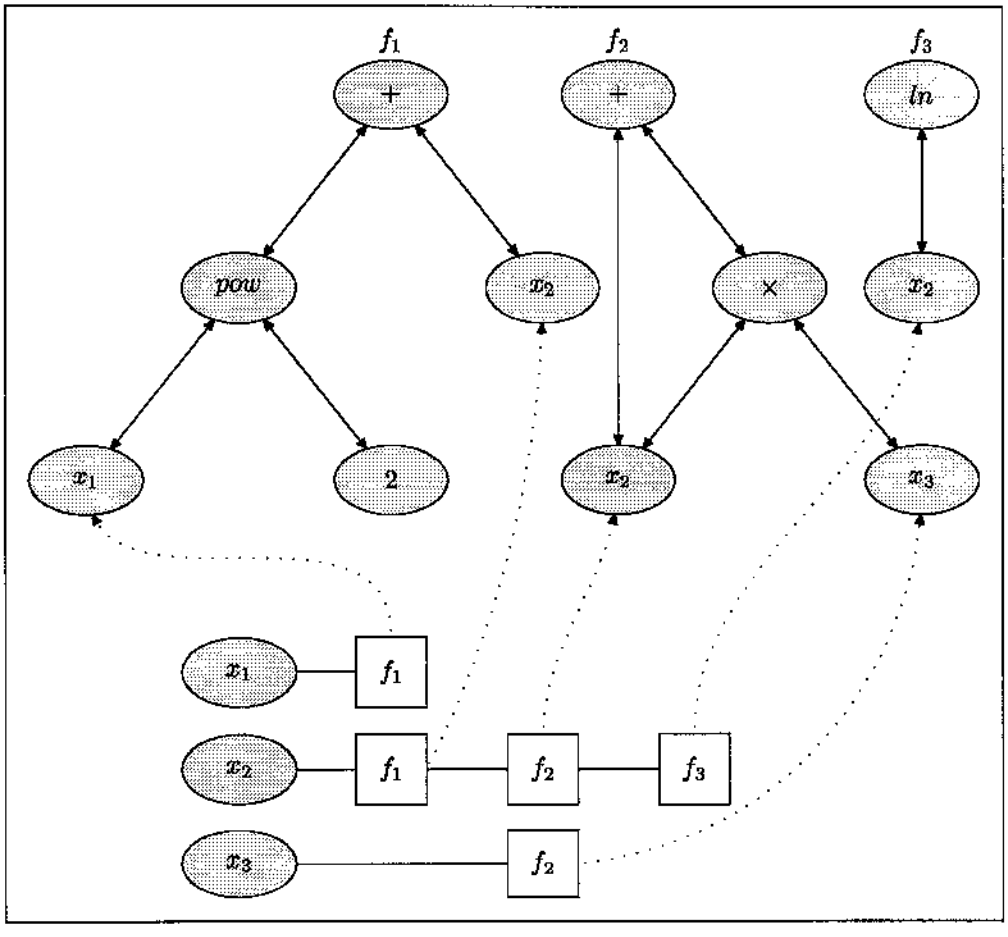


Figura 2.6: Estruturas de dados associadas à função (2.4).

A seguir descrevemos alguns detalhes de implementação das técnicas de diferenciação fornecidas por ADIC++.

2.4.1 Diferenças divididas

Uma implementação simples de diferenças finitas, que utilize a aproximação

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + h_i\, e_i) - f(x)}{h_i},$$

precisa de $n+1$ avaliações da função f para calcular f e aproximar ∇f; e tem um custo de $O(n+1)$ vezes o custo de calcular f. Porém, como mencionamos anteriormente, aproveitando expressões comuns entre avaliações da forma $f(\bar{x})$ e $f(\bar{x} + h_i\, e_i)$ este custo pode ser diminuído. Além disso,

uma implementação "tradicional" não tem como calcular só as derivadas parciais não-nulas e, assim, o custo acima mencionado independe da esparsidade do gradiente.

Na nossa ferramenta, a detecção das derivadas parciais estruturalmente nulas é resolvido utilizando as listas de referências. O aproveitamento de expressões comuns é feito como se segue:

1. Quando a função f é avaliada no ponto $\bar{x}$ o valor de cada variável dependente é armazenado no nó correspondente com um campo indicando que o valor armazenado é VERDADEIRO.

2. Para aproximar $\partial f/\partial \bar{x}_i$ o grafo computacional é percorrido começando pelo nó da variável x_i e indicando que o valor armazenado nos nós é FALSO.

3. Para calcular $f(\bar{x}+h_i\, e_i)$, são percorridos só aqueles nós com o indicador em FALSO, calculando as variáveis dependentes no ponto $\bar{x} + h_i e_i$ e pondo o indicador em VERDADEIRO, mas sem armazenar o valor calculado.

Assim, no fim, a função f é avaliada no ponto $\bar{x} + h_i e_i$ calculando só os valores necessários e o grafo computacional está pronto para uma nova avaliação num outro ponto da forma $\bar{x} + h_k e_k$.

2.4.2 Diferenciação simbólica

A diferenciação simbólica baseia-se na idéia de, dada uma função representada de alguma forma, criar a representação de suas derivadas. Esta representação pode ser simbólica ou algorítmica, *i. e.*, uma subrotina escrita numa linguagem de alto nível qualquer. Nesta implementação criamos, a partir do grafo computacional da função, e utilizando as listas de referências, os grafos computacionais de suas derivadas parciais não-nulas. Esta tarefa é realizada uma única vez.

O aproveitamento de expressões comuns entre a função e suas derivadas é resolvido interligando os grafos computacionais. Consideremos uma função escalar $f = g(h(x), k(x))$. Se, ao criar o grafo computacional de uma de suas derivadas parciais aparece a própria expressão da g ou de algum dos seus parâmetros (h ou k), então o grafo computacional da derivada fica apontando para o nó correspondente no grafo computacional da função. A Figura 2.7 mostra os grafos computacionais da função

$$f(x_1, x_2) = \sin(x_1)\sin(x_2)$$

e sua derivada parcial

$$\partial f/\partial x_1 = \cos(x_1)\sin(x_2) + \sin(x_1)\cos(x_2).$$

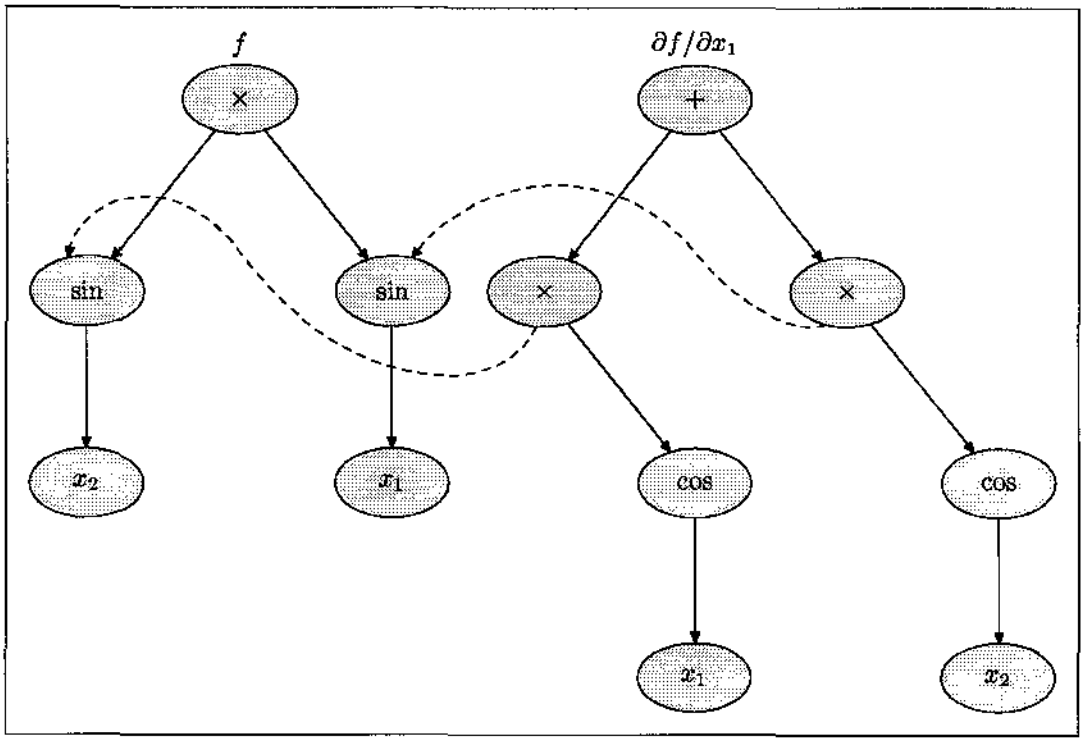


Figura 2.7: Aproveitamento de expressões comuns na diferenciação simbólica.

Os maiores inconvenientes da diferenciação simbólica são: (i) o tempo de criação dos grafos computacionais das derivadas e (ii) a memória necessária para o seu armazenamento.

2.4.3 Diferenciação automática com acumulação direta

Há duas formas de implementar o modo direto de diferenciação automática. Na primeira delas calculam-se todas as derivadas parciais simultaneamente (utilizando ou não vetores esparsos). A outra forma é calcular as derivadas não-nulas em forma independente, percorrendo o grafo computacional uma vez para cada derivada parcial.

Pelo fato de estarmos desenvolvendo uma ferramenta orientada ao tratamento de problemas de grande porte, com gradientes potencialmente esparsos, descartamos a possibilidade de calcular todas as derivadas parciais simultaneamente utilizando vetores de dimensão estática. Assim, restam as possibilidades de calcular o gradiente utilizando alguma representação dinâmica para vetores ou calcular as derivadas parciais não-nulas em forma independente. Escolhemos esta

última alternativa que não apresenta o *overhead* de trabalhar com estruturas dinâmicas. Porém a escolha depende muito do problema e merece uma analise detalhada. Veja em [17] o desenvolvimento de uma biblioteca de rotinas para o tratamento de estruturas esparsas associadas ao modo direto de diferenciação automática.

2.4.4 Diferenciação automática com acumulação inversa

Neste modo da diferenciação automática, o grafo computacional associado à função f é percorrido duas vezes. A primeira para avaliar a função e a segunda, em sentido inverso, para calcular as derivadas parciais. Na primeira varredura é preciso armazenar os valores das variáveis dependentes e o "caminho" que deverá ser percorrido no sentido inverso. Uma possibilidade é armazenar estes dados num arquivo seqüencial ou *tape* [69]. Uma outra alternativa é criar o grafo computacional [74]. Assim, guardamos os valores das variáveis dependentes nos nós e representamos as dependências com os arcos. Na primeira das alternativas temos a vantagem de poder armazenar um número arbitrário de variáveis dependentes e a desvantagem de acessar memória lenta (arquivos). Na alternativa dos grafos computacionais temos a vantagem de armazenar os dados em memória rápida (RAM) e a desvantagem de estarmos limitados pela quantidade de memória do computador utilizado. Optamos por criar os grafos computacionais, mas, novamente, a decisão depende de vários fatores que devem ser analisados cuidadosamente.

2.5 Resultados numéricos

O *Kissing Problem* é um dos mais famosos problemas de empacotamento. Dada uma esfera de raio r no espaço d-dimensional $\mathbb{R}^d$, o objetivo é maximizar o número de esferas não-superpostas e de raio r que sejam tangentes à uma esfera central. Claramente, este número é 6 se $d = 2$. Durante muitos anos, matemáticos com a reputação de Newton e Gregory discutiram a respeito do *kissing number* associado com $d = 3$. Só no século XX foi provado que este número é 12. Muitos problemas relacionados ao *kissing number* para diferentes dimensões continuam abertos (ver [37]).

Como mostra-se em [89], todos os *Kissing Problems* poderiam ser resolvidos achando a solução

global do seguinte problema:

$$\begin{aligned} & \text{Minimizar } z \\ \text{sujeito a } & ||y_k||^2 = 1, \text{ para } k = 1, \ldots, p, \\ & z - \langle y_i, y_j \rangle - w_{ij} = 0 \text{ e} \\ & w_{ij} \geq 0, \text{ para todo } i \neq j, 1 \leq i, j \leq p, \end{aligned} \tag{2.5}$$

onde $y_k \in \mathbb{R}^d$, $||\cdot||$ é a norma euclideana e $p \in I\!N$ é arbitrário. De fato, se na solução global do problema (2.5) o valor da função objetivo é maior que 1, então o *kissing number* associado a d é maior ou igual a p.

Os métodos de programação não-linear são, em geral, capazes de achar minimizadores locais ou pontos estacionários mas, usualmente, não garantem ter achado um minimizador global. Obviamente, iniciando um método de programação não-linear com vários pontos iniciais diferentes, aumenta-se a probabilidade de achar um minimizador global do problema. Este enfoque é utilizado em [89] associado a um método do tipo Lagrangeano aumentado.

Com a mudança de variáveis $w_{ij} = v_{ij}^2$, para todo $i \neq j$, o problema (2.5) tem o formato geral

$$\begin{aligned} & \text{Minimizar } f(x) \\ & \text{sujeito a } h(x) = 0. \end{aligned} \tag{2.6}$$

Assim (ver [86]), as soluções do sistema de equações não-lineares

$$\begin{aligned} \nabla f(x) + h'(x)^T \lambda = 0 \\ h(x) = 0 \end{aligned} \tag{2.7}$$

serão candidatas a minimizador local do problema (2.5). Uma forma de resolver o sistema de equações (2.7) é pelo método de Newton. Assim, resulta necessário calcular o Jacobiano do sistema (2.7). Nesta seção, testamos as técnicas de diferenciação apresentadas anteriormente para calcular, em forma independente e pelo modo reverso, as linhas destes Jacobianos. A Tabela 2.1 mostra os problemas considerados (veja [37] para uma relação dos problemas abertos). A dimensão dos problemas ($n = pd + p(p-1)/2 + 1$) varia entre 34 e 3896. No caso $n = 3896$ o número de derivadas parciais não-nulas do Jacobiano é 130995 de um total de $3896 \times 3896 = 15178816$. A característica principal destes Jacobianos é que o número de variáveis de cada equação é pequeno e não depende de n.

Todos os experimentos foram rodados numa SPARCstation 20, com 128 MBytes de memória RAM e com o sistema operacional SunOs Release 4.1.3_U1. O código de ADIC++ está na

Problema			Tempo para o	Número de derivadas
d	p	n	cálculo da função	parciais não-nulas
2	6	34	0.00	1494
4	24	397	0.12	7740
4	25	426	0.13	8400
5	40	1021	0.40	26340
5	46	1312	0.55	34845
6	72	3061	1.57	100980
6	82	3896	2.08	130995

Tabela 2.1: Características das funções testadas.

	Diferenças finitas		Taxa
n	Sem aproveitamento	Com aproveitamento	de aproveitamento
34	0.03	0.02	0.50
115	0.43	0.13	0.70
397	5.75	1.75	0.70
426	6.85	1.98	0.71
1021	40.87	11.52	0.72
1312	68.95	19.42	0.72
3061	410.33	113.58	0.72
3896	646.90	192.15	0.70

Tabela 2.2: Aproveitamento de expressões comuns em diferenças finitas.

linguagem C/C++ e foi compilado com o compilador g++ (GNU project C and C++ compiler v2.7.2) com a opção de otimização do compilador -O4.

A Tabela 2.2 mostra o tempo de CPU utilizado para aproximar os Jacobianos por diferencias avançadas com e sem aproveitamento nos cálculos. Observemos que, descartando a primeira medição, a taxa média de aproveitamento é de 71.43%. A Tabela 2.3 mostra o tempo de CPU utilizado pelos diferentes métodos descritos na seção anterior. Para $n \geq 397$ o método de diferenciação simbólica precisou de mais memória que a disponível para criar os grafos computacionais das derivadas e, portanto, não pode ser utilizado. Claramente, o modo reverso de diferenciação automática foi o mais bem sucedido nos nossos problemas teste.

2.6 Conclusões

Neste capítulo mostramos as principais técnicas de diferenciação automática. Também descrevemos o desenvolvimento de ADIC++, uma nova ferramenta de DC. Realizamos alguns testes

n	Diferenciação simbólica	Diferenças finitas	Diferenciação automática	
			Modo direto	Modo reverso
34	0.13 + 0.08	0.02	0.03	0.00
115	1.40 + 0.85	0.13	0.30	0.02
397		1.75	4.08	0.07
426		1.98	4.40	0.08
1021		11.52	25.37	0.25
1312		19.42	41.32	0.32
3061		113.58	263.58	0.95
3896		192.15	430.13	1.25

Tabela 2.3: Comportamento das diferentes técnicas para o cálculo de derivadas.

computacionais com dois objetivos: (i) verificar as propriedades teóricas dos modos direto e reverso de DA e (ii) testar ADIC++. Analisando os testes realizados, concluímos que:

1. No problema escolhido, o modo reverso mostrou-se claramente superior.

2. O reaproveitamento de cálculos na aproximação por diferenças finitas foi considerável.

3. A nossa implementação de diferenciação simbólica mostrou claramente o principal problema desta técnica: a quantidade de recursos necessários impossibilita o tratamento de problemas de grande porte.

4. Desenvolvemos uma ferramenta confiável que implementa as técnicas básicas de DA.

Consideramos importante mencionar que ADIC++ é uma ferramenta teste. Nos experimentos numéricos, detectamos dois pontos onde ADIC++ poderia ser melhorado: a linguagem para a representação das funções e as estruturas de dados que utilizam memória dinâmica.

A respeito da linguagem para representar as funções há duas alternativas: melhorar a existente ou utilizar uma linguagem tradicional como Fortran ou C. A primeira alternativa simplificaria o ingresso dos dados e facilitaria a transformação da função numa representação na qual seja simples de diferenciar. A segunda alternativa permitiria diferenciar funções já programadas nas linguagens mencionadas e pouparia ao usuário de ter que aprender uma nova linguagem e reescrever suas funções. Esta escolha será motivo de estudos futuros.

ADIC++ utiliza estruturas de dados que foram idealizadas para trabalhar com vários métodos de diferenciação. Verificadas as propriedades dos modos direto e reverso de DA, as outras técnicas não são mais necessárias. Assim, as estruturas de dados deveriam ser analisadas e aperfeiçoadas

para trabalhar exclusivamente com DA. Este assunto também está relacionado com o anterior pois, como já mencionamos, dependendo do formato de entrada, o modo direto não precisa de estruturas de dados.

Assim, uma nova versão de ADIC++ deverá ser desenvolvida, melhorando os módulos existentes e criando outros para ampliar as capacidades da ferramenta.

Capítulo 3

Diferenciação Computacional em Otimização

3.1 Introdução

Muitos problemas práticos de física, engenharia, economia e outras ciências são modelados de maneira muito conveniente por sistemas não-lineares da forma $F(x) = 0$. É usual, nesses casos, que alguma versão do método de Newton [116] seja utilizada com sucesso (ver [86]). Dada

$$F : \mathbb{R}^n \to \mathbb{R}^n,\ F = [f_1, f_2, \ldots, f_n]^T \in C^1(\mathbb{R}^n),$$

consideremos o problema de achar x^* tal que

$$F(x^*) = 0.$$

O método de Newton é um método iterativo que, baseado na aproximação linear

$$L_k(x) = F(x_k) + F'(x_k)(x - x_k),$$

define x_{k+1} como a solução de

$$L_k(x) = 0.$$

Desta forma, a iteração k do método de Newton consiste em calcular

$$F'(x_k)s_k = -F(x_k) \tag{3.1}$$

e

$$x_{k+1} = x_k + s_k.$$

Se substituirmos em (3.1) o Jacobiano analítico pela sua aproximação por diferenças finitas $B(x_k)$ obtemos o chamado método de Newton discreto, onde (3.1) é substituída por

$$B(x_k)s_k = -F(x_k). \tag{3.2}$$

Na k-ésima iteração destes métodos há duas tarefas principais a serem realizadas: (i) calcular $F'(x_k)$ ou $B(x_k)$ e (ii) resolver o sistema (3.1) ou (3.2). Neste capítulo, com o intuito de testar as técnicas de diferenciação introduzidas no Capítulo 2 combinadas com um método de otimização, implementamos os métodos de Newton e Newton discreto. Para isso, desenvolvemos estruturas de dados e subrotinas para calcular a fatoração LU de matrizes esparsas e resolver sistemas triangulares. Ambos métodos foram testados no conjunto de problemas introduzido em [60]. A Seção 3.2 destaca alguns detalhes do cálculo dos Jacobianos. A Seção 3.3 trata a resolução dos sistemas de Newton esparsos. Na Seção 3.4, mostramos os resultados numéricos. Algumas conclusões são apresentadas na Seção 3.5.

3.2 Cálculo de Jacobianos esparsos

Há muitos trabalhos relacionados com o cálculo eficiente de Jacobianos esparsos. A maioria deles transformam o problema num problema de otimização combinatória do qual conjetura-se que seja NP-hard. Alguns trabalhos [38, 33, 94] tratam este problema combinado com a aproximação do Jacobiano por diferenças finitas. Em [65], estas técnicas são analisadas no contexto da diferenciação automática. Um dos problemas apontados em [3] para o cálculo de Jacobianos esparsos consiste em localizar na posição correta os elementos calculados de $F'(x)$. Porém, esse problema é facilmente resolvido se, dada uma função vetorial, contamos com uma estrutura de dados como a utilizada pela nossa ferramenta de DC (veja a Figura 2.6 do Capítulo 2). Um outro problema mais complexo consiste no aproveitamento de expressões comuns entre as colunas do Jacobiano. As soluções propostas em [3, 18] baseiam-se na detecção de colunas **estruturalmente ortogonais**, definidas como colunas que não possuem elementos diferentes de zero na mesma posição. Em [38], o primeiro trabalho a respeito, a separação em grupos de colunas estruturalmente ortogonais é feito da seguinte forma. As colunas são consideradas na ordem natural e uma coluna é incluída no conjunto atual se não possui um elemento diferente de zero numa posição já ocupada no conjunto. Em [33] este enfoque é melhorado mostrando que, ao considerar o problema como um problema de coloração de grafos, as colunas podem ser visitadas numa ordem mais eficiente. Em [31, 32] descrevem-se *softwares* para o problema de particionamento. Uma vez separadas as colunas em p

grupos estruturalmente ortogonais, $F'(x)$ pode ser determinado com p avaliações da forma $F'(x)v$ (ver [3]). Para cada grupo j, calculamos $F'(x)v_j$, onde $[v_j]_i = 1$ se a i-ésima coluna pertence ao grupo j e senão $[v_j]_i = 0$. Daqui em diante, $[y]_i$ denota a i-ésima coordenada do n-vetor y. Mais ainda, no enfoque do **Jacobiano compacto**, unem-se os p vetores v_j numa matriz $V \in \mathbb{R}^{n\times p}$ e calcula-se $F'(x)V$ para aproveitar a avaliação de expressões comuns entre os diferentes grupos. Quanto maior o número p de grupos, maior a vantagem de calcular o Jacobiano compacto. Os cálculos de $F'(x)v_j$ e $F'(x)V$ obtêm-se da diferenciação de $v_j^T F(x)$ e $V^T F(x)$, respectivamente. Resta ainda a tarefa de recuperar o Jacobiano original a partir do compacto, mais resulta trivial pela condição de ortogonalidade estrutural dos grupos.

Em [3] a estratégia do Jacobiano compacto (associada com ADIFOR) é comparada com a aproximação por diferenças finitas e com Jacobianos calculados à mão. Os problemas utilizados na comparação, incluídos na coleção de problemas MINPACK-2 [2], têm uma característica importante. O número p de grupos estruturalmente esparsos é independente de n (a dimensão do problema). Esta característica tem um impacto fundamental nos resultados apresentados, pois representa uma situação ideal para a estratégia dos Jacobianos compactos. Assim a comparação mostra simplesmente o esperado: resumindo, a estratégia de Jacobianos compactos ganha da aproximação por diferenças finitas e perde quando comparada com as derivadas calculadas à mão. Quantificando, "ganha" e "perde" significam que o cálculo é 3.5 vezes mais rápido no primeiro caso e 2.5 vezes mais lento no segundo. Estes dados dependem, como se mostra em [3], da plataforma computacional utilizada.

Em [18], uma comparação muito mais interessante é apresentada. A livraria SparsLinC (*Sparse Linear Combination*) [17, 14] é um pacote de rotinas na linguagem C para a exploração de esparsidade em diferenciação automática. Em [18] a combinação SparsLinC/ADIFOR é comparada com ADIFOR (não-esparso) e com a combinação Jacobianos compactos/ADIFOR. A intenção da combinação de ADIFOR com SparsLinC é diminuir o custo da tarefa mais pesada do modo direto: a combinação linear de vetores (ver Capítulo 2). SparsLinC baseia-se numa representação "inteligente" para vetores esparsos. Vetores com um único elemento, vetores esparsos e vetores com entradas não-nulas em blocos são armazenados de formas diferentes. Há heurísticas para a mudança dinâmica de uma representação à outra e estratégias especiais para a reciclagem e reutilização de memória.

Existe uma relação clara entre a estrutura de esparsidade do Jacobiano compacto e a técnica que deve ser aplicada. Se consideramos os caso em que o Jacobiano compacto é denso, então a estratégia dos Jacobianos compactos é a adequada, pois não fará operações com elementos nulos

nem terá o *overhead* de trabalhar com estruturas dinâmicas. Por outro lado, se o Jacobiano compacto é esparso, trabalhar com uma estratégia que utilize memória dinâmica resulta conveniente pois o *overhead* introduzido é compensado pela poupança de trabalhar só com os elementos não-nulos. Uma outra vantagem do **enfoque dinâmico** (SparsLinC) é que não precisa de procedimentos especiais para a detecção da estrutura de esparsidade, enquanto o **enfoque estático** (Jacobianos compactos) precisa, por exemplo, resolver um problema de coloração de grafos para a detecção de colunas estruturalmente ortogonais.

As conclusões apresentadas em [18] mostram que a comparação entre os enfoques dinâmico e estático depende de p, o número de grupos estruturalmente ortogonais do Jacobiano compacto. Se p é independente de n, as combinações SparsLinC/ADIFOR e Jacobianos compactos/ADIFOR comportam-se de forma similar sendo que a segunda estratégia é um pouco mais eficiente. Se p cresce com n, o que praticamente neutraliza a estratégia dos Jacobianos compactos, então o enfoque dinâmico e claramente o mais adequado.

Veja [41, 4, 42] para aplicações de alocação de memória dinâmica e estruturas de dados esparsas para o armazenamento das derivadas parciais não-nulas em problemas de pequeno e médio porte ($n \leq 500$).

Os três problemas tratados neste trabalho provêm da discretização de sistemas de equações diferenciais parciais não-lineares. Assim, o número de variáveis por equação é constante e não depende da dimensão do problema (número de pontos internos da malha de discretização). Estes problemas têm duas características principais: (i) as equações não possuem expressões comuns para serem aproveitadas e (ii) p, o número de grupos de colunas estruturalmente ortogonais, não depende de n, pois o Jacobiano é uma matriz banda. Esta última característica encaixa nossos problemas no caso em que a estratégia dos Jacobianos compactos parece a mais adequada (segundo as conclusões em [18]). Porém, neste caso (onde p independe de n), o comportamento das estratégias estática e dinâmica pode ser considerado equivalente, desprezando um fator constante que depende de detalhes de implementação (comentaremos ao respeito nas Conclusões gerais deste trabalho). Assim, visando a generalidade do nosso *software* (ver Capítulo 2) optamos por uma estrutura dinâmica de dados que também seja útil nos casos em que p depende de n. As linhas do Jacobiano podem ser calculadas pelo modo direto ou o modo reverso. Descartamos o modo direto porque nestes problemas, onde as equações são "simples", o modo reverso não apresenta o problema de grandes requerimentos de memória e, portanto, torna-se mais eficiente. Assim, utilizamos o modo reverso de diferenciação automática para calcular, individualmente, as linhas de $F'(x)$, *i.e.*, $\nabla f_i(x)^T$. Os elementos não-nulos calculados são armazenados na estrutura

dinâmica que descreveremos na próxima seção. Em [18] uma estratégia diferente para o cálculo de Jacobianos esparsos com o modo reverso é analisada.

Para a aproximação de $F'(x_k)$ no método de Newton discreto, escolhemos a aproximação por diferenças avançadas

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x + h_j\, e_j) - f_i(x)}{h_j}$$

que requer n avaliações extra de $f_i(x)$ e tem um erro da ordem $O(h_j)$. A aproximação por diferenças centrais

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x + h_j\, e_j) - f_i(x - h_j\, e_j)}{2h_j},$$

embora tenha um erro da ordem $O(h_j^2)$, foi descartada por precisar de $2n$ avaliações extra da função $f_i(x)$. Nas fórmulas acima, $\{e_1, e_2, \ldots, e_n\}$ representa a base canônica de $\mathbb{R}^n$. O parâmetro de discretização h_j é determinado por

$$h_j = \sqrt{\texttt{tol}(x_j)},$$

onde $\texttt{tol}(x_j)$ é o menor valor representável tal que $x_j + \texttt{tol}(x_j) \neq x_j$. Lembremos que, ao aproximar a linha i de $F'(x)$, aproveitaremos as expressões comuns entre as avaliações de $f_i(x)$ e $f_i(x+h_j\, e_j)$, para $j = 1, 2, \ldots, n$, como foi descrito no Capítulo 2. Estas considerações não foram levadas em conta em [3] onde a implementação de diferenças finitas, utilizada na comparação com as técnicas de diferenciação automática, considera as funções escalares f_i como "caixas pretas".

3.3 Resolução de sistemas esparsos

Tratado o cálculo dos Jacobianos, vejamos agora como resolver os sistemas lineares. Supondo que $A \in \mathbb{R}^{n\times n}$ é uma matriz esparsa, neste parágrafo consideramos a resolução do sistema linear $Ax = b$ por meio de sua fatoração LU (ver, por exemplo, [58, 114, 115]) e a resolução dos sistemas triangulares $Ly = b$ e $Ux = y$ (com L e U armazenadas na estrutura de A). Daqui em diante, $a_{ij}^{(k)}$ denota o elemento na posição (i, j) da matriz $A^{(k)}$, onde $A^{(k)}$ é a matriz A com as modificações realizadas até a iteração k. Para o armazenamento de matrizes esparsas foi criada uma estrutura de memória dinâmica por linhas. Definimos um vetor `Linhas` de dimensão n que tem em cada entrada uma lista de pares ordenados `[Coluna,Valor]` que representam as entradas não-nulas da matriz A (ver Figura 3.1). O problema da instabilidade numérica é abordado escolhendo o pivô da iteração k dentre os elementos da coluna k tal que

$$|a_{ik}^{(k)}| \geq \eta \max_{h\geq k} |a_{hk}^{(k)}|, \text{ para } i \geq k, \tag{3.3}$$

onde η é um parâmetro de entrada tal que $0 \leq \eta \leq 1$ (ver [115]). Para esta escolha do pivô e necessário percorrer, na iteração k, a coluna k da diagonal para baixo. Embora percorrer colunas não pareça uma tarefa fácil numa estrutura organizada por linhas, o problema é trivialmente resolvido da seguinte forma. Define-se um vetor auxiliar `ColunaAtual` que contém, na posição i (para $i = k, \ldots, n$), um ponteiro ao elemento $a_{ik}^{(k)}$ (o próximo se tal elemento não existir ou NULL se não houver nenhum elemento da forma $a_{ih}^{(k)}$ com $h \geq k$). Para manter atualizado este vetor, ele começa sendo igual ao vetor `Linhas` (que aponta ao primeiro elemento de cada linha) e, ao fim da iteração k, se o elemento $a_{ik}^{(k)}$ estava sendo apontado no vetor `ColunaAtual` ele é substituído por um ponteiro ao próximo elemento na lista da linha i (ou NULL se ele for o último elemento da linha). Uma estratégia igualmente simples é adotada para acessar a matriz U que é armazenada na parte superior da matriz A. A matriz L é armazenada na parte inferior de A e portanto facilmente acessada. Quando é feita a substituição para frente para resolver o sistema $Ly = b$ guardamos num vetor auxiliar `Diagonal` os elementos da diagonal. Depois, quando resolvemos, com substituição para trás, o sistema $Ux = y$, percorremos as linhas da matriz A começando pela diagonal obtendo acesso direto a matriz U. A inserção de novos elementos e a troca de linhas são triviais nesta estrutura de dados.

Nesta implementação da fatoração LU são considerados só dois elementos para a escolha do pivô: (i) o elemento da diagonal e (ii) o de maior valor absoluto na coluna. Se $a_{kk}^{(k)}$ satisfaz (3.3) então ele fica como pivô, senão, o elemento de maior valor absoluto é o escolhido. Por enquanto, a escolha do pivô não tem incorporada nenhuma estratégia para reduzir o preenchimento. Uma possível estratégia seria escolher, dentre os elementos que satisfazem (3.3), aquele que tenha menos entradas diferentes de zero à direita, o que diminuiria a criação de novos elementos diferentes de zero na matriz.

3.4 Experimentos numéricos

Tal como se sugere em [60], os problemas considerados consistem em achar $u : [0,1] \times [0,1] \to \mathbb{R}$ tal que

$$G_\lambda = f(s,t) \tag{3.4}$$

com condições de contorno, onde G é um operador que envolve derivadas parciais de segunda ordem de u. Em todos os problemas dividimos o intervalo $[0,1]$ em 64 subintervalos. Conseqüentemente teremos uma malha com $63 \times 63 = 3969$ pontos e as incógnitas do problema discretizado serão os valores de u nos pontos desta malha. Na discretização do problema, todas

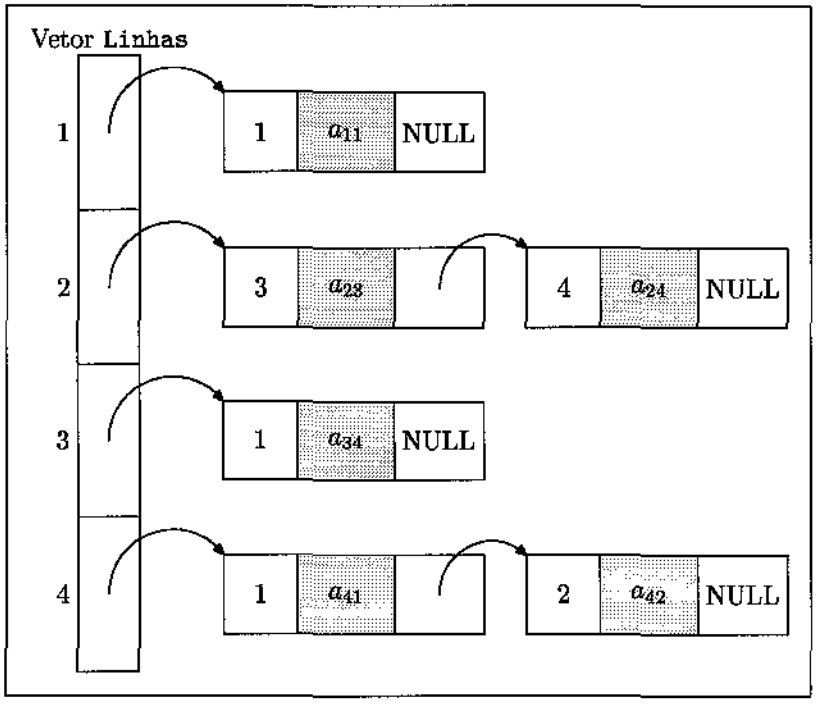


Figura 3.1: Estrutura de dados para matrizes esparsas

as derivadas serão aproximadas utilizando diferenças centrais. Trocando em (3.4) a função e as derivadas pelas suas aproximações, e utilizando as condições de contorno, obtemos um sistema de equações não-lineares onde o número de equações e incógnitas é igual ao número de pontos na malha. Usaremos uma "solução conhecida" do sistema não-linear. Tal solução é

$$u^*(s,t) = 10st(1-s)(1-t)\exp(s^{4.5}).$$

A função $f(s,t)$ é determinada, nos pontos da malha, de tal modo que x^*, a discretização de u^*, seja uma solução exata do sistema $F(x) = 0$. Isto nos permite utilizar o critério de parada artificial

$$|[x_k]_i - [x^*]_i| \leq 10^{-4}. \tag{3.5}$$

Os operadores G_λ considerados neste trabalho são definidos embaixo. Em todos os casos as condições de contorno são $u = 0$ e Δ é o operador Laplaciano em duas dimensões.

Problema de Poisson:

$$G_\lambda(u) = -\Delta u + \lambda \frac{u^3}{1+s^2+t^2}$$

Problema de Bratu:

$$G_\lambda(u) = -\Delta u + \lambda e^u$$

Problema de Convecção-Difusão:

$$G_\lambda(u) = -\Delta u + \lambda u(u_s + u_t)$$

Tentamos resolver os problemas acima definidos com os métodos de Newton e Newton discreto, para diferentes valores de λ e começando com o vetor inicial nulo. Para evitar passos muito grandes (resultantes de instabilidades numéricas ou singularidade do Jacobiano), incorporamos um controle $0 < w \in \mathbb{R}$ no tamanho do incremento s_k (ver [54, 59]) tal que se

$$\|s_k\|_\infty > w$$

então

$$s_k = w\frac{s_k}{\|s_k\|_\infty}.$$

Nos experimentos numéricos utilizamos $w = 10$.

Para a determinação do parâmetro η da fatoração LU realizamos testes comparando o desempenho para diferentes valores de $\eta \in \{1, 10^{-1}, \ldots, 10^{-16}, 0\}$ nos três problemas acima definidos com $\lambda = 0$. Lembremos que o parâmetro η é um parâmetro de compromisso entre estabilidade numérica e custo computacional. Com $\eta = 1$ obtemos pivoteamento parcial puro (o pivô será sempre o elemento de maior módulo na coluna) e com $\eta = 0$ a permutação de linhas só é aceita quando o elemento da diagonal é 0. Neste último caso o pivô escolhido também será o elemento de maior módulo na coluna.

Os algoritmos foram rodados numa SUN SPARCstation-20 com 128 Mbytes de memória RAM com o sistema operacional SunOS Release 4.1.3 U1 e compilados com o compilador g++ de GNU versão 2.7.2 com a opção de otimização do compilador -O4. Nas seguintes tabelas mostramos os resultados obtidos. A nomenclatura é a seguinte:

CX: Convergência para a solução conhecida com o critério de parada (3.5).

CF: A execução terminou porque $\|F(x_k)\|_\infty \leq 10^{-10}$.

INF: O tempo de CPU superou os 400 segundos ou a quantidade de iterações foi maior que 20.

Utilizando $\lambda = 0$ os problemas considerados convertem-se em sistemas de equações lineares que são resolvidos pelo método de Newton numa única iteração. Testamos diferentes valores de η

para os problemas de Bratu, Poisson e Convecção-Difusão. As Tabelas 3.1 e 3.2 mostram o comportamento dos métodos de Newton e Newton discreto, respectivamente. Reportamos o número de iterações (IT) e o tempo de CPU em segundos (Tempo). Estes testes apontaram $\eta = 10^{-10}$ e $\eta = 10^{-5}$ como a escolha adequada nos métodos de Newton e Newton discreto, respectivamente. Observemos que para valores de $\eta \leq 10^{-13}$ a instabilidade numérica fez com que o método de Newton discreto precisasse de mais de uma iteração para convergir à solução do sistema linear. No entanto, o método de Newton convergiu para a solução numa só iteração com todos os valores de η.

As Tabelas 3.3, 3.4 e 3.5 mostram o desempenho do método de Newton discreto com $\eta = 1$ e $\eta = 10^{-5}$ nos problemas de Poisson, Bratu e Convecção-Difusão. O ganho no tempo médio por iteração no método de Newton discreto ao escolher $\eta = 10^{-5}$ foi de 3.33% nos problemas com $\lambda = 0$ e teve um prejuízo de 1.3% na totalidade dos problemas. Isto mostra que, com a estrutura de dados utilizada, a permutação de linhas na fatoração LU influencia muito pouco no custo computacional.

As Tabelas 3.6, 3.7 e 3.8 mostram o desempenho do método de Newton com $\eta = 1$ e $\eta = 10^{-5}$. Neste caso mostramos $\eta = 10^{-5}$ no lugar de $\eta = 10^{-10}$, que foi apontado como a melhor escolha para os problemas com $\lambda = 0$, porque com $\eta = 10^{-10}$ o método de Newton divergiu em muitos problemas nos quais convergiu com $\eta = 1$. Assim, preferimos utilizar um valor de η que diminuísse a instabilidade numérica. Finalmente, o ganho de utilizar $\eta = 10^{-5}$ foi 0.18% comparado à escolha $\eta = 1$.

Na totalidade dos problemas, as quatro combinações testadas chegaram à mesma solução com o mesmo critério de parada e o mesmo número de iterações. Dentre os métodos de Newton e Newton discreto ($\eta = 10^{-5}$) o mais rápido foi o método de Newton com uma diferença de 8.17%.

3.5 Conclusões

De acordo com o indicado em [39], o comportamento dos métodos foi praticamente indistinguível. Comparando os métodos de Newton e Newton discreto (com $\eta = 10^{-5}$) o método que utilizou os valores exatos das derivadas foi, em media, 8.17% mais rápido. Dado que ambos métodos precisaram do mesmo número de iterações, como desprende-se dos testes realizados no Capítulo 2, esta diferença corresponde ao tempo utilizado para o cálculo dos Jacobianos. Tal diferença não foi maior pois a tarefa dominante dos métodos tipo Newton no nosso conjunto de problemas foi a resolução dos sistemas lineares e não a avaliação dos Jacobianos.

η	Bratu		Poisson		Convecção-Difusão	
	IT	Tempo	IT	Tempo	IT	Tempo
1	1	40.45	1	39.77	1	36.57
10^{-1}	1	40.35	1	38.55	1	37.72
10^{-2}	1	41.15	1	41.20	1	37.42
10^{-3}	1	40.45	1	41.90	1	39.35
10^{-4}	1	41.27	1	40.28	1	36.22
10^{-5}	1	41.25	1	42.18	1	36.63
10^{-6}	1	44.90	1	41.62	1	38.38
10^{-7}	1	41.80	1	42.47	1	36.17
10^{-8}	1	39.87	1	41.80	1	36.40
10^{-9}	1	40.90	1	43.05	1	36.45
10^{-10}	1	35.43	1	41.68	1	35.68
10^{-11}	1	40.88	1	43.00	1	36.47
10^{-12}	1	41.23	1	36.35	1	36.70
10^{-13}	1	40.13	1	43.03	1	37.10
10^{-14}	1	40.15	1	41.10	1	36.15
10^{-15}	1	40.80	1	41.83	1	36.93
10^{-16}	1	38.65	1	38.13	1	36.90
0	1	40.78	1	42.15	1	36.72

Tabela 3.1: Escolha do parâmetro η da decomposição LU para o método de Newton.

η	Bratu		Poisson		Convecção-Difusão	
	IT	Tempo	IT	Tempo	IT	Tempo
1	1	41.40	1	39.52	1	41.35
10^{-1}	1	39.78	1	39.12	1	40.20
10^{-2}	1	39.77	1	40.65	1	40.73
10^{-3}	1	40.75	1	39.42	1	41.75
10^{-4}	1	39.57	1	39.92	1	40.50
10^{-5}	1	40.10	1	39.32	1	38.78
10^{-6}	1	42.45	1	39.55	1	40.02
10^{-7}	1	40.15	1	39.17	1	39.58
10^{-8}	1	40.43	1	39.10	1	39.30
10^{-9}	1	40.27	1	39.42	1	38.72
10^{-10}	1	40.72	1	39.87	1	38.62
10^{-11}	1	42.77	1	40.13	1	38.77
10^{-12}	1	42.80	1	39.98	1	38.87
10^{-13}	2	78.33	2	81.30	2	84.78
10^{-14}	2	77.75	2	76.97	2	77.38
10^{-15}	2	77.25	2	78.65	2	74.40
10^{-16}	3	115.42	3	114.33	3	117.83
0	3	116.58	3	120.52	3	117.35

Tabela 3.2: Escolha do parâmetro η da decomposição LU para o método de Newton discreto.

λ	$\eta = 1$			$\eta = 10^{-5}$		
	IT	Tempo	Convergência	IT	Tempo	Convergência
-2000	11	425.80	INF	11	421.65	INF
-1900	11	429.83	INF	11	417.93	INF
-1800	11	439.15	INF	11	440.08	INF
-1700	11	435.10	INF	11	439.23	INF
-1600	11	434.62	INF	10	406.27	INF
-1500	11	434.73	INF	10	409.53	INF
-1400	11	437.52	INF	10	406.97	INF
-1300	11	435.83	INF	11	438.95	INF
-1200	11	424.03	INF	11	437.37	INF
-1100	11	437.68	INF	10	409.02	INF
-1000	11	431.60	INF	10	401.28	INF
-900	11	430.97	INF	11	436.43	INF
-800	11	425.57	INF	11	438.08	INF
-700	11	423.57	INF	11	428.93	INF
-600	10	403.17	INF	11	432.48	INF
-500	11	414.95	INF	11	434.43	INF
-400	11	413.58	INF	11	419.10	INF
-300	11	408.77	INF	11	426.20	INF
-200	11	409.90	INF	11	420.77	INF
-100	4	146.67	CF	4	151.82	CF
0	1	38.33	CX	1	39.27	CX
100	5	186.20	CX	5	189.60	CX
200	6	223.63	CX	6	224.60	CX
300	7	264.35	CX	7	258.23	CX
400	8	293.52	CX	8	306.23	CX
500	8	296.07	CX	8	298.23	CX
600	9	337.55	CX	9	353.33	CX
700	9	329.68	CX	9	350.65	CX
800	9	337.32	CX	9	350.43	CX
900	10	366.58	CX	10	386.33	CX
1000	10	379.25	CX	10	391.00	CX
1100	10	365.48	CX	10	409.63	CX
1200	10	367.70	CX	10	388.08	CX
1300	10	394.33	CX	10	379.45	CX
1400	10	370.67	CX	10	383.23	CX
1500	10	367.60	CX	10	383.60	CX
1600	10	368.38	CX	10	371.27	CX
1700	10	371.28	CX	10	395.27	CX
1800	10	368.22	CX	10	376.20	CX
1900	10	367.85	CX	10	375.12	CX
2000	10	370.23	CX	10	385.45	CX

Tabela 3.3: Newton discreto no problema de Poisson.

λ	$\eta = 1$			$\eta = 10^{-5}$		
	IT	Tempo	Convergência	IT	Tempo	Convergência
-2000	4	157.18	CX	4	153.18	CX
-1900	7	272.70	CX	7	280.97	CX
-1800	4	151.90	CX	4	159.65	CX
-1700	11	432.47	INF	10	401.30	INF
-1600	5	193.25	CX	5	191.67	CX
-1500	5	185.55	CX	5	191.32	CX
-1400	11	422.63	INF	11	435.92	INF
-1300	4	148.12	CX	4	158.50	CX
-1200	11	430.33	INF	11	428.22	INF
-1100	5	184.85	CX	5	180.70	CX
-1000	6	221.63	CX	6	215.57	CX
-900	5	183.08	CX	5	182.35	CX
-800	11	424.97	INF	11	418.37	INF
-700	4	147.98	CX	4	144.17	CX
-600	5	187.45	CX	5	182.92	CX
-500	11	421.88	INF	11	406.57	INF
-400	6	233.02	CX	6	222.33	CX
-300	5	196.33	CX	5	182.63	CX
-200	6	233.63	CX	6	216.32	CX
-100	11	434.17	INF	11	411.07	INF
0	1	45.12	CX	1	38.02	CX
100	4	176.73	CX	4	147.82	CX
200	4	175.62	CX	4	151.98	CX
300	4	178.12	CX	4	154.15	CX
400	4	177.17	CX	4	147.38	CX
500	4	195.68	CX	4	148.27	CX
600	4	180.13	CX	4	147.12	CX
700	4	181.40	CX	4	151.32	CX
800	4	151.37	CX	4	145.57	CX
900	4	148.93	CX	4	144.40	CX
1000	4	158.60	CX	4	146.48	CX
1100	4	155.62	CX	4	145.88	CX
1200	4	159.90	CX	4	149.62	CX
1300	4	154.63	CX	4	158.37	CX
1400	4	151.95	CX	4	150.67	CX
1500	4	146.50	CX	4	152.05	CX
1600	4	147.00	CX	4	145.73	CX
1700	4	155.77	CX	4	150.82	CX
1800	4	150.62	CX	4	156.13	CX
1900	4	151.52	CX	4	149.67	CX
2000	4	149.18	CX	4	153.23	CX

Tabela 3.4: Newton discreto no problema de Bratu.

λ	$\eta = 1$			$\eta = 10^{-5}$		
	IT	Tempo	Convergência	IT	Tempo	Convergência
-100	11	424.22	INF	10	401.90	INF
-90	11	428.55	INF	11	439.27	INF
-80	10	378.28	CX	10	399.65	CX
-70	9	344.63	CX	9	352.25	CX
-60	9	338.48	CX	9	360.92	CX
-50	8	301.40	CX	8	314.85	CX
-40	6	237.27	CX	6	237.22	CX
-30	5	194.08	CX	5	205.20	CX
-20	4	162.17	CX	4	161.68	CX
-10	3	116.58	CX	3	125.90	CX
0	1	40.05	CX	1	43.27	CX
10	3	116.23	CX	3	122.10	CX
20	4	154.50	CX	4	167.13	CX
30	5	188.40	CX	5	201.32	CX
40	6	228.27	CX	6	249.13	CX
50	7	286.62	CX	7	299.58	CX
60	8	304.00	CX	8	334.12	CX
70	9	339.30	CX	9	370.63	CX
80	9	343.45	CX	9	396.55	CX
90	10	393.23	CX	10	408.00	CX
100	10	409.08	INF	10	424.12	INF

Tabela 3.5: Newton discreto no problema de Convecção-Difusão.

λ	$\eta = 1$			$\eta = 10^{-5}$		
	IT	Tempo	Convergência	IT	Tempo	Convergência
-2000	11	405.80	INF	11	420.72	INF
-1900	11	412.02	INF	11	423.35	INF
-1800	11	411.77	INF	11	420.10	INF
-1700	11	418.37	INF	11	417.68	INF
-1600	11	403.40	INF	11	430.05	INF
-1500	11	411.10	INF	11	419.48	INF
-1400	11	410.58	INF	11	426.52	INF
-1300	11	416.90	INF	11	420.45	INF
-1200	11	402.20	INF	11	404.85	INF
-1100	11	407.48	INF	11	409.62	INF
-1000	11	404.52	INF	11	416.35	INF
-900	11	406.33	INF	11	406.15	INF
-800	11	403.55	INF	11	410.00	INF
-700	11	400.80	INF	12	432.33	INF
-600	11	413.05	INF	11	404.52	INF
-500	12	423.23	INF	12	431.88	INF
-400	12	423.75	INF	12	433.10	INF
-300	12	422.95	INF	12	433.28	INF
-200	12	437.08	INF	12	427.30	INF
-100	4	143.02	CF	4	136.82	CF
0	1	36.07	CX	1	35.22	CX
100	5	175.93	CX	5	173.23	CX
200	6	214.03	CX	6	208.22	CX
300	7	248.05	CX	7	245.85	CX
400	8	281.25	CX	8	279.37	CX
500	8	288.82	CX	8	279.33	CX
600	9	324.98	CX	9	318.18	CX
700	9	322.12	CX	9	313.45	CX
800	9	321.95	CX	9	319.78	CX
900	10	360.22	CX	10	357.92	CX
1000	10	356.23	CX	10	366.27	CX
1100	10	358.92	CX	10	352.48	CX
1200	10	364.77	CX	10	355.92	CX
1300	10	363.07	CX	10	369.25	CX
1400	10	355.18	CX	10	380.37	CX
1500	10	359.58	CX	10	368.88	CX
1600	10	351.28	CX	10	374.37	CX
1700	10	355.80	CX	10	376.35	CX
1800	10	359.23	CX	10	366.70	CX
1900	10	360.08	CX	10	368.97	CX
2000	10	360.47	CX	10	368.05	CX

Tabela 3.6: Newton no problema de Poisson.

λ	$\eta = 1$			$\eta = 10^{-5}$		
	IT	Tempo	Convergência	IT	Tempo	Convergência
-2000	4	147.10	CX	4	144.35	CX
-1900	7	244.77	CX	7	245.17	CX
-1800	4	140.68	CX	4	136.32	CX
-1700	12	430.32	INF	12	429.58	INF
-1600	5	175.77	CX	5	173.62	CX
-1500	5	175.33	CX	5	184.52	CX
-1400	12	436.27	INF	11	402.27	INF
-1300	4	138.93	CX	4	145.92	CX
-1200	11	400.13	INF	12	436.17	INF
-1100	5	177.27	CX	5	175.83	CX
-1000	6	208.65	CX	6	210.78	CX
-900	5	179.45	CX	5	173.52	CX
-800	12	433.07	INF	11	404.40	INF
-700	4	139.47	CX	4	138.28	CX
-600	5	176.18	CX	5	176.90	CX
-500	12	432.63	INF	12	434.70	INF
-400	6	209.97	CX	6	206.93	CX
-300	5	174.70	CX	5	173.68	CX
-200	6	214.35	CX	6	212.47	CX
-100	12	422.90	INF	12	414.80	INF
0	1	35.70	CX	1	34.17	CX
100	4	143.48	CX	4	137.38	CX
200	4	139.67	CX	4	139.77	CX
300	4	139.28	CX	4	141.52	CX
400	4	140.47	CX	4	136.85	CX
500	4	145.43	CX	4	136.78	CX
600	4	142.32	CX	4	138.22	CX
700	4	143.85	CX	4	140.43	CX
800	4	140.37	CX	4	137.65	CX
900	4	141.72	CX	4	137.82	CX
1000	4	145.97	CX	4	140.00	CX
1100	4	143.92	CX	4	140.30	CX
1200	4	141.60	CX	4	143.83	CX
1300	4	142.02	CX	4	143.77	CX
1400	4	141.68	CX	4	138.32	CX
1500	4	142.57	CX	4	139.50	CX
1600	4	141.20	CX	4	136.12	CX
1700	4	143.58	CX	4	136.70	CX
1800	4	147.83	CX	4	135.95	CX
1900	4	139.52	CX	4	138.43	CX
2000	4	140.92	CX	4	135.82	CX

Tabela 3.7: Newton no problema de Bratu.

λ	$\eta = 1$			$\eta = 10^{-5}$		
	IT	Tempo	Convergência	IT	Tempo	Convergência
-100	11	409.15	INF	11	411.93	INF
-90	12	428.10	INF	12	425.27	INF
-80	10	356.52	CX	10	361.62	CX
-70	9	319.57	CX	9	329.87	CX
-60	9	319.82	CX	9	328.85	CX
-50	8	284.23	CX	8	284.90	CX
-40	6	216.35	CX	6	215.30	CX
-30	5	177.75	CX	5	179.13	CX
-20	4	141.92	CX	4	142.73	CX
-10	3	106.37	CX	3	110.90	CX
0	1	35.55	CX	1	35.43	CX
10	3	106.25	CX	3	106.67	CX
20	4	142.30	CX	4	139.78	CX
30	5	175.87	CX	5	174.63	CX
40	6	211.87	CX	6	208.70	CX
50	7	250.13	CX	7	249.23	CX
60	8	283.77	CX	8	282.87	CX
70	9	318.47	CX	9	315.98	CX
80	9	326.00	CX	9	315.35	CX
90	10	365.58	CX	10	351.20	CX
100	12	433.45	INF	12	424.57	INF

Tabela 3.8: Newton no problema de Convecção-Difusão.

Com a estrutura de dados utilizada para armazenar matrizes esparsas, e neste conjunto de problemas, o parâmetro η é irrelevante. Assim nos inclinamos por utilizar um pivoteamento parcial puro que garanta maior estabilidade. Com respeito ao tratamento de matrizes esparsas, embora as estruturas dinâmicas utilizadas foram escolhidas, principalmente, pela sua simplicidade, mostraram-se muito eficientes nos problemas testados. Além da sua simplicidade, a escolha de memória dinâmica tem ao menos duas características a serem ressaltadas: (i) não utiliza mais espaço do necessário para armazenar os elementos não-nulos da matriz e (ii) não requer nenhum tipo de previsão para a reserva de espaço para um possível preenchimento na fatoração [54]. Porém, tem uma crítica a ser feita. Em geral, a utilização de memória dinâmica introduz um *overhead* considerável no custo total do processo. Nas Conclusões finais deste trabalho serão feitos alguns comentários a respeito, não só das estruturas dinâmicas deste capítulo, mas também dos grafos computacionais apresentados no Capítulo 2.

Capítulo 4

Estimação das Constantes Óticas e da Espessura de um Filme Fino Utilizando Minimização Irrestrita

4.1 Introdução

O problema consiste em achar a espessura d, o índice de refração $r(\lambda)$ e o coeficiente de atenuação $\kappa(\lambda)$ de um filme fino, utilizando só dados de transmissão. Emite-se luz com diferentes comprimentos de onda λ_i e mede-se a luz transmitida T_i^{meas} do outro lado do filme. Com um conjunto razoável de medições (λ_i, T_i^{meas}) tentamos descobrir as constantes óticas acima mencionadas. A transmissão ótica fornece informação precisa no intervalo do espectro onde o material vai da opacidade completa a algum grau de transparência [24, 71]. Algumas soluções aproximadas têm sido encontradas em casos onde a transmissão mostra um padrão de interferência numa região altamente transparente do espectro [83, 109, 110]. Porém, até agora, a solução geral do problema não tem sido satisfatória porque o sistema de equações é altamente indeterminado. Recentemente, foi reportado um método novo, baseado num enfoque de minimização "ponto a ponto" com restrições, que permite resolver o caso geral [29, 28]. O método define um problema de programação não-linear, cujas variáveis são os coeficientes a serem estimados, com restrições que representam conhecimentos prévios da solução física. O novo método foi bem sucedido na recuperação de d, $r(\lambda)$ e $\kappa(\lambda)$ a partir de diferentes espectros de transmissão de filmes artificiais e reais [29, 28]. A maior dificuldade do enfoque do problema de otimização ponto a ponto com

restrições [29, 28] é que é um problema de grande porte relativamente complexo de programação não-linear com restrições lineares cuja solução só pode ser obtida por meio de códigos sofisticados, e não sempre disponíveis, que trabalhem eficientemente com a esparsidade da matriz de restrições [91, 92].

Consideramos o problema de estimar o coeficiente de atenuação, o índice de refração e a espessura do filme, utilizando só dados de transmissão. Dado o comprimento de onda λ, o índice de refração do substrato s, e as incógnitas d (espessura), $r(\lambda)$ (índice de refração) e $\kappa(\lambda)$ (coeficiente de atenuação), a transmissão teórica é dada por uma fórmula conhecida [71, 109, 110]. Tendo medidas da transmissão em (bastantes) comprimentos de onda diferentes, queremos estimar as incógnitas acima mencionadas. Numa primeira olhada, este problema é altamente indeterminado. Para cada comprimento de onda, a equação

$$\text{Transmissão teórica} = \text{Transmissão medida} \tag{4.1}$$

tem três incógnitas $d, r(\lambda), \kappa(\lambda)$ e só d repete-se para todos os valores de λ. A idéia principal em [29, 28] foi incorporar conhecimentos prévios das funções $r(\lambda)$ e $\kappa(\lambda)$ para diminuir o grau de liberdade de (4.1) ao ponto que só estimativas com significado físico sejam aceitas.

A idéia de supor fórmulas fechadas para r e κ dependendo de uns poucos parâmetros tem sido reportada em [71, 109, 110]. Os métodos originados nessas idéias são eficientes quando a curva de transmissão tem um padrão de interferência em regiões relativamente grandes do espectro onde $\kappa(\lambda)$ é quase nula. Em outros casos, a satisfação de (4.1) é muito grosseira ou as curvas $r(\lambda)$ e $\kappa(\lambda)$ são fisicamente inaceitáveis.

Em [29, 28], no lugar de impor formas funcionais para $r(\lambda)$ e $\kappa(\lambda)$, as restrições fenomenológicas que restringem a variabilidade dessas funções foram consideradas explicitamente de forma tal que o problema de estimação tomou a forma:

$$\begin{gathered} \textit{Minimizar} \sum_{\lambda} [\text{Transmissão teórica}(\lambda) - \text{Transmissão medida}(\lambda)]^2 \\ \text{sujeito a } \textbf{Restrições Físicas.} \end{gathered} \tag{4.2}$$

Desta forma, funções $r(\lambda)$ e $\kappa(\lambda)$ bem comportadas podem ser obtidas sem restrições fortes que poderiam danificar a qualidade do ajuste (4.1).

A maior contribuição deste trabalho é estabelecer um método para resolver o problema de estimação onde (4.2) é substituído por um problema de otimização sem restrições. Resolvemos este problema utilizando o Método do Gradiente Espectral (SGM) introduzido recentemente por Raydan [101]. Este método baseia-se numa idéia muito efetiva para problemas de minimização

irrestrita de grande porte e consiste em utilizar só direções de gradiente com comprimentos de passo que garantem uma convergência rápida. A redução de (4.2) a um problema de minimização sem restrições precisou do cálculo de derivadas complicadas que não teria sido possível sem a utilização de técnicas de diferenciação automática. Aqui utilizamos o modo reverso de diferenciação automática descrito no Capítulo 2.

4.2 Formulação irrestrita do problema de estimação

A transmissão T de um filme fino absorvente depositado num substrato transparente grosso (ver [109, 110]) é dado por:

$$T = \frac{Ax}{B - Cx + Dx^2}, \tag{4.3}$$

onde

$$A = 16s(r^2 + \kappa^2), \tag{4.4}$$

$$B = [(r+1)^2 + \kappa^2][(r+1)(r+s^2) + \kappa^2], \tag{4.5}$$

$$\begin{aligned} C = {} & [(r^2 - 1 + \kappa^2)(r^2 - s^2 + \kappa^2) - 2\kappa^2(s^2+1)]2\cos\varphi \\ & -\kappa[2(r^2 - s^2 + \kappa^2) + (s^2+1)(r^2 - 1 + \kappa^2)]2\sin\varphi, \end{aligned} \tag{4.6}$$

$$D = [(r-1)^2 + \kappa^2][(r-1)(r-s^2) + \kappa^2], \tag{4.7}$$

$$\varphi = 4\pi r d/\lambda, \quad x = exp(-\alpha d), \quad \alpha = 4\pi\kappa/\lambda. \tag{4.8}$$

Nas fórmulas (4.4)–(4.8) é utilizada a seguinte notação:

(a) λ é o comprimento de onda;

(b) $s = s(\lambda)$ é o índice de refração do substrato transparente (que supomos conhecido);

(c) $r = r(\lambda)$ é o índice de refração do filme;

(d) $\kappa = \kappa(\lambda)$ é o índice de atenuação do filme (α é o coeficiente de absorção);

(e) d é a espessura do filme.

Dado um conjunto de dados experimentais $(\lambda_i, T^{meas}(\lambda_i))$, $\lambda_{min} \leq \lambda_i < \lambda_{i+1} \leq \lambda_{max}$, para $i = 1, \ldots, N$, queremos estimar d, $r(\lambda)$ e $\kappa(\lambda)$. Este problema parece altamente indeterminado. De fato, para d conhecido e um λ dado, a seguinte equação deve ser satisfeita:

$$T(\lambda, s(\lambda), d, r(\lambda), \kappa(\lambda)) = T^{meas}(\lambda). \tag{4.9}$$

Esta equação tem duas incógnitas $r(\lambda)$ e $\kappa(\lambda)$ e, portanto, em geral, o seu conjunto de soluções é uma curva no espaço bidimensional $(r(\lambda), \kappa(\lambda))$. Portanto, o conjunto de funções (r, κ) que satisfazem (4.9) para um dado d é infinito e, informalmente, é representado por uma variedade não-linear de dimensão N em $\mathbb{R}^{2N}$.

Porém, restrições físicas reduzem drasticamente o conjunto viável das incógnitas $r(\lambda)$ e $\kappa(\lambda)$. Por exemplo, na vizinhança do eixo fundamental de absorção (dispersão normal), essas restrições físicas são:

PC1: $r(\lambda) \geq 1$, $\kappa(\lambda) \geq 0$ para todo $\lambda \in [\lambda_{min}, \lambda_{max}]$;

PC2: $r(\lambda)$ e $\kappa(\lambda)$ são funções decrescentes de λ;

PC3: $r(\lambda)$ é convexa;

PC4: existe $\lambda_{infl} \in [\lambda_{min}, \lambda_{max}]$ tal que $\kappa(\lambda)$ é convexa se $\lambda \geq \lambda_{infl}$ e côncava se $\lambda < \lambda_{infl}$.

Observemos que **PC1** é satisfeita supondo **PC2** válida, $r(\lambda_{max}) \geq 1$ e $\kappa(\lambda_{max}) \geq 0$. As restrições **PC2**, **PC3** e **PC4** podem ser escritas, respectivamente, como

$$n'(\lambda) \leq 0 \text{ e } \quad \kappa'(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{max}]; \tag{4.10}$$

$$n''(\lambda) \geq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{max}]; \tag{4.11}$$

$$\kappa''(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{infl}] \tag{4.12}$$

e

$$\kappa''(\lambda) \geq 0 \text{ para todo } \lambda \in [\lambda_{infl}, \lambda_{max}]. \tag{4.13}$$

Claramente, as restrições

$$n''(\lambda) \geq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{max}] \text{ e } n'(\lambda_{max}) \leq 0$$

implicam

$$n'(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{max}].$$

Mais ainda,

$$\kappa''(\lambda) \geq 0 \text{ para todo } \lambda \in [\lambda_{infl}, \lambda_{max}] \text{ e } \kappa'(\lambda_{max}) \leq 0$$

implicam

$$\kappa'(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{infl}, \lambda_{max}].$$

Finalmente,

$$\kappa''(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{infl}] \text{ e } \kappa'(\lambda_{min}) \leq 0$$

implicam

$$\kappa'(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{infl}].$$

Portanto, **PC2** pode ser substituída por

$$n'(\lambda_{max}) \leq 0, \quad \kappa'(\lambda_{max}) \leq 0, \quad \kappa'(\lambda_{min}) \leq 0. \tag{4.14}$$

Resumindo, as restrições **PC1–PC4** serão satisfeitas se, e somente se,

$$r(\lambda_{max}) \geq 1, \quad \kappa(\lambda_{max}) \geq 0, \tag{4.15}$$

$$n'(\lambda_{max}) \leq 0, \quad \kappa'(\lambda_{max}) \leq 0, \tag{4.16}$$

$$n''(\lambda) \geq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{max}], \tag{4.17}$$

$$\kappa''(\lambda) \geq 0 \text{ para todo } \lambda \in [\lambda_{infl}, \lambda_{max}], \tag{4.18}$$

$$\kappa''(\lambda) \leq 0 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{infl}], \tag{4.19}$$

$$\kappa'(\lambda_{min}) \leq 0. \tag{4.20}$$

Logo, a solução de quadrados mínimos contínuos do problema de estimação é a solução $(d, r(\lambda), \kappa(\lambda))$ de

$$\text{Minimizar} \int_{\lambda_{min}}^{\lambda_{max}} |T(\lambda, s(\lambda), d, r(\lambda), \kappa(\lambda)) - T^{meas}(\lambda)|^2 d\lambda \tag{4.21}$$

sujeito às restrições (4.15)–(4.20).

Nossa idéia neste trabalho é eliminar as restrições do problema, tanto quanto seja possível, por meio de mudança de variáveis. Informalmente, vamos escrever a função objetivo (4.21) dependendo das derivadas segundas de $r(\lambda)$ e $\kappa(\lambda)$ e os valores das funções e suas derivadas primeiras em λ_{max}. Além disso, a não-negatividade será garantida expressando as variáveis como quadrados de variáveis auxiliares. De fato, escrevemos

$$r(\lambda_{max}) = 1 + u^2, \quad \kappa(\lambda_{max}) = v^2, \tag{4.22}$$

$$n'(\lambda_{max}) = -u_1^2, \quad \kappa'(\lambda_{max}) = -v_1^2, \tag{4.23}$$

$$n''(\lambda) = w(\lambda)^2 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{max}], \tag{4.24}$$

$$\kappa''(\lambda) = z(\lambda)^2 \text{ para todo } \lambda \in [\lambda_{infl}, \lambda_{max}], \tag{4.25}$$

$$\kappa''(\lambda) = -z(\lambda)^2 \text{ para todo } \lambda \in [\lambda_{min}, \lambda_{infl}]. \tag{4.26}$$

Neste ponto, para evitar uma enfadonha formulação contínua do problema, consideramos a situação real, na qual os dados são representados por um conjunto de N pontos igualmente espaçados no intervalo $[\lambda_{min}, \lambda_{max}]$. Portanto, definimos

$$h = (\lambda_{max} - \lambda_{min})/(N-1),$$

$$\lambda_i = \lambda_{min} + (i-1)h, \quad i = 1, \ldots, N.$$

A transmissão medida em λ_i será chamada T_i^{meas}. Usaremos a notação r_i, κ_i, w_i, z_i para os estimadores por diferenças finitas de $r(\lambda_i)$, $\kappa(\lambda_i)$, $w(\lambda_i)$ e $z(\lambda_i)$:

$$r_i \approx r(\lambda_i), \quad \kappa_i \approx \kappa(\lambda_i),$$

$$w_i \approx w(\lambda_{i+1}), \quad z_i \approx z(\lambda_{i+1}),$$

para $i = 1, \ldots, N$. A discretização das relações diferenciais (4.22–4.26) nos leva a:

$$r_N = 1 + u^2, \quad v_N = v^2, \tag{4.27}$$

$$r_{N-1} = r_N + u_1^2 h, \quad \kappa_{N-1} = \kappa_N + v_1^2 h, \tag{4.28}$$

$$r_i = w_i^2 h^2 + 2n_{i+1} - r_{i+2}, \quad i = 1, \ldots, N-2, \tag{4.29}$$

$$\kappa_i = z_i^2 h^2 + 2\kappa_{i+1} - \kappa_{i+2}, \quad \text{se } \lambda_{i+1} \geq \lambda_{infl}, \tag{4.30}$$

$$\kappa_i = -z_i^2 h^2 + 2\kappa_{i+1} - \kappa_{i+2}, \quad \text{se } \lambda_{i+1} < \lambda_{infl}. \tag{4.31}$$

Finalmente a função objetivo (4.21) é aproximada por uma soma de quadrados, levando-nos ao problema de otimização

$$\text{Minimizar} \sum_{i=1}^{N} [T(\lambda_i, s(\lambda_i), d, r_i, \kappa_i) - T_i^{meas}]^2 \tag{4.32}$$

sujeito a

$$\kappa_1 \geq \kappa_2 \tag{4.33}$$

Como r_i and κ_i dependem de u, u_1, v, v_1, w, z e λ_{infl} por (4.27–4.31), o problema (4.32) toma a forma

$$\text{Minimizar } f(d, \lambda_{infl}, u, u_1, v, v_1, w_1, \ldots, w_{N-2}, z_1, \ldots, z_{N-2}) \tag{4.34}$$

sujeito a (4.33).

Como esperamos que a restrição (4.33) não seja ativa na solução de (4.34–4.33), vamos considerar o problema irrestrito (4.34). As variáveis que aparecem em (4.34) têm natureza diferente. A espessura d é uma variável dimensional (medida em nanômetros em nossos problemas reais) que pode ser determinada utilizando as observações $T^{meas}(\lambda_i)$ para $\lambda_i \geq \lambda_{bound}$, onde λ_{bound}, um limitante superior de λ_{infl}, reflete nosso conhecimento prévio do problema. Por essa razão, nosso primeiro passo no processo de estimação será estimar d com os dados que correspondem a $\lambda_i \geq \lambda_{bound}$. Para atingir este objetivo resolvemos o problema

$$\text{Minimizar } \bar{f}(u, u_1, v, v_1, w, z) \equiv \sum_{\lambda_i \geq \lambda_{bound}} [T(\lambda_i, s(\lambda_i), d, r_i, \kappa_i) - T_i^{meas}]^2 \tag{4.35}$$

para diferentes valores de d e tomamos como espessura estimada aquela que corresponda ao menor valor de função. Neste caso, a restrição (4.33) é irrelevante porque é automaticamente satisfeita pela convexidade de κ e o fato de que a derivada primeira de κ em λ_{max} é não-positiva. Daqui em diante consideramos que d está fixa como resultado do procedimento anterior.

O segundo passo consiste em determinar λ_{infl}, junto com u, u_1, v, v_1, w, z. Com este propósito, observemos que, dados d e λ_{infl} o problema

$$\text{Minimizar } \sum_{i=1}^{N} [T(\lambda_i, s(\lambda_i), d, r_i, \kappa_i) - T_i^{meas}]^2 \tag{4.36}$$

é (desprezando (4.33)) um problema de minimização irrestrita cujas variáveis são u, u_1, v, v_1, w, z ($2N$ variáveis). Resolvemos esse problema para vários valores de λ_{infl} e tomamos como estimativas de r e κ a combinação de variáveis correspondente ao menor valor da função objetivo. Para minimizar esta função e resolver (4.35) para diferentes espessuras, utilizamos o algoritmo de minimização irrestrita descrito na próxima seção.

4.3 Descrição do algoritmo de minimização irrestrita

Como vimos na seção anterior, os problemas de minimização irrestrita (4.35) e (4.36) têm a forma

$$\text{Minimizar } f(u, u_1, v, v_1, w_1, \ldots, w_{N-2}, z_1, \ldots, z_{N-2}). \tag{4.37}$$

Para simplificar a notação, nesta seção vamos a escrever

$$x = (u, u_1, v, v_1, w_1, \ldots, w_{N-2}, z_1, \ldots, z_{N-2}).$$

As derivadas parciais de f são usualmente necessárias em algoritmos de otimização porque fornecem a informação de primeira ordem da função objetivo que permite seguir trajetórias de descida. Neste caso, as derivadas são difíceis de calcular e por esta razão utilizamos o modo reverso de diferenciação automática para o cálculo das derivadas. Veja o Capítulo 2 para mais detalhes.

Em princípio, qualquer algoritmo de otimização irrestrita pode ser utilizado para resolver (4.37) (veja [39, 55]). Como o problema tem, potencialmente, um número grande de variáveis, nossa escolha deve se restringir aos métodos que sejam capazes de encarar essa situação. Um trabalho recente de Raydan [101] nos levou a escolher o Método do Gradiente Espectral (SGM), uma implementação do método de Barzilai and Borwein para quadráticas introduzido em [101]. De fato, Raydan mostrou, utilizando um conhecido conjunto de problemas teste clássicos, que SGM superou métodos de gradientes conjugados (veja [51, 55]) em problemas de otimização irrestrita de grande porte. O método do gradiente espectral de Raydan é fácil de implementar, um fato que contribuiu em nossa decisão porque permite-nos ficar independentes de "caixas pretas" como *softwares* fechados. Nossa descrição do SGM é essencialmente, a mesma apresentada por Raydan com uma pequena diferença na escolha do passo α_k quando $b_k \leq 0$.

Denotamos $g(x) = \nabla f(x)$. O algoritmo começa com $x_0 \in \mathbb{R}^n$ e utiliza um inteiro $M \geq 0$, um parâmetro pequeno α_{min}, um parâmetro grande $\alpha_{max} > \alpha_{min}$, um parâmetro de decréscimo suficiente $\gamma \in (0,1)$ e parâmetros de salvaguarda $0 < \sigma_1, < \sigma_2 < 1$. Inicialmente, $\alpha_0 \in [\alpha_{min}, \alpha_{max}]$ é arbitrário. Dado $x_k \in \mathbb{R}^n$ e $\alpha_k \in [\alpha_{min}, \alpha_{max}]$, o Algoritmo 3.1 descreve como obter x_{k+1} e α_{k+1}, e quando terminar o processo.

Algoritmo 3.1

Passo 1: *Detectar se o ponto atual é estacionário.*

Se $\|g(x_k))\| = 0$, parar declarando que x_k é estacionário.

Passo 2: *Busca linear.*

Passo 2.1: Fazer $\lambda \leftarrow \alpha_k$.

Passo 2.2: Fazer $x_+ = x_k - \lambda g(x_k)$.

Passo 2.3: Se

$$f(x_+) \leq \max_{0 \leq j \leq \min\{k, M-1\}} f(x_{k-j}) + \gamma \langle x_+ - x_k, g(x_k) \rangle, \tag{4.38}$$

então definir $x_{k+1} = x_+$, $s_k = x_{k+1} - x_k$ e $y_k = g(x_{k+1}) - g(x_k)$.

Senão, definir

$$\lambda_{new} \in [\sigma_1\lambda, \sigma_2\lambda], \tag{4.39}$$

fazer $\lambda \leftarrow \lambda_{new}$ e voltar ao Passo 2.2.

Passo 3: *Calcular o comprimento de passo espectral.*

Calcular $b_k = \langle s_k, y_k \rangle$.

Se $b_k \leq 0$, fazer $\alpha_{k+1} = \alpha_{max}$,

senão, calcular $a_k = -\langle s_k, g(x_k) \rangle$ e

$$\alpha_{k+1} = \min \{\alpha_{max}, \max \{\alpha_{min}, a_k/b_k\}\}.$$

Na prática, o cálculo de λ_{new} utiliza uma interpolação quadrática unidimensional com as salvaguardas (4.39).

4.4 Resultados numéricos

Para testar a confiabilidade do novo enfoque de minimização irrestrita, utilizamos filmes artificiais (criados no computador) depositados em substratos de vidro ou silício cristalino. Nas simulações, o índice de refração do vidro $s_{glass}(\lambda)$ é dado por:

$$s_{glass}(\lambda) = \sqrt{1 + 1/(0.76194 - 7940/\lambda^2)},$$

e o índice de refração do substrato de silício $s_{Si}(\lambda)$ é dado por:

$$s_{Si}(\lambda) = 3.71382 - 8.6912310^{-5}\lambda - 2.4712510^{-8}\lambda^2 + 1.0467710^{-11}\lambda^3.$$

Em todas as simulações supomos que o comprimento de onda e a espessura são medidos em nanômetros. A transmissão $T^{meas}(\lambda)$ para cada filme foi gerada no intervalo $\lambda \in [\lambda_{min}, \lambda_{max}]$ utilizando espessura d_{true}, índice de refração $r_{true}(\lambda)$ e coeficiente de absorção $\alpha_{true}(\lambda)$ conhecidos. Para considerar situações realísticas, incluindo a falta de precisão experimental, consideramos cálculos alternativos de $T^{meas}(\lambda)$, onde a transmissão real (gerada) foi arredondada para 4, 3 e 2 casas decimais. Também realizamos experimentos numéricos utilizando diferentes números de pontos de transmissão: 100, 50 e 25. A descrição dos filmes gerados e os correspondentes resultados numéricos são apresentados abaixo.

Filme A: Este filme simula um filme fino de germânio amorfo com $d_{true} = 118$nm depositado num substrato de vidro. $\lambda_{min} = 600$nm e $\lambda_{max} = 2000$nm.

Filme B: Este filme é idêntico ao Filme A com $d_{true} = 782$nm. $\lambda_{min} = 1000$nm e $\lambda_{max} = 2000$nm.

Filme C: Este filme simula um filme fino de germânio amorfo com $d_{true} = 147$nm depositado em um substrato cristalino de silício. $\lambda_{min} = 1250$nm e $\lambda_{max} = 2500$nm.

Filme D: Este filme é idêntico ao Filme C com $d_{true} = 640$nm. $\lambda_{min} = 640$nm e $\lambda_{max} = 1250$nm.

Filme E: Este filme simula um filme fino de silício amorfo hidrogenado com $d_{true} = 624$nm depositado em vidro. $\lambda_{min} = 600$nm e $\lambda_{max} = 1600$nm.

Para os nossos cálculos necessitamos estimativas iniciais de $\kappa(\lambda)$ e $r(\lambda)$. Como estimativa inicial de $\kappa(\lambda)$ utilizamos uma função linear por partes cujos valores são 0.1 em λ_{min}, 0.01 em $\lambda_{min} + 0.2(\lambda_{max} - \lambda_{min})$ e 10^{-10} em λ_{max}. A estimativa inicial de $r(\lambda)$ foi uma função linear cujos valores extremos variam entre 5 e 3 com passo 1 (esses valores foram escolhidos pelo nosso conhecimento prévio dos materiais simulados). Como excluímos as funções constantes que mostraram em testes preliminares levar o método a mínimos locais, temos três possibilidades para a estimativa inicial de $r(\lambda)$: as funções lineares decrescentes definidas pelos pares de pontos $[(\lambda_{min}, 4); (\lambda_{max}, 3)]$, $[(\lambda_{min}, 5); (\lambda_{max}, 3)]$ e $[(\lambda_{min}, 5); (\lambda_{max}, 4)]$.

O esquema geral para obter os parâmetros ótimos destes filmes é como se segue. Primeiro, dividimos o espectro em duas partes: $[\lambda_{min}, \lambda_{bound}]$ e $[\lambda_{bound}, \lambda_{max}]$, onde λ_{bound} é um limitante conhecido de λ_{infl}. Em todos os casos utilizamos $\lambda_{bound} = (\lambda_{min} + \lambda_{max})/2$. Para estimar a espessura utilizamos os dados com abscissa em $[\lambda_{bound}, \lambda_{max}]$. O procedimento consiste em rodar o Algoritmo 3.1 para diferentes valores de d entre $d_{min} = \frac{1}{2}d_{kick}$ e $d_{max} = \frac{3}{2}d_{kick}$ com passo 10 $(d_{min}, d_{min} + 10, d_{min} + 20, \ldots)$, onde d_{kick} pode ser uma estimativa inicial grosseira de d. Desta forma obtemos d_{trial}, o valor da espessura que fornece o menor erro quadrático. Depois, repetimos o procedimento com $d_{min} = d_{trial} - 10$, $d_{max} = d_{trial} + 10$ e passo 1 para obter, finalmente, a espessura estimada d_{best}.

Para estimar o ponto de inflexão procedemos em forma análoga, utilizando o espectro completo, a espessura fixa em d_{best} e tentando diferentes pontos de inflexão (obviamente entre λ_{min} e λ_{bound}). Tomamos como estimativa de λ_{bound} o valor que nos fornece o menor erro quadrático. Em todas as simulações, permitimos só 3000 iterações do Algoritmo 3.1. O passo final do método consiste em, fixando d_{best} e λ_{infl}, rodar o Algoritmo 3.1 mais uma vez permitindo 30000 iterações.

Todos os experimentos foram simulados numa SPARCstation Sun Ultra 1, com um processador UltraSPARC de 64 bits, *clock* de 167 MHz e 128 MBytes de memória RAM. Utilizamos a

		25		50		100	
		r	α	r	α	r	α
2	E_{ph-min}	0.1136	1.9021×10^{-4}	0.1063	4.3521×10^{-4}	0.1139	4.3108×10^{-4}
	E_{ph-max}	0.8742	4.4408×10^{-3}	1.4727	4.9223×10^{-3}	0.4761	1.9139×10^{-3}
3	E_{ph-min}	0.0442	2.7240×10^{-4}	0.1317	6.5932×10^{-4}	0.2249	1.1136×10^{-3}
	E_{ph-max}	1.3505	4.4571×10^{-3}	0.2298	1.4237×10^{-3}	0.1278	1.0697×10^{-3}
4	E_{ph-min}	0.0552	3.3624×10^{-4}	0.1093	4.0392×10^{-4}	0.1103	4.3715×10^{-4}
	E_{ph-max}	1.3631	4.5240×10^{-3}	0.1418	1.0016×10^{-3}	0.1149	3.8185×10^{-4}
todos	E_{ph-min}	0.0358	3.9590×10^{-4}	0.0749	3.0955×10^{-4}	0.0184	1.6248×10^{-4}
	E_{ph-max}	1.4558	4.5031×10^{-3}	0.2367	7.5514×10^{-4}	0.0117	1.9621×10^{-4}

Tabela 4.1: **Filme A**: Erros quadráticos nos índices de refração e coeficientes de absorção estimados variando a precisão e o número de pontos de transmissão medida utilizados.

linguagem C/C++ com o compilador g++ (GNU project C and C++ compiler v2.7) e a opção de otimização do compilador -O4. Apesar das muitas execuções do algoritmo de minimização irrestrita que são necessárias para resolver o problema, o tempo total de CPU utilizado para o processo completo nunca excedeu os 10 minutos na plataforma acima mencionada.

A Tabela 4.1 corresponde só ao filme A. Ela mostra a precisão obtida em $r(\lambda)$ e $\alpha(\lambda)$ utilizando 25, 50 e 100 pontos de transmissão medida, e arredondando os dados de transmissão para 2, 3 e 4 casas decimais e, finalmente, sem arredondar. Os erros reportados são os máximos valores de $\mid r(\lambda) - r_{true}(\lambda) \mid$ e $\mid \alpha(\lambda) - \alpha_{true}(\lambda) \mid$ para as regiões espectrais de alta e baixa energia do fóton, respectivamente. A Tabela 4.2 corresponde ao mesmo filme. Ela mostra a espessura estimada com 25, 50 e 100 pontos de dados e para diferentes números de casas decimais em $T^{meas}(\lambda)$. A Tabela 4.3 mostra, para todos os filmes, as espessuras estimadas e os erros quadráticos obtidos no processo de minimização.

As Figuras 4.1–4.5 são auto-explicáveis. As linhas contínuas representam as transmissões verdadeiras, os índices de refração verdadeiros e os coeficientes de absorção verdadeiros para os cinco filmes considerados. Os círculos abertos representam as melhores estimativas (utilizando todas as casas decimais). Notar que as funções $r(\lambda)$ e $\alpha(\lambda)$ estão representadas em função da energia do fóton $(1240/\lambda)$. Finalmente, as Figuras 4.6, 4.7 e 4.8 mostram como funciona o processo de estimar a espessura para cada filme.

4.5 Conclusões

A análise dos resultados numéricos permite-nos esboçar as seguintes conclusões:

	25	50	100
2	121	121	121
3	119	122	124
4	119	121	121
todos	118	119	119

Tabela 4.2: **Filme A**: espessura estimada variando a precisão da transmissão e o número de pontos de transmissão medidos.

Filme	Espessura verdadeira	Espessura estimada	Erro quadrático
A	118	119	6.9296×10^{-7}
B	782	782	2.2031×10^{-7}
C	147	152	6.2249×10^{-6}
D	640	639	1.3653×10^{-6}
E	624	624	2.1210×10^{-7}

Tabela 4.3: Espessuras verdadeiras e estimadas e erros quadráticos.

1. O procedimento proposto é altamente confiável para estimar a espessura verdadeira em todos os filmes quando 4 (ou todos) dígitos dos dados de transmissão são utilizados. O método fornece uma recuperação muito boa da transmissão verdadeira nos casos onde métodos não aproximados são úteis, *i.e.*, filmes muito finos ou camadas absorventes.

2. A precisão dos dados da transmissão "medida" tem efeito na precisão das estimações de $r(\lambda)$ mas é quase irrelevante para a estimação de $\alpha(\lambda)$. Em situações reais, utilizando um espectrofotômetro moderno, as transmissões podem ser obtidas com 3 ou 4 casas decimais. Neste caso, e utilizando 100 pontos medidos de transmissão, o erro na estimação de $r(\lambda)$ é próximo de 0.11 (para o **Filme A**). As diferenças entre os resultados obtidos com 100 e 50 pontos não são significativas.

3. Na maioria dos casos, o erro quadrático em função da espessura estimada (Figuras 4.6, 4.7 e 4.8) é uma função com vários minimizadores locais não-globais. Portanto, a estratégia de separar a variável d das outras variáveis do problema de otimização parece ser correta, dado que tende a evitar convergências espúrias a estes minimizadores locais.

4. A comparação dos presentes resultados com os obtidos previamente, utilizando o algoritmo descrito em [29, 28], parece confirmar que o novo método é, no mínimo, tão eficiente quanto

o enfoque prévio de minimização com restrições. Além disso, o *software* resultante é mais portável e fácil de manipular.

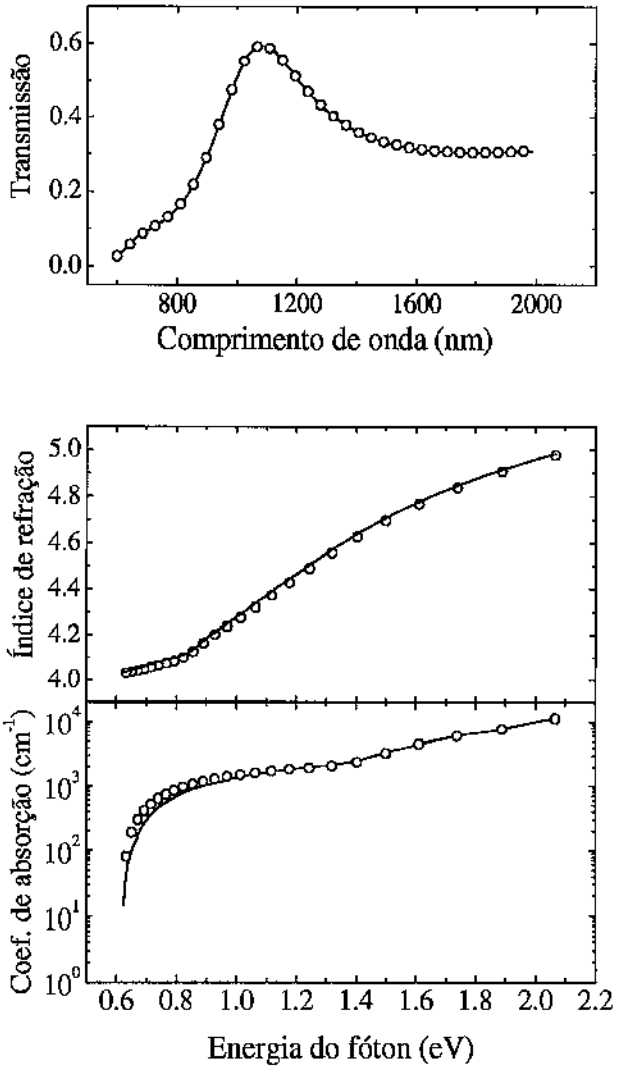


Figura 4.1: Valores "verdadeiros" e estimados da transmissão, o índice de refração e o coeficiente de absorção de um filme muito fino gerado numericamente com espessura $d = 118$nm que simula uma capa de a-Ge depositada sobre vidro. Note a boa qualidade das constantes óticas e a transmissão encontradas.

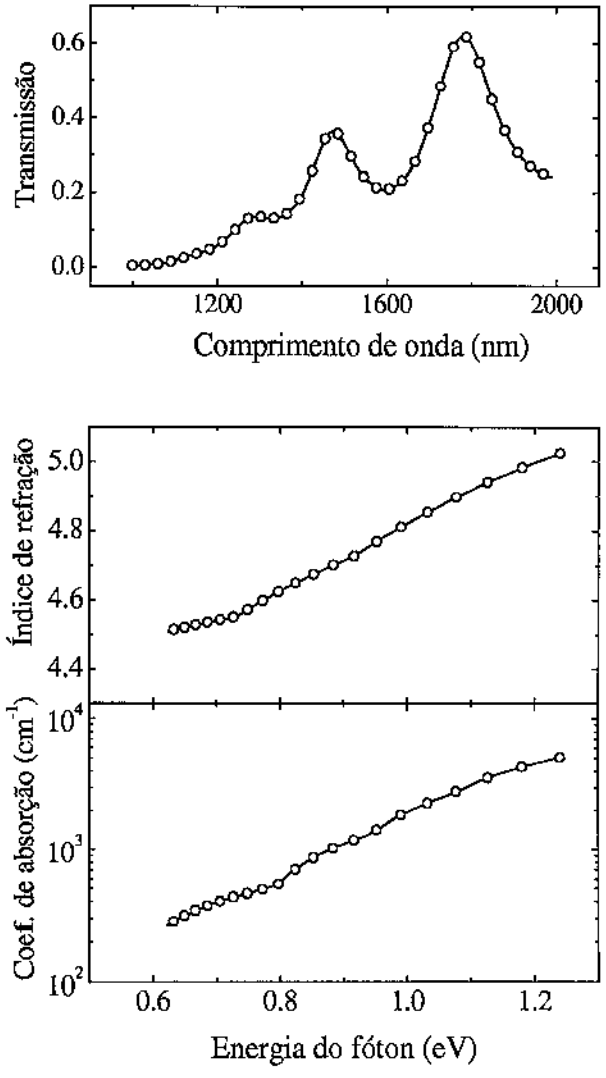


Figura 4.2: Valores "verdadeiros" e estimados da transmissão, o índice de refração e o coeficiente de absorção de um filme gerado numericamente com espessura $d = 782$nm simulando uma capa de a-Ge depositada sobre vidro. Note a boa qualidade das constantes óticas e a transmissão encontradas.

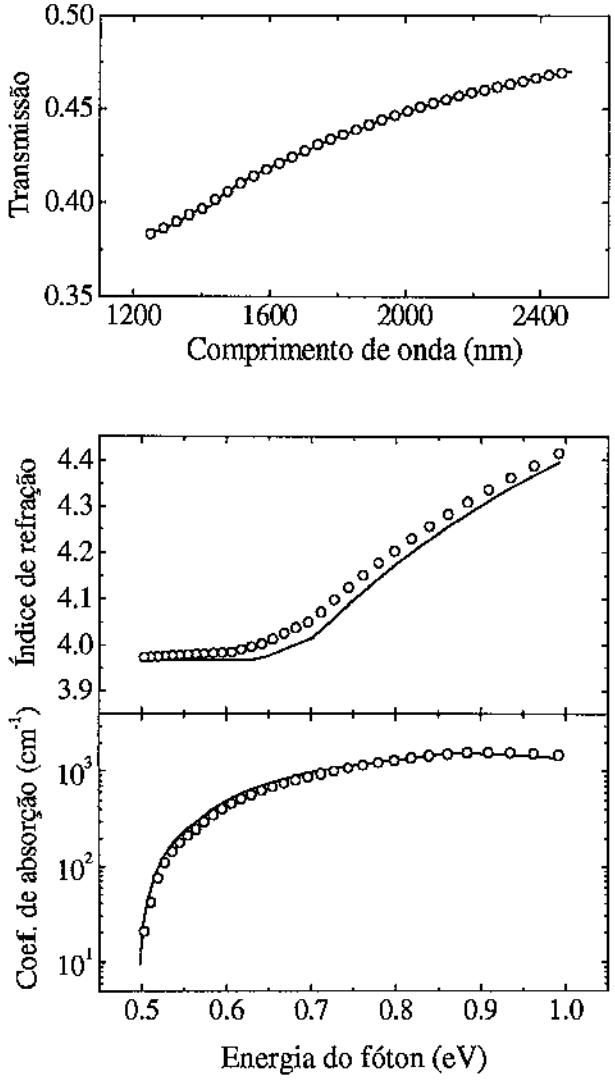


Figura 4.3: Valores "verdadeiros" e estimados da transmissão, o índice de refração e o coeficiente de absorção de um filme muito fino gerado numericamente com espessura $d = 147$nm que simula uma capa de a-Ge depositada num substrato de silício cristalino. Note a boa qualidade das constantes óticas e a transmissão encontradas apesar da suavidade e falta de interferências da transmissão verdadeira.

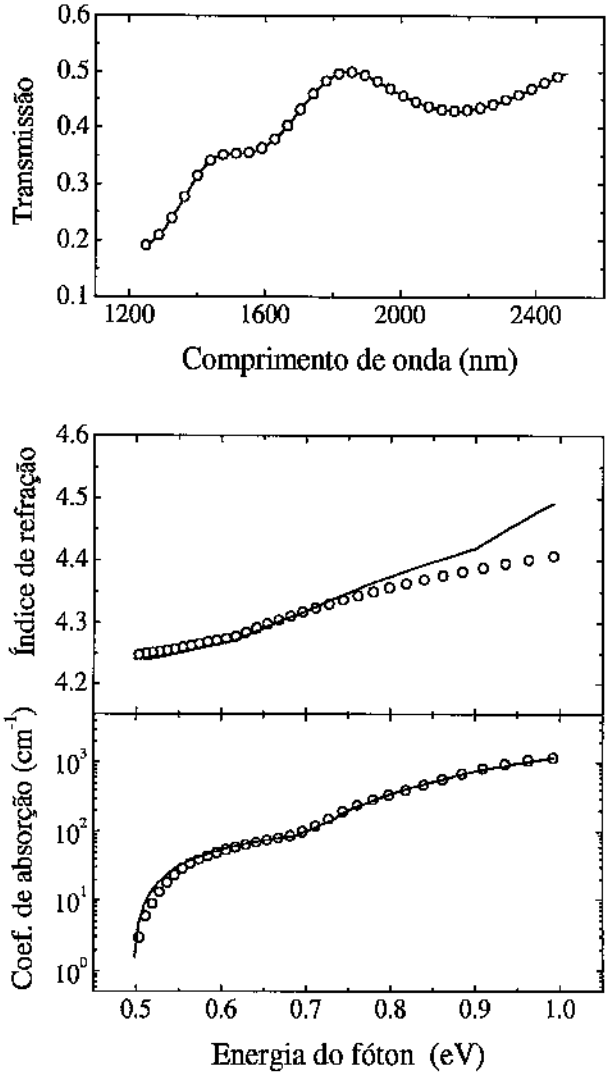


Figura 4.4: Valores "verdadeiros" e estimados da transmissão, o índice de refração e o coeficiente de absorção de um filme fino gerado numericamente com espessura $d = 640$nm simulando uma capa de a-Ge depositada num substrato de silício cristalino. Note a boa qualidade das constantes óticas e a transmissão encontradas. A recuperação do "verdadeiro" índice de refração mostra alguns defeitos na região de altas energias do fóton.

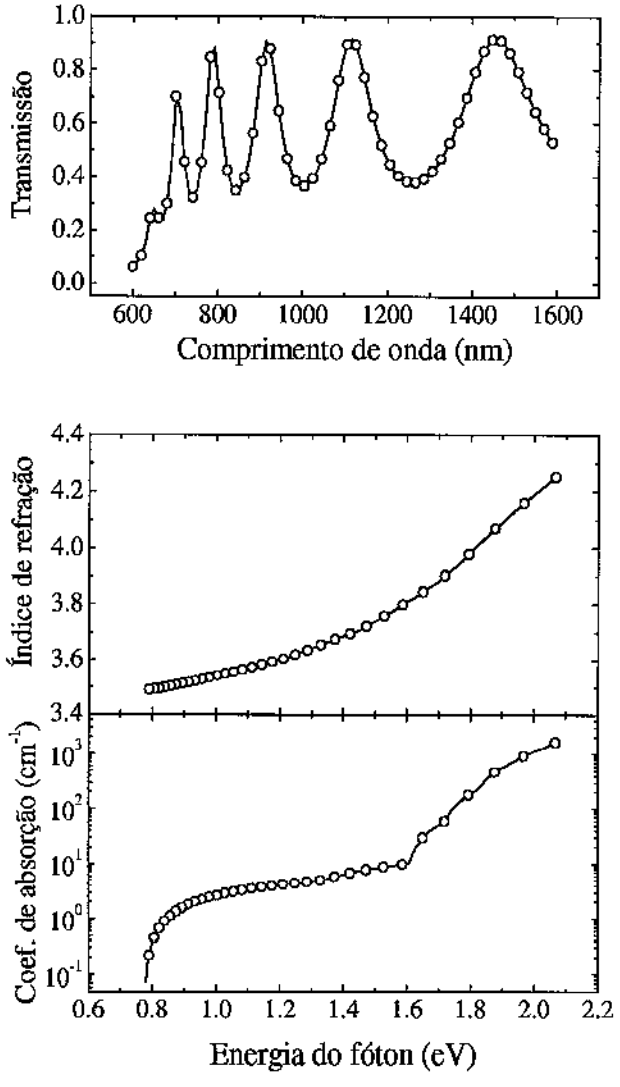


Figura 4.5: Valores "verdadeiros" e estimados da transmissão, o índice de refração e o coeficiente de absorção de um filme fino gerado numericamente com espessura $d = 624$nm simulando uma capa de a-Si depositada sobre vidro. A coincidência das constantes óticas e da transmissão é muito boa.

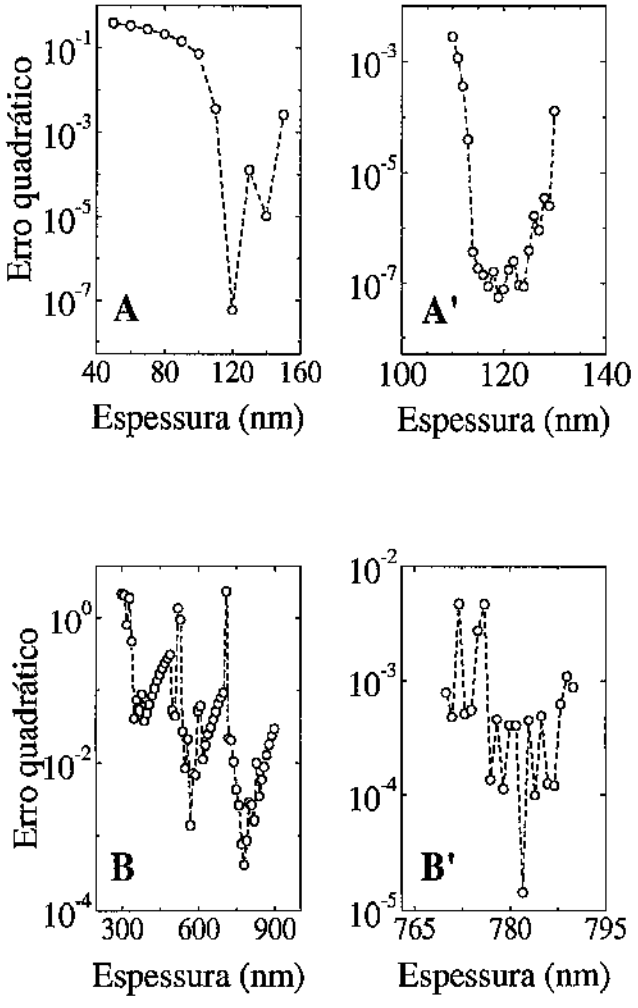


Figura 4.6: Erro quadrático do processo de minimização em função das espessuras para os Filmes A e B. Do lado esquerdo o passo é 10nm enquanto do lado direito o passo refinado é 1nm. Note os minimizadores locais.

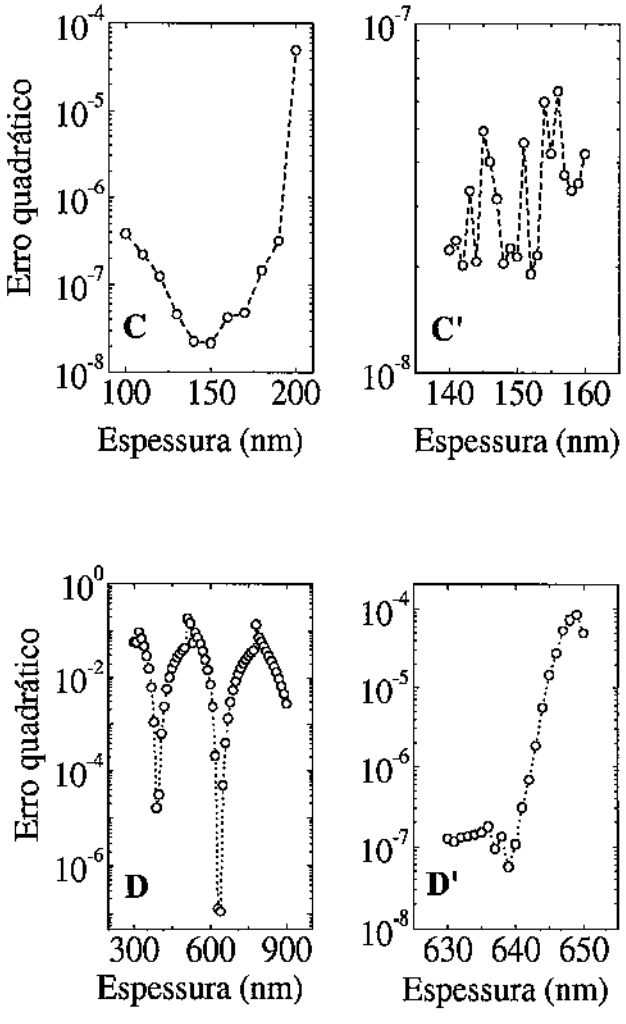


Figura 4.7: Erro quadrático do processo de minimização em função das espessuras para os Filmes C e D. Do lado esquerdo o passo é 10nm enquanto do lado direito o passo refinado é 1nm. Note os minimizadores locais.

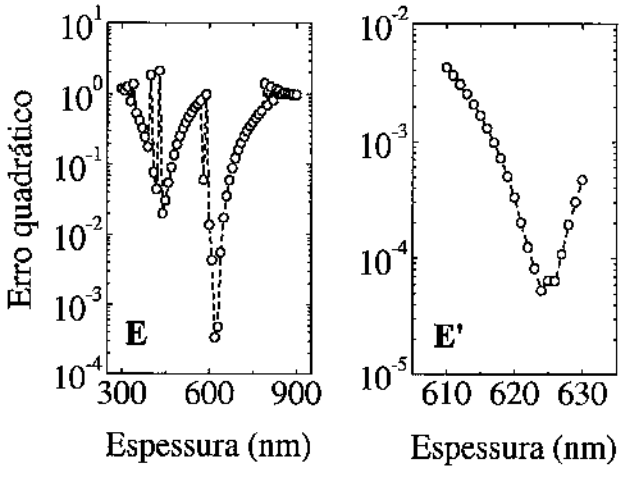


Figura 4.8: Erro quadrático do processo de minimização em função das espessuras para o Filme E. Do lado esquerdo o passo é 10nm enquanto do lado direito o passo refinado é 1nm. Note os minimizadores locais.

Capítulo 5

Métodos Não-monótonos de Gradientes Espectrais Projetados para Conjuntos Convexos

5.1 Introdução

Neste capítulo consideramos um método de gradientes projetados para a minimização de uma função diferenciável em conjuntos fechados e convexos não-vazios. Nas últimas décadas, houve muitas variações diferentes do método de gradientes projetados que podem ser consideradas como as extensões restritas do método do gradiente otimal para minimização sem restrições. Todas elas têm em comum a propriedade de manter a factibilidade dos iterandos projetando-os no conjunto viável. Este processo é em geral a parte mais custosa de qualquer método de gradientes projetados. Mais ainda, se projetar não é custoso, como no caso de restrições de caixa, o método é considerado tão lento quanto o seu análogo para minimização sem restrições, o método do gradiente otimal (também conhecido como "método de máxima descida"). O lado positivo é que o método de gradientes projetados é relativamente simples de implementar e muito efetivo para problemas de grande porte.

Os fatos acima citados motivaram-nos a combinar o método de gradientes projetados com dois ingredientes recentemente desenvolvidos em otimização. Primeiro, estendemos as estratégias típicas de globalização associadas com esses métodos aos esquemas de busca linear não-monótonos desenvolvidos por Grippo, Lampariello e Lucidi [70] para métodos do tipo Newton. Segundo,

propomos associar o comprimento espectral do passo, introduzido por Barzilai e Borwein [5] e analisado por Raydan [100]. Esta escolha do comprimento do passo tem um baixo custo computacional e aumenta surpreendentemente a velocidade de convergência do método do gradiente. De fato, enquanto o método dos gradientes espectrais parece ser uma generalização do método do gradiente, é claro pela sua construção que está relacionado com a família de métodos Quase-Newton através de uma equação secante aproximada. A diferença fundamental é que este é um método de dois pontos enquanto o método de máxima descida não. A idéia principal por detrás da escolha espectral do comprimento do passo é que o método de máxima descida é muito lento, mas pode ser acelerado tomando comprimentos de passo que, no lugar de derivar da minimização da função ao longo do gradiente do iterando atual, derivem da minimização unidimensional do passo anterior. Veja Glunt *et al.* [56] para uma relação com o método das potências com deslocamentos para aproximar autovalores e autovetores, e também para uma interessante aplicação química. Veja também Raydan [101] para uma combinação da escolha espectral do comprimento do passo com técnicas não-monótonas de busca linear para resolver problemas de minimização irrestrita.

Assim, é natural e relativamente simples transportar a idéia do gradiente espectral com buscas lineares não-monótonas para o caso do gradiente projetado para acelerar a sua convergência. Em particular, neste trabalho estendemos a versão de [6] que exige uma condição do tipo Armijo sobre o caminho curvilíneo das projeções. Esta versão prática está baseada na versão original proposta por Goldstein [57] e Levitin e Polak [80]. Também aplicamos estes novos ingredientes ao caminho contínuo projetado factível que será propriamente definido na Seção 5.2.

As propriedades de convergência do método de gradientes projetados para diferentes escolhas de passo têm sido amplamente estudadas. Veja [6, 7, 27, 43, 57, 80, 90, 106] e outros autores. Veja [27] para uma interessante revisão dos diferentes resultados de convergência. Para uma revisão completa veja Dunn [44].

Na Seção 5.2 deste trabalho definimos os algoritmos de gradientes espectrais projetados e comentamos os resultados de convergência global. Na Seção 5.3 apresentamos experimentos numéricos. Este conjunto de experimentos mostra que, de fato, a escolha espectral do comprimento do passo representa um progresso considerável em relação às escolhas constantes e que a estratégia não-monótona é muito útil. Algumas considerações finais são apresentadas na Seção 5.4. Em particular, elaboramos a respeito da relação entre o método de gradientes espectrais e a família de métodos Quase-Newton.

5.2 Algoritmos de gradientes projetados não-monótonos

Os algoritmos de gradientes projetados não-monótonos introduzidos nesta seção aplicam-se a problemas da forma

$$\text{Minimizar } f(x) \quad \text{sujeito a} \quad x \in \Omega,$$

onde Ω é um conjunto fechado e convexo em $\mathbb{R}^n$. Ao longo deste trabalho assumimos que f está definida e tem derivadas parciais contínuas num conjunto aberto que contém Ω. $\|\cdot\|$ denota a norma euclidiana de vetores e matrizes embora em alguns casos possa ser substituída por uma norma arbitrária.

Dado $z \in \mathbb{R}^n$ definimos $P(z)$ como a projeção ortogonal de z em Ω. Denotamos $g(x) = \nabla f(x)$. Os algoritmos começam com $x_0 \in \Omega$ e utilizam um inteiro $M \geq 1$, um parâmetro pequeno $\alpha_{min} > 0$, um parâmetro grande $\alpha_{max} > \alpha_{min}$, um parâmetro de decréscimo suficiente $\gamma \in (0,1)$ e parâmetros de salvaguarda $0 < \sigma_1 < \sigma_2 < 1$. Inicialmente, $\alpha_0 \in [\alpha_{min}, \alpha_{max}]$ é arbitrário. Dados $x_k \in \Omega$ e $\alpha_k \in [\alpha_{min}, \alpha_{max}]$, os Algoritmos 2.1 e 2.2 descrevem como obter x_{k+1} e α_{k+1} e quando terminar o processo.

Algoritmo 2.1

Passo 1: *Detectar se o ponto atual é estacionário.*

Se $\|P(x_k - g(x_k)) - x_k\| = 0$, parar declarando que x_k é estacionário.

Passo 2: *Busca linear.*

Passo 2.1: Fazer $\lambda \leftarrow \alpha_k$.

Passo 2.2: Fazer $x_+ = P(x_k - \lambda g(x_k))$.

Passo 2.3: Se

$$f(x_+) \leq \max_{0 \leq j \leq \min\{k, M-1\}} f(x_{k-j}) + \gamma \langle x_+ - x_k, g(x_k) \rangle, \tag{5.1}$$

então definir $\lambda_k = \lambda$, $x_{k+1} = x_+$, $s_k = x_{k+1} - x_k$ e $y_k = g(x_{k+1}) - g(x_k)$.

Senão, definir

$$\lambda_{new} \in [\sigma_1 \lambda, \sigma_2 \lambda], \tag{5.2}$$

fazer $\lambda \leftarrow \lambda_{new}$ e voltar ao Passo 2.2.

Passo 3: *Calcular o comprimento de passo espectral.*

Calcular $b_k = \langle s_k, y_k \rangle$.

Se $b_k \leq 0$, fazer $\alpha_{k+1} = \alpha_{max}$,

senão, calcular $a_k = \langle s_k, s_k \rangle$ e

$$\alpha_{k+1} = \min \{\alpha_{max}, \max \{\alpha_{min}, a_k/b_k\}\}.$$

A busca unidimensional do Algoritmo 2.1 (chamado SPG1 daqui em diante) leva em consideração pontos da forma $P(x_k - \lambda g(x_k))$, para $\lambda \in (0, \alpha_k]$ os quais, em geral, formam um caminho curvilíneo (linear por partes se Ω é um conjunto poliédrico). Por essa razão, o produto escalar $\langle x_+ - x_k, g(x_k) \rangle$, na condição de Armijo não-monótona (5.1), tem que ser calculado para cada ponto x_+. Mais ainda, na formulação de SPG1 a distância entre duas tentativas consecutivas pode ser muito pequena ou nula, o que representa uma perda de informação. Esta observação motivou-nos a construir o Algoritmo 2.2. Este algoritmo também está baseado na direção do gradiente espectral projetado $P(x_k - \alpha_k g(x_k)) - x_k$, sendo α_k a "inversa do quociente de Rayleigh $\frac{\langle s_{k-1}, s_{k-1} \rangle}{\langle s_{k-1}, y_{k-1} \rangle}$ com salvaguardas. Observe que $\frac{\langle s_{k-1}, y_{k-1} \rangle}{\langle s_{k-1}, s_{k-1} \rangle}$ é de fato o quociente de Rayleigh correspondente à matriz Hessiana média $\int_0^1 \nabla^2 f(x_{k-1} + t s_{k-1}) dt$. Porém, no caso de rejeição do primeiro ponto, o próximo ponto é calculado ao longo da mesma direção. Como conseqüência, $\langle x_+ - x_k, g(x_k) \rangle$ deve ser calculado só no primeiro ponto e a operação de projeção deve ser feita só uma vez por iteração. Portanto, o Algoritmo 2.2, que será chamado SPG2 daqui em diante, coincide com SPG1 exceto na busca linear que é dada em continuação.

Algoritmo 2.2:

Passo 2: *Busca linear*

Passo 2.1: Calcular $d_k = P(x_k - \alpha_k g(x_k)) - x_k$.

Fazer $\lambda \leftarrow 1$.

Passo 2.2: Fazer $x_+ = x_k + \lambda d_k$.

Passo 2.3: Se

$$f(x_+) \leq \max_{0 \leq j \leq \min\{k, M-1\}} f(x_{k-j}) + \gamma \lambda \langle d_k, g(x_k) \rangle, \tag{5.3}$$

então definir $\lambda_k = \lambda$, $x_{k+1} = x_+$, $s_k = x_{k+1} - x_k$ e $y_k = g(x_{k+1}) - g(x_k)$,

senão, definir λ_{new} como em (5.2), fazer $\lambda \leftarrow \lambda_{new}$ e

voltar ao Passo 2.2.

Nos dois algoritmos, o cálculo de λ_{new} utiliza uma interpolação quadrática unidimensional com as salvaguardas (5.2). Notar também que as condições (5.1) e (5.3) da busca linear garantem que a seqüência $\{x_k\}$ mantém-se em $\Omega_0 \equiv \{x \in \Omega : f(x) \leq f(x_0)\}$.

Ambos algoritmos estão bem definidos e têm a propriedade de que todo ponto de acumulação $\bar{x}$ de uma seqüência $\{x_k\}$ gerada pelos Algoritmos 2.1 ou 2.2 é um ponto estacionário restrito, *i.e.*,

$$\langle g(\bar{x}), x - \bar{x} \rangle \geq 0 \text{ para todo } x \in \Omega.$$

A demonstração destes resultados será omitida neste trabalho e o leitor interessado pode-se remeter a [10].

5.3 Resultados numéricos

Os algoritmos SPG1 e SPG2 introduzidos na seção anterior calculam pelo menos uma projeção por iteração no conjunto factível Ω. Portanto, estamos especialmente interessados nos casos nos quais essas projeções são fáceis de calcular. Uma situação importante na qual tais projeções são triviais é quando Ω é uma caixa n-dimensional, possivelmente com alguns limitantes iguais a infinito. De fato, bons algoritmos para minimização em caixas são essenciais para o desenvolvimento de métodos de Lagrangeano aumentado para programação não-linear em geral (veja [34, 35]). Com essa idéia em mente, implementamos os algoritmos de gradientes espectrais projetados para o caso no qual Ω é descrito por limitantes nas variáveis. Para mostrar a eficácia dos novos algoritmos, os testamos comparando-os com um algoritmo para minimização de problemas de grande porte com restrições de caixa baseado em regiões de confiança e na abordagem dos métodos tipo Newton truncado.

TNBOX [52] é um método iterativo para resolver problemas com restrições de caixa que, a cada iteração, aproxima a função objetivo por uma função quadrática e minimiza (aproximadamente) este modelo na caixa determinada pelas restrições originais e a caixa auxiliar que representa a região onde a aproximação quadrática é confiável (região de confiança). Se a função objetivo é suficientemente reduzida no minimizador (aproximado) da função quadrática, o ponto correspondente é aceito como um novo iterando. De outro modo, a região de confiança é reduzida. A principal diferença entre TNBOX e o método utilizado no bem conhecido pacote LANCELOT [34] é que em [52] a função quadrática é explorada em toda a intersecção das restrições originais e a região de confiança, enquanto que em [34] só a face determinada por um "ponto de Cauchy aproximado" é explorada. Uma comparação entre estes dois métodos para minimização com

restrições de caixa foi mostrada em [40], revelando que seus comportamentos são muito similares.

O método utilizado para resolver o subproblema quadrático (chamado QUACAN daqui em diante) visita as diferentes faces do domínio utilizando direções conjugadas no interior de cada face e "gradientes chopados" como direções de busca para deixar as faces. Veja [9, 53, 52]. A cada iteração deste subproblema, um produto matriz-vetor entre a aproximação da Hessiana e um vetor tem que ser calculado. Na abordagem dos métodos do tipo Newton truncado, cada produto Hessiana-vetor é substituído por um quociente incremental do gradiente da função objetivo ao longo da direção dada pelo vetor (ver [93]). O código TNBOX tem vários parâmetros envolvendo precisões relativas aos subproblemas, estratégias para atualizar o raio da região de confiança, estratégias internas de QUACAN para deixar as faces e assim por diante. Nos nossos testes utilizamos os parâmetros *default* recomendados no trabalho experimental [40].

As funções-teste são as mesmas utilizadas por Raydan em [101], utilizamos os mesmos pontos iniciais e introduzimos limitantes da forma $\ell_i \leq [x]_i \leq u_i$ com $\ell_i = \ell_1, u_i = u_1$ para todo $i = 1, \ldots, n$ de acordo com a Tabela 5.1.

Todos os experimentos foram rodados numa SPARCstation Sun Ultra 1, com um processador UltraSPARC de 64 bits, um *clock* de 167 MHz e 128 MBytes de memória RAM. O código de TNBOX está em Fortran e foi compilado com o compilador f77 (SC 1.0 Fortran v1.4) e os códigos do SPG estão na linguagem C/C++ e foram compilados com o compilador g++ (GNU project C and C++ compiler v2.7). Em ambos casos utilizamos a opção de otimização do compilador -O4.

Para os métodos SPG utilizamos $\gamma = 10^{-4}$, $\alpha_{min} = 10^{-30}$, $\alpha_{max} = 10^{30}$, $\sigma_1 = 0.1$, $\sigma_2 = 0.9$ e $\alpha_0 = 1/\|g_1(x_0)\|$. Para escolher o parâmetro M testamos o problema 11 com 100 variáveis e sem restrições. Este problema foi reportado em [101] como um problema difícil e mal condicionado. Escolhemos $M = 18$ dentro do intervalo $[10, 20]$ pois ele produziu os melhores resultados para o problema testado. Para decidir quando parar a execução dos algoritmos declarando convergência, rodamos primeiro TNBOX com o critério $\|g_P(x_k)\| \leq 10^{-5}$ e definimos $\bar{x} = x_k$. Depois rodamos os algoritmos SPG com o critério de convergência $\|g_P(x_k)\| \leq \max\{10^{-10}, \|g_P(\bar{x})\|\}$. Também paramos a execução dos algoritmos depois de 7000 iterações sem satisfazer o critério de convergência.

Para completar os resultados numéricos, rodamos o Algoritmo de Gradientes Projetados (PGA) que é idêntico a SPG1 com $\alpha_k \equiv 1$ como escolha inicial do comprimento do passo. Nesse caso implementamos as versões monótona e não-monótona do PGA, que correspondem a $M = 1$ e $M = 18$.

O comportamento de TNBOX no conjunto de problemas testados é mostrado na Tabela 5.2.

Problema	Nome	Dimensão n	Limitantes inferiores	Limitantes superiores
1	Strictly convex 1	100	−10	10
1		1000	0.5	∞
1		10000	−∞	0.5
2	Strictly convex 2	100	−10	10
2		500	−∞	0.5
2		1000	0.5	100
3	Brown almost linear	100	−10	10
3		1000	0.75	∞
3		10000	0	0.9
4	Trigonometric	100	−10	5e−03
4		1000	1e−05	∞
4		10000	−10	10
5	Broyden tridiagonal	100	−10	10
5		1000	−0.8	∞
5		3000	−∞	−0.6
6	Oren's power	100	−10	10
6		1000	0.3	∞
6		10000	−∞	0.7
7	Extended Rosenbrock	100	−0.5	∞
7		1000	−1000	0.5
7		10000	0.5	1000
8	Penalty 1	100	4	∞
8		1000	0	100
8		10000	−10	10
9	Tridiagonal 1	100	−∞	100
9		1000	0	1
10	Variably dimensioned	100	0	∞
10		1000	−∞	6
11	Extended Powell	100	0	∞
11		1000	−∞	0
12	Generalized Rosenbrock	100	−1	500
12		500	−∞	0.5
13	Extended ENGLV1	100	−∞	0.8
13		1000	0	1
13		10000	−8000	∞
14	Extended Freudenstein and Roth	100	−∞	0
14		1000	−10	10
14		10000	0	∞
15	Wrong Extended Wood	100	−1000	1000
15		1000	0	∞

Tabela 5.1: Os limitantes

Nas Tabelas 5.3 e 5.4 mostramos o comportamento dos Algoritmos de Gradientes Projetados, enquanto que nas Tabelas 5.5 e 5.6 mostramos o desempenho de SPG1 e SPG2 respectivamente.

Reportamos o número de iterações (IT), o número de avaliações de função (FE), o número de avaliações de gradiente (GE), o número de buscas lineares (LS) (exceto no caso de TNBOX), o tempo de CPU em segundos (Tempo), o valor final da função objetivo ($f(x)$) e a norma euclideana do gradiente contínuo projetado no iterando final ($\|g_P(x)\|$).

Claramente, o desempenho do Algoritmo de Gradientes Projetados (Tabelas 5.3 e 5.4) é muito pior do que o desempenho dos outros métodos considerados neste trabalho. O desempenho deste método é reportado para mostrar a evidência de que os benefícios dos algoritmos SPG devem-se à escolha espectral do comprimento do passo e não às bem conhecidas características do gradiente projetado. Por isso, daqui em diante nos concentraremos na comparação entre TNBOX, SPG1 e SPG2.

Como conseqüência do critério de parada adotado, a $\|g_P(x)\|$ encontrada pelos métodos SPG é menor que a encontrada por TNBOX em todos os experimentos, exceto no problema 10 com $n = 1000$, no qual SPG2 não satisfez, em 7000 iterações, a precisão imposta por TNBOX. Neste caso, relaxamos o critério de parada para $\|g_P(x)\| \leq 10^{-5}$ e SPG2 terminou com os seguintes resultados:

Problema	n	IT	FE	GE	LS	Tempo	$f(x)$	$\|g_P(x)\|$
10	1000	1619	3022	1620	406	8,05	1,9476e−11	8,8263e−06

Em 39 dos 40 problemas, todos os métodos encontraram soluções com pequenas diferenças no valor da função objetivo. A única exceção foi o problema 15 com $n = 100$. Neste caso, SPG2 achou a melhor solução com $f(x_*) = 5.7 \times 10^{-17}$ enquanto SPG1 e TNBOX acharam pontos estacionários alternativos com valores 3.939 e 385.6 respectivamente.

As Tabelas 5.7, 5.8 e 5.9 mostram comparações dois a dois entre TNBOX, SPG1 e SPG2. Por exemplo, na Tabela 5.7 lemos que em 13 problemas TNBOX utilizou menos tempo de CPU que SPG1 e nos 26 problemas restantes SPG1 foi mais rápido. Observe que em 29 problemas TNBOX utilizou menos avaliações de função que SPG1 enquanto que em 7 problemas SPG1 utilizou menos avaliações de função e em 3 problemas ambos métodos utilizaram o mesmo número. Esses resultados são praticamente invertidos quando observamos as avaliações de gradiente. Qualitativamente, a comparação TNBOX-SPG2 nos fornece os mesmos resultados. O Problema 15 ($n = 100$) foi excluído das comparações por as razoẽs acima citadas.

Na maioria dos casos, TNBOX utilizou menos avaliações de função e muitas mais avaliações de

Problema	n	IT	FE	GE	Tempo	$f(x)$	$\|g_P(x)\|$
1	100	4	5	14	0.10	1.0000e+02	7.5831e−07
1	1000	1	2	8	0.09	1.1487e+03	0.0000e+00
1	10000	4	5	14	0.35	1.0000e+04	7.5769e−07
2	100	8	9	92	0.09	5.0500e+02	4.0311e−06
2	500	8	9	178	0.25	1.2525e+04	4.8901e−06
2	1000	1	2	8	0.09	5.7493e+04	0.0000e+00
3	100	2	3	7	0.09	8.9319e−22	6.0522e−09
3	1000	2	4	10	0.09	7.2388e−18	5.3915e−06
3	10000	1	2	5	3.23	1.0001e+10	0.0000e+00
4	100	3	8	37	0.08	5.9535e−15	1.5431e−07
4	1000	27	56	3541	16.46	7.3442e−07	3.6815e−06
4	10000	5	13	1982	76.32	5.9376e−13	1.5420e−06
5	100	7	8	43	0.08	2.0526e−13	4.0994e−06
5	1000	7	8	48	0.14	1.1193e−14	1.2382e−06
5	3000	8	9	62	0.40	7.2004e−01	2.3872e−06
6	100	24	25	167	0.09	2.8805e−09	4.4993e−06
6	1000	4	5	29	0.11	2.0290e+09	0.0000e+00
6	10000	38	46	1484	32.28	7.2062e−10	3.9721e−06
7	100	27	35	129	0.08	6.7887e−14	2.3285e−07
7	1000	16	22	86	0.20	1.2500e+02	5.6930e−08
7	10000	9	12	44	0.81	2.5301e−15	2.2351e−06
8	100	8	9	62	0.09	2.5592e+06	0.0000e+00
8	1000	34	37	176	0.33	9.6861e−03	6.3193e−07
8	10000	24	25	74	1.33	9.9001e−02	1.0334e−07
9	100	12	13	181	0.08	5.1755e−15	1.0971e−06
9	1000	2	3	11	0.10	5.0496e−17	4.0510e−08
10	100	25	39	117	0.08	7.1451e−17	6.8089e−07
10	1000	37	53	157	0.31	3.5785e−13	1.1964e−06
11	100	14	15	57	0.11	3.9813e−08	9.9855e−06
11	1000	10	11	33	0.22	8.7813e−14	2.9633e−07
12	100	65	98	1966	0.34	1.0000e+00	1.6442e−06
12	500	10	13	1091	0.57	4.9364e+02	6.1326e−08
13	100	7	8	39	0.07	1.0913e+02	1.4074e−06
13	1000	8	9	39	0.13	1.1081e+03	3.7460e−06
13	10000	10	15	61	1.16	1.1099e+04	2.6672e−05
14	100	8	11	36	0.09	1.2027e+04	5.7063e−05
14	1000	11	12	46	0.14	1.2147e+05	4.5524e−07
14	10000	2	3	8	0.21	1.0098e+07	5.1418e−07
15	100	12	13	67	0.08	3.8597e+02	78.0520e−07
15	1000	1	3	8	0.08	0.0000e+00	0.0000e+00

Tabela 5.2: Comportamento de TNBOX.

Problema	n	IT	FE	GE	LS	Tempo	$f(x)$	$\|g_P(x)\|$
1	100	6	7	7	0	0.00	1.0000e+02	1.9175e−11
1	1000	2	3	3	0	0.02	1.1487e+03	0.0000e+00
1	10000	5	6	6	0	0.35	1.0000e+04	7.1368e−08
2	100	384	766	385	380	0.25	5.0500e+02	3.8608e−06
2	500	1123	2813	1124	1122	4.22	1.2525e+04	4.7426e−06
2	1000	6	7	7	0	0.05	5.7493e+04	0.0000e+00
3	100	15	89	16	15	0.02	1.4471e−24	2.2677e−10
3	1000	21	162	22	20	0.12	1.9409e−21	8.8931e−08
3	10000	1	2	2	0	0.08	1.0001e+10	0.0000e+00
4	100	4	7	5	1	0.02	1.7331e−14	1.3165e−07
4	1000	1	2	2	0	0.03	9.8505e−08	0.0000e+00
4	10000	1	4	2	1	0.43	2.7885e−23	5.2806e−12
5	100	71	212	72	70	0.05	2.6691e−13	3.8515e−06
5	1000	78	235	79	77	0.60	1.7499e−14	9.5028e−07
5	3000	63	380	64	62	2.68	7.2005e−01	1.3915e−06
6	100	1203	1338	1204	58	0.27	1.1686e−08	4.4962e−06
6	1000	1	2	2	0	0.00	2.0291e+09	0.0000e+00
6	10000	7000	26611	7001	6973	526.22	5.7527e−07	1.4671e−04
7	100	722	2185	723	720	0.45	6.3353e−14	2.2603e−07
7	1000	2	6	3	1	0.02	1.2500e+02	2.4825e−13
7	10000	6626	23316	6627	6624	569.70	5.6219e−12	2.2330e−06
8	100	1	2	2	0	0.00	2.5592e+06	0.0000e+00
8	1000	14	17	15	2	0.05	9.6862e−03	2.6624e−09
8	10000	7000	74635	7001	5657	1671.43	9.9002e−02	1.5354e−07
9	100	1456	5821	1457	1455	1.35	3.1531e−14	1.0865e−06
9	1000	7000	39495	7001	6999	85.82	2.0927e−12	9.0783e−06
10	100	13	137	14	12	0.03	1.0317e−28	1.1624e−11
10	1000	11	244	12	11	0.50	3.6989e−24	7.0279e−08
11	100	7	19	8	5	0.00	0.0000e+00	0.0000e+00
11	1000	7000	20954	7001	6997	162.40	1.1911e−06	1.3076e−04
12	100	7000	27997	7001	6999	11.12	1.0000e+00	1.3589e−03
12	500	7000	271659	7001	6999	405.02	4.9364e+02	9.0817e−07
13	100	32	114	33	31	0.02	1.0913e+02	1.1365e−06
13	1000	122	245	123	121	0.32	1.1082e+03	1.8704e−06
13	10000	125	253	126	124	4.15	1.1099e+04	7.7663e−06
14	100	179	892	180	179	0.08	1.2027e+04	1.7252e−05
14	1000	7000	200744	7001	7000	156.87	1.2147e+05	3.8153e−04
14	10000	7000	14001	7001	6999	244.52	1.0098e+07	8.4162e−05
15	100	7000	72926	7001	6999	35.23	3.7810e+02	3.2919e−06
15	1000	827	3303	828	826	16.78	7.0868e−23	9.2431e−11

Tabela 5.3: Comportamento do Algoritmo de Gradientes Projetados (monótono).

Problema	n	IT	FE	GE	LS	Tempo	$f(x)$	$\|g_P(x)\|$
1	100	6	7	7	0	0.02	1.0000e+02	1.9175e−11
1	1000	2	3	3	0	0.03	1.1487e+03	0.0000e+00
1	10000	5	6	6	0	0.33	1.0000e+04	7.1368e−08
2	100	409	688	410	260	0.23	5.0500e+02	2.6744e−06
2	500	2989	9112	2990	2739	12.83	1.2525e+04	4.6410e−06
2	1000	6	7	7	0	0.03	5.7493e+04	0.0000e+00
3	100	3339	15644	3340	3041	1.05	8.9320e−22	6.0522e−09
3	1000	377	3096	378	374	2.13	3.4110e−20	3.6963e−07
3	10000	1	2	2	0	0.17	1.0001e+10	0.0000e+00
4	100	4	7	5	1	0.00	1.7331e−14	1.3165e−07
4	1000	1	2	2	0	0.03	9.8505e−08	0.0000e+00
4	10000	1	4	2	1	0.43	2.7885e−23	5.2806e−12
5	100	1571	4434	1572	1570	1.07	4.2057e−14	3.6193e−06
5	1000	2115	5993	2116	2112	15.62	4.5674e−15	1.1932e−06
5	3000	6861	40993	6862	6853	293.35	7.2005e−01	2.2651e−06
6	100	195	836	196	135	0.07	7.2628e−10	4.4469e−06
6	1000	1	2	2	0	0.00	2.0291e+09	0.0000e+00
6	10000	7000	26844	7001	6891	556.12	2.3716e−07	5.9601e−05
7	100	1125	3605	1126	1125	0.70	6.4472e−14	2.2802e−07
7	1000	1356	4177	1357	1336	9.27	1.2500e+02	4.4251e−10
7	10000	4472	12264	4473	3059	322.12	2.1856e−15	2.0924e−06
8	100	1	2	2	0	0.00	2.5592e+06	0.0000e+00
8	1000	47	50	48	2	0.15	9.6862e−03	3.5655e−09
8	10000	1636	2952	1637	183	91.75	9.9002e−02	1.0331e−07
9	100	5990	23253	5991	5989	5.55	3.4510e−16	1.0797e−06
9	1000	7000	37922	7001	6982	81.57	6.0094e−14	2.6800e−06
10	100	1403	11852	1404	1396	2.30	1.5453e−20	1.4462e−07
10	1000	1357	18583	1358	1357	37.67	2.1929e−22	5.4113e−07
11	100	7000	14003	7001	6967	10.50	1.7005e−07	1.4533e−03
11	1000	7000	13994	7001	6970	108.40	1.2197e−06	1.2206e−03
12	100	7000	27500	7001	6999	9.05	1.0000e+00	1.2310e−03
12	500	7000	53516	7001	6999	82.33	4.9364e+02	3.0893e−06
13	100	3860	13765	3861	3758	1.48	1.0913e+02	1.4046e−06
13	1000	5105	17981	5106	5043	17.42	1.1082e+03	3.6337e−06
13	10000	442	924	443	420	15.22	1.1099e+04	2.5386e−05
14	100	2062	9312	2063	2061	0.82	1.2027e+04	7.2244e−06
14	1000	7000	66704	7001	6997	58.72	1.2147e+05	4.5352e−05
14	10000	268	796	269	199	11.40	1.0098e+07	3.8830e−07
15	100	453	1804	454	452	0.85	3.9391e+00	8.0355e−07
15	1000	741	2957	742	740	14.95	3.9085e−23	9.9795e−11

Tabela 5.4: Comportamento do Algoritmo de Gradientes Projetados (não-monótono).

Problema	n	IT	FE	GE	LS	Tempo	$f(x)$	$\|g_P(x)\|$
1	100	7	8	8	0	0.00	1.0000e+02	1.6355e−10
1	1000	2	3	3	0	0.03	1.1487e+03	0.0000e+00
1	10000	6	7	7	0	0.38	1.0000e+04	3.1193e−07
2	100	83	86	84	2	0.03	5.0500e+02	2.1611e−06
2	500	244	278	245	28	0.62	1.2525e+04	4.5894e−06
2	1000	7	8	8	0	0.03	5.7493e+04	0.0000e+00
3	100	2	6	3	1	0.00	1.4471e−24	2.2677e−10
3	1000	2	3	3	0	0.02	7.1801e−18	5.3692e−06
3	10000	1	2	2	0	0.22	1.0001e+10	0.0000e+00
4	100	5	8	6	1	0.02	4.0900e−17	6.3953e−09
4	1000	1	2	2	0	0.03	9.8505e−08	0.0000e+00
4	10000	1	4	2	1	0.42	2.7885e−23	5.2806e−12
5	100	33	34	34	0	0.02	1.4356e−13	2.3828e−06
5	1000	40	41	41	0	0.13	2.7727e−15	4.9740e−07
5	3000	37	38	38	0	0.42	7.2005e−01	4.1071e−07
6	100	105	113	106	2	0.02	7.1607e−09	4.1145e−06
6	1000	1	2	2	0	0.02	2.0291e+09	0.0000e+00
6	10000	1187	1479	1188	202	34.28	9.6518e−09	3.9325e−06
7	100	65	77	66	7	0.02	3.2518e−19	2.0187e−08
7	1000	3	4	4	0	0.02	1.2500e+02	2.4825e−12
7	10000	37	39	38	1	1.43	6.3071e−16	1.1240e−06
8	100	1	2	2	0	0.00	2.5592e+06	0.0000e+00
8	1000	122	2029	123	37	3.83	9.6862e−03	3.2255e−09
8	10000	64	569	65	10	12.60	9.9002e−02	3.8221e−09
9	100	182	189	183	6	0.05	1.3256e−14	1.0688e−06
9	1000	949	1080	950	109	3.40	9.2697e−17	3.9397e−08
10	100	37	38	38	0	0.02	9.4947e−29	1.1108e−11
10	1000	47	101	48	1	0.27	4.9334e−24	8.1164e−08
11	100	161	177	162	10	0.20	1.5263e−08	4.3545e−06
11	1000	656	840	657	87	8.15	9.3780e−10	2.7266e−07
12	100	1193	2183	1194	220	0.78	1.0000e+00	1.6384e−06
12	500	180	182	181	1	0.37	4.9364e+02	2.5351e−09
13	100	29	30	30	0	0.02	1.0913e+02	9.9372e−07
13	1000	27	28	28	0	0.05	1.1082e+03	1.4593e−06
13	10000	29	30	30	0	0.75	1.1099e+04	1.6610e−05
14	100	22	29	23	2	0.00	1.2027e+04	1.0052e−05
14	1000	146	167	147	17	0.33	1.2147e+05	1.5118e−07
14	10000	2	3	3	0	0.05	1.0098e+07	0.0000e+00
15	100	95	170	96	1	0.10	7.8770e+00	3.4002e−07
15	1000	115	214	116	5	1.17	1.8324e−23	3.4579e−11

Tabela 5.5: Comportamento de SPG1.

Problema	n	IT	FE	GE	LS	Tempo	$f(x)$	$\|g_P(x)\|$
1	100	7	8	8	0	0.00	1.0000e+02	1.6355e−10
1	1000	2	3	3	0	0.02	1.1487e+03	0.0000e+00
1	10000	6	7	7	0	0.42	1.0000e+04	3.1193e−07
2	100	83	86	84	2	0.03	5.0500e+02	2.1611e−06
2	500	191	220	192	24	0.50	1.2525e+04	4.6407e−06
2	1000	7	8	8	0	0.05	5.7493e+04	0.0000e+00
3	100	2	5	3	1	0.00	1.1915e−23	7.0559e−10
3	1000	2	3	3	0	0.00	7.1801e−18	5.3692e−06
3	10000	1	2	2	0	0.13	1.0001e+10	0.0000e+00
4	100	5	8	6	1	0.03	4.0900e−17	6.3953e−09
4	1000	1	2	2	0	0.03	9.8505e−08	8.0357e−19
4	10000	1	4	2	1	0.42	2.7885e−23	5.2806e−12
5	100	33	34	34	0	0.02	1.4356e−13	2.3828e−06
5	1000	40	41	41	0	0.13	2.7727e−15	4.9740e−07
5	3000	37	38	38	0	0.53	7.2005e−01	4.1071e−07
6	100	59	61	60	1	0.02	7.1192e−09	3.1542e−06
6	1000	1	2	2	0	0.00	2.0291e+09	1.7554e−15
6	10000	1280	1604	1281	229	37.28	9.7823e−09	3.9353e−06
7	100	68	82	69	10	0.02	5.7384e−15	6.7725e−08
7	1000	3	4	4	0	0.02	1.2500e+02	2.4825e−12
7	10000	36	38	37	1	1.38	2.5860e−25	4.4324e−12
8	100	1	2	2	0	0.00	2.5592e+06	0.0000e+00
8	1000	64	109	65	20	0.27	9.6862e−03	7.5192e−08
8	10000	59	87	60	9	2.87	9.9002e−02	7.3095e−09
9	100	182	189	183	6	0.05	1.3256e−14	1.0688e−06
9	1000	813	915	814	82	2.93	9.7533e−17	3.9991e−08
10	100	37	38	38	0	0.02	9.4947e−29	1.1108e−11
10	1000	7000	13575	7001	1996	35.48	1.0095e−11	6.3546e−06
11	100	177	198	178	14	0.20	5.2865e−08	9.9590e−06
11	1000	311	379	312	33	3.80	5.9512e−10	2.0164e−07
12	100	1897	2407	1898	343	0.90	1.0000e+00	1.5550e−06
12	500	180	182	181	1	0.38	4.9364e+02	2.0485e−08
13	100	29	30	30	0	0.02	1.0913e+02	9.9372e−07
13	1000	27	28	28	0	0.07	1.1082e+03	1.4593e−06
13	10000	29	30	30	0	0.78	1.1099e+04	1.6610e−05
14	100	27	30	28	2	0.00	1.2027e+04	1.9482e−05
14	1000	143	170	144	20	0.33	1.2147e+05	1.4654e−07
14	10000	2	3	3	0	0.10	1.0098e+07	0.0000e+00
15	100	92	97	93	2	0.05	5.7011e−17	9.0995e−08
15	1000	115	215	116	6	1.10	4.2342e−23	6.9816e−11

Tabela 5.6: Comportamento de SPG2.

	TNBOX	SPG1
FE	29	7
GE	7	30
Time	13	26

Tabela 5.7: TNBOX vs. SPG1.

	TNBOX	SPG2
FE	29	7
GE	8	30
Time	13	26

Tabela 5.8: TNBOX vs. SPG2.

gradiente que os métodos SPG. Essas avaliações de gradiente são utilizadas na tentativa de achar uma boa aproximação da direção de Newton que permita ao método utilizar poucas tentativas de passos e poucas iterações. Parece-nos que, em termos de tempo computacional, este trabalho é compensado pelas características dos métodos SPG.

A Tabela 5.9 não mostra diferenças significativas entre SPG1 e SPG2. Finalmente, na Tabela 5.10 vemos que a estratégia de *backtraking* foi utilizada pelos métodos SPG só em 13% das iterações e que, quando necessária, SPG2 foi mais rápido no processo de achar um ponto que satisfaça as condições de decréscimo suficiente. Os Problemas 8 ($n = 1000$) e 10 ($n = 1000$), onde os métodos SPG mostraram as maiores diferenças quando comparados um com outro, foram excluídos desta comparação.

5.4 Conclusões

É costumeiro interpretar a primeira tentativa de passo de um algoritmo de minimização como o minimizador do modelo quadrático $q(x)$ na região viável ou numa aproximação dela. Sempre é imposto que a informação de primeira ordem no ponto atual deve coincidir com a informação de primeira ordem do modelo quadrático. Portanto, a aproximação quadrática no ponto x_{k+1} deveria ser

$$q(x) = \frac{1}{2}\langle x - x_{k+1}, B_{k+1}(x - x_{k+1})\rangle + \langle g(x_{k+1}), x - x_{k+1}\rangle + f(x_{k+1})$$

	SPG1	SPG2
FE	8	8
GE	6	8
LS	7	6
Time	11	11

Tabela 5.9: SPG1 vs. SPG2.

	SPG1	SPG2
Buscas lineares por iteração	0,126	0,133
Avaliações de função por busca linear	3,37	1,61

Tabela 5.10: Estatísticas para os métodos SPG.

e

$$\nabla q(x) = B_{k+1}(x - x_{k+1}) + g(x_{k+1}).$$

Os métodos secantes são motivados pela condição de interpolação $\nabla f(x_k) = \nabla q(x_k)$. Vamos impor aqui a condição mais fraca

$$D_{s_k} q(x_k) = D_{s_k} f(x_k), \tag{5.4}$$

onde $D_d\varphi(x)$ denota a derivada direcional de φ ao longo da direção d (portanto $D_d\varphi(x) = \langle \nabla\varphi(x), d\rangle$). Um pequeno cálculo mostra que a condição (5.4) é equivalente a

$$\langle s_k, B_{k+1}s_k\rangle = \langle s_k, y_k\rangle. \tag{5.5}$$

Claramente, a escolha espectral

$$B_{k+1} = \frac{\langle s_k, y_k\rangle}{\langle s_k, s_k\rangle} I \tag{5.6}$$

(onde I é a matriz identidade) satisfaz (5.5). Agora, suponhamos que z é ortogonal a s_k e que x pertence a $\mathcal{L}_k$, a linha determinada por x_k e x_{k+1}. Calculando a derivada direcional de q ao longo de z em qualquer ponto $x \in \mathcal{L}_k$, e utilizando (5.6), obtemos

$$D_z q(x) = \langle B_{k+1}(x - x_{k+1}) + g(x_{k+1}), z\rangle = \langle g(x_{k+1}), z\rangle = D_z f(x_{k+1}).$$

Mais ainda, é fácil ver que a escolha espectral de B_{k+1} é a única escolha possível que preserva (5.5) e a propriedade

$$D_z q(x) = D_z f(x_{k+1}) \quad \text{para todo} \quad x \in \mathcal{L}_k \quad \text{e} \quad z \perp s_k. \tag{5.7}$$

Em contraste com a propriedade (5.7), satisfeita pela escolha espectral de B_{k+1}, os modelos gerados pela escolha secante têm a propriedade de que a derivada direcional do modelo coincide com a derivada direcional da função objetivo em x_k. A propriedade (5.7) diz que o modelo foi escolhido de tal forma que a informação de primeira ordem com respeito às direções ortogonais s_k é a mesma que a informação de primeira ordem da função objetivo em x_{k+1} para todos os pontos na linha $\mathcal{L}_k$. Isto significa que a informação de primeira ordem no ponto atual é privilegiada na construção do modelo quadrático, com relação à informação de segunda ordem que provêm da iteração anterior. Talvez esta seja uma das razões por trás da inesperada eficácia dos algoritmos de gradientes espectrais em relação a métodos secantes um pouco arbitrários. Não é necessário dizer que a forma especial de B_{k+1} trivializa o problema de minimizar o modelo no conjunto viável quando este é suficientemente simples. Este fato é sumamente explorado em SPG1 e SPG2.

As caixas não são o único tipo de conjuntos onde é trivial projetar. O problema de regularização com restrições de norma [72, 86, 88, 113], definido por

$$\text{Minimizar } f(x) \text{ sujeito a } x^T A x \leq r \tag{5.8}$$

onde A é simétrica e definida positiva pode ser reduzido a um problema de minimização com restrições de bola mediante uma mudança de variáveis é, neste caso, as projeções podem ser trivialmente calculadas. Um caso particular de (5.8) e o clássico subproblema de regiões de confiança, onde f é quadrática. Recentemente (veja [81, 97]) foram encontrados procedimentos para escapar de pontos estacionários não-globais deste problema. Assim, resulta sumamente importante obter algoritmos rápidos para achar pontos críticos, especialmente em problemas de grande porte. Veja [99, 102, 107].

Talvez, a característica mais importante dos algoritmos SPG é que eles são extremamente simples de programar, ao ponto de que qualquer um pode escrever seu próprio código utilizando qualquer linguagem de programação em um par de horas. Mais ainda, seu baixo requerimento de memória o faz um algoritmo muito atrativo para problemas de grande porte. É um pouco surpreendente que tão simples ferramenta seja competitiva com algoritmos elaborados que utilizam subrotinas e procedimentos numéricos amplamente testados.

Capítulo 6

Diferenciação Automática e Métodos de Gradientes Espectrais Projetados para Problemas de Controle Ótimo

6.1 Introdução

Muita atenção tem sido enfocada para o desenvolvimento de métodos numéricos para resolver problemas de controle ótimo. O enfoque mais popular neste campo tem sido a redução do problema original num problema de programação não-linear (veja, por exemplo, [98, 45, 111, 112]). Em [46], foi mostrado que o cálculo do gradiente neste caso particular está fortemente relacionado a técnicas de *fast automatic differentiation* (FAD), ou modo reverso de diferenciação automática (veja o Capítulo 2 ou [75, 76, 64, 77, 65]). Em [47], utilizando expressões gerais de FAD, o gradiente exato da função objetivo de um processo multipasso geral foi expresso utilizando fórmulas canônicas simples. Um dos objetivos deste trabalho é mostrar a aplicação destas fórmulas canônicas a processos de controle ótimo integrados por métodos numéricos na família de métodos Runge-Kutta. Outro objetivo é testar o comportamento dos métodos de gradientes espectrais projetados introduzidos no Capítulo 5. Estes métodos combinam o gradiente projetado clássico com dois ingredientes recentemente desenvolvidos em otimização: (i) os esquemas de busca linear não-monótona de Grippo, Lampariello e Lucidi [70] e (ii) o comprimento de passo espectral (introduzido por Barzilai e Borwein [5] e analisado por Raydan [100, 101]). Esta escolha do comprimento do passo requer um trabalho computacional pequeno e acelera muito a convergência

dos métodos do tipo gradiente. Os resultados numéricos apresentados no capítulo anterior, que mostram o bom desempenho destes métodos rápidos e fáceis de implementar, motivaram-nos a combinar os métodos de gradientes espectrais projetados com diferenciação automática. Ambas ferramentas são utilizadas neste trabalho para o desenvolvimento de códigos para a resolução numérica de problemas de controle ótimo.

Na Seção 6.2 deste capítulo, aplicamos as fórmulas canônicas à versão discreta do problema de controle ótimo. Na Seção 6.3, apresentamos informação concisa dos métodos de gradientes espectrais projetados. A Seção 6.4 apresenta alguns resultados numéricos. As conclusões são apresentadas na Seção 6.5.

6.2 Fórmulas canônicas

O problema de controle ótimo básico pode ser descrito como segue. Seja um processo governado por um sistema de equações diferenciais ordinárias

$$\frac{dx(t)}{dt} = f(x(t), u(t), \xi), \quad T_0 \leq t \leq T_f, \tag{6.1}$$

onde $x : [T_0, T_f] \to \mathbb{R}^{n_x}$, $u : [T_0, T_f] \to U \subseteq \mathbb{R}^{n_u}$, U é compacto e $\xi \in V \subseteq \mathbb{R}^{n_\xi}$. A função x é chamada estado, u é o controle e ξ é o vetor de parâmetros de desenho. A solução de (6.1) é uma função $x(t)$ com condição inicial $x(T_0) = x_0$. Em geral, os escalares T_0, T_f e o vetor x_0 são fixos. Se T_0, T_f ou x_0 têm que ser otimizados, podemos incluí-los no vetor de parâmetros de desenho ξ.

O problema consiste em achar uma função de controle $u(t) \in U$ e um vetor de parâmetros de desenho $\xi \in V$ tal que minimizem a função de custo $W(T_0, T_f, x(T_f), u(T_f), \xi)$ sujeitos às restrições mistas no estado, no controle e no vetor de parâmetros de desenho:

$$h(x(t), u(t), \xi) = 0, \quad q(x(t), u(t), \xi) \leq 0, \quad T_0 \leq t \leq T_f. \tag{6.2}$$

Via de regra, este problema é reduzido a um problema de programação matemática utilizando um esquema de discretização. A função de controle $u(t)$ é aproximada por uma função linear por partes, onde a precisão da discretização depende do problema que será resolvido. Tendo alguma experiência em resolver problemas práticos, chegamos a conclusão de que, comumente, a precisão da integração deve ser maior do que a precisão da discretização da função de controle. Portanto, por simplicidade, é possível supor que o vetor de controle u é constante em cada intervalo de integração. Discretizando o sistema (6.1), obtemos um processo de N passos no qual as funções

x e u são naturalmente representadas como vetores,

$$x^T = [x_0^T, x_1^T, \ldots, x_N^T] \text{ e } u^T = [u_0^T, u_1^T, \ldots, u_N^T],$$

onde $x_i = x(t_i) \in \mathbb{R}^{n_x}$, $u_i = u(t_i) \in \mathbb{R}^{n_u}$ e $t_i = T_0 + \sum_{k=0}^{i-1} \Delta t_k$ para $i = 0, 1, \ldots, N$, $0 < \Delta t_i \in \mathbb{R}$ são os passos de discretização e satisfazem

$$\sum_{i=0}^{N-1} \Delta t_i = T_f - T_0, \tag{6.3}$$

$x \in \mathbb{R}^{n_x(N+1)}$ e $u \in \mathbb{R}^{n_u(N+1)}$. A versão discreta de (6.1) é decomposta nas N relações

$$x_i = F(X_i, U_i, \xi), \quad i = 1, 2, \ldots, N, \tag{6.4}$$

onde X_i e U_i são conjuntos dados de variáveis x_j e u_j, respectivamente, e o índice j toma valores de 0 a N. No passo inicial temos $X_0 = U_0 = \emptyset$ e, por simplicidade, escrevemos $x_0 = F(X_0, U_0, \xi)$. As restrições mistas (6.2) são consideradas em cada ponto da malha como

$$h(x_i, u_i, \xi) = 0, \quad q(x_i, u_i, \xi) \leq 0, \quad i = 0, 1, \ldots, N, \tag{6.5}$$

e o problema de controle ótimo discretizado, para achar uma solução aproximada do problema original, pode ser escrito como

$$\begin{aligned} &\text{Minimizar } W(T_0, T_f, x_N, u_N, \xi) \\ &\text{sujeito a } \quad u_i \in U, \xi \in V \\ &\qquad\qquad\quad h(x_i, u_i, \xi) = 0 \\ &\qquad\qquad\quad q(x_i, u_i, \xi) \leq 0. \end{aligned} \tag{6.6}$$

Fixando os vetores u e ξ obtemos, por (6.4), o vetor de estado $x_N(u, \xi)$ e definimos a função composta $\mathcal{W}(u, \xi) = W(T_0, T_f, x_N(u, \xi), u_N, \xi)$. Para a minimização numérica desta função é importante conhecer as derivadas totais de $\mathcal{W}$ com respeito a u e ξ, dado que isto nos permite utilizar algoritmos eficientes baseados na direção do gradiente. Em [47], foi mostrado que, para processos multipasso como (6.4), fórmulas para calcular as derivadas totais $d\mathcal{W}/du$ e $d\mathcal{W}/d\xi$ podem ser obtidas como se segue.

Para cada conjunto X_i e U_i introduzimos os conjuntos de índices Q_i e K_i contendo os índices das variáveis x_j e u_j que pertencem aos conjuntos X_i e U_i, respectivamente, *i.e.*,

$$Q_i = \{j : x_j \in X_i\}, \text{ e } K_i = \{j : u_j \in U_i\}.$$

Também definimos seus conjuntos de índices conjugados

$$\bar{Q}_i = \{j : x_i \in X_j\} \text{ e } \bar{K}_i = \{j : u_i \in U_j\}.$$

Os conjuntos Q_i e K_i são os conjuntos de índices de **entrada** e $\bar{Q}_i$ e $\bar{K}_i$ são os conjuntos de índices de **saída** do i-ésimo passo. Definimos os vetores adjuntos $p_i \in \mathbb{R}^{n_x}$ para $i = 0, 1, \ldots, N$, o vetor adjunto total $p^T = [p_0^T, p_1^T, \ldots, p_N^T] \in \mathbb{R}^{n_x(N+1)}$ e a função auxiliar

$$E(x, u, \xi, p) = W(T_0, T_f, x_N, u_N, \xi) + \sum_{i=0}^{N} F(X_i, U_i, \xi)^T p_i. \tag{6.7}$$

Assim, as fórmulas para o cálculo dos vetores adjuntos p_i e as derivadas totais $d\mathcal{W}/du$ e $d\mathcal{W}/d\xi$ podem ser escritas da seguinte **forma canônica**:

$$x_i = E_{p_i}(x, u, \xi, p) = F_{x_i}(X_j, U_j, \xi), \tag{6.8}$$

$$p_i = E_{x_i}(x, u, \xi, p) = W_{x_i}(T_0, T_f, x_N, u_N, \xi) + \sum_{j \in \bar{Q}_i} F_{x_i}(X_j, U_j, \xi)^T p_j, \tag{6.9}$$

$$\frac{d\mathcal{W}(u, \xi)}{du_i} = E_{u_i}(x, u, \xi, p) = W_{u_i}(T_0, T_f, x_N, u_N, \xi) + \sum_{j \in \bar{K}_i} F_{u_i}(X_j, U_j, \xi)^T p_j, \tag{6.10}$$

$$\frac{d\mathcal{W}(u, \xi)}{d\xi} = E_{\xi}(x, u, \xi, p) = W_{\xi}(T_0, T_f, x_N, u_N, \xi) + \sum_{j=0}^{N} F_{\xi}(X_j, U_j, \xi)^T p_j, \tag{6.11}$$

para i variando de 0 a N. Nas fórmulas acima, e daqui em diante, Y_w denota a derivada parcial da função Y com respeito a w, *i.e.*, $Y_w = \partial Y/\partial w$ enquanto dZ/dw denota a derivada total de Z com respeito a w. Supomos que a relação (6.8) define um processo explícito, *i.e.*, a cada passo i o conjunto de entrada Q_i é tal que $k < i$ para todo $k \in Q_i$. Neste caso, por (6.9), temos

$$p_N = W_{x_N}(T_0, T_f, x_N, u_N, \xi). \tag{6.12}$$

Notemos que, considerando a expressão (6.9), podemos concluir que os valores adjuntos p_i são as derivadas parciais de $\mathcal{W}$ com respeito as variáveis de estado x_i.

Como aplicação prática, consideremos o processo multipasso (6.4) dado pela família de métodos de Runge-Kutta definidos por

$$\begin{cases} x_{i+1}^0 = F(X_{i+1}, U_{i+1}, \xi) = x_i^0 + \Delta t_i \sum_{j=0}^{M-1} \alpha_j f(z_i^j), \\ x_i^{j+1} = x_i^0 + \beta_j \Delta t_i f(z_i^j), \quad j = 0, \ldots, M-2, \end{cases} \tag{6.13}$$

Método de Integração	M	α	β
Runge-Kutta order 1 or Euler	1	$\alpha_0 = 1$	
Runge-Kutta order 2 or Modified Euler	2	$\alpha_0 = 0\ \alpha_1 = 1$	$\beta_0 = 0.5$
Runge-Kutta order 2	2	$\alpha_0 = \alpha_1 = 0.5$	$\beta_0 = 1$
Runge-Kutta order 4	4	$\alpha_0 = \alpha_3 = 1/6$ $\alpha_1 = \alpha_2 = 1/3$	$\beta_0 = \beta_1 = 0.5$ $\beta_2 = 1$

Tabela 6.1: Exemplos da família de métodos Runge-Kutta.

para $i = 0, \ldots, N-1$. Aqui, $x_i^0 \equiv x_i$, $t_i^0 \equiv t_i$, $t_i^j = t_i + \beta_{j-1}\Delta t_i$, $u_i^j = u(t_i^j)$, $z_i^j = (x_i^j, u_i^j, \xi)$ e os $x_i^j \in \mathbb{R}^{n_x}$ são vetores auxiliares. Como estamos supondo que as variáveis de controle u_i são constantes dentro do intervalo de integração, temos $u_i = u_i^j$ e $z_i^j = (x_i^j, u_i, \xi)$. A Tabela 6.1 mostra alguns exemplos da família de métodos de Runge-Kutta (veja, por exemplo, [108]).

Substituindo (6.13) em (6.7) obtemos

$$\begin{aligned} E(x,u,\xi,p) &= W(T_0,T_f,z_N^0) + \sum_{i=0}^{N-1}\sum_{j=0}^{M-2}[x_i^0 + \Delta t_i\beta_j f(z_i^j)]^T p_i^{j+1} \\ &+ \sum_{i=0}^{N-1}\left[x_i^0 + \sum_{j=0}^{M-1}\Delta t_i\alpha_j f(z_i^j)\right]^T p_{i+1}^0, \end{aligned} \tag{6.14}$$

e, aplicando as fórmulas canônicas (6.8)–(6.11), temos

$$p_i^0 = W_{x_i^0}(T_0,T_f,z_N^0) + p_i^1 + \Delta t_i\beta_0 f_{x_i^0}(z_i^0)^T p_i^1 + p_{i+1}^0 + \Delta t_i\alpha_0 f_{x_i^0}(z_i^0)^T p_{i+1}^0, \tag{6.15}$$

$$p_i^j = W_{x_i^j}(T_0,T_f,z_N^0) + \Delta t_i\beta_j f_{x_i^j}(z_i^j)^T p_i^{j+1} + \Delta t_i\alpha_j f_{x_i^j}(z_i^j)^T p_{i+1}^0, \tag{6.16}$$

$$\frac{dE}{du_i} = W_{u_i}(T_0,T_f,z_N^0) + \sum_{j=0}^{M-2}\Delta t_i\beta_j f_{u_i}(z_i^j)^T p_i^{j+1} + \sum_{j=0}^{M-1}\Delta t_i\alpha_j f_{u_i}(z_i^j)^T p_{i+1}^0, \tag{6.17}$$

para $i = 0, 1, \ldots, N-1$ e $j = 1, 2, \ldots, M-1$, e

$$\frac{dE}{du_N} = W_{u_N}(T_0,T_f,z_N^0), \tag{6.18}$$

$$\frac{dE}{d\xi} = W_\xi(T_0,T_f,z_N^0) + \sum_{i=0}^{N-1}\sum_{j=0}^{M-2}\Delta t_i\beta_j f_\xi(z_i^j)^T p_i^{j+1} + \sum_{i=0}^{N-1}\sum_{j=0}^{M-1}\Delta t_i\alpha_j f_\xi(z_i^j)^T p_{i+1}^0. \tag{6.19}$$

Finalmente, arrumando as fórmulas (6.15)–(6.19) e descartando as derivadas nulas, chegamos à Subrotina 2.1 para o cálculo de $\mathcal{W}(u,\xi)$, $d\mathcal{W}(u,\xi)/du$ e $d\mathcal{W}(u,\xi)/d\xi$. É necessário destacar que o enfoque acima apresentado (fórmulas (6.15)–(6.19)) é mais simples e próximo à implementação computacional do que os resultados análogos apresentados em [45] (pp. 379–382). Mais ainda, os nossos resultados podem ser aplicados a todos os métodos da família Runge-Kutta. Estas melhorias provêm da aplicação da função auxiliar (6.7) e das fórmulas canônicas (6.8)–(6.11) introduzidas em [47].

Subrotina 2.1

Iniciar $x_0^0 \leftarrow x_0$.

Para $i = 0, \ldots, N-1$ *(crescente)*

  Para $j = 0, \ldots, M-2$ *(crescente, definido só para $M \geq 2$)*

    Fazer $y \leftarrow f(z_i^j)$,

    calcular $x_i^{j+1} = x_i^0 + \Delta t_i \beta_j y$ e

    calcular $x_{i+1}^0 = x_{i+1}^0 + \Delta t_i \alpha_j y$.

  fimpara

  Fazer $y \leftarrow f(z_i^{M-1})$ e

  calcular $x_{i+1}^0 = x_{i+1}^0 + \Delta t_i \alpha_{M-1} y$.

fimpara

Calcular $W(T_0, T_f, z_N^0)$.

Fazer $dE/d\xi \leftarrow W_\xi(T_0, T_f, z_N^0)$,

fazer $dE/du_N \leftarrow W_{u_N}(T_0, T_f, z_N^0)$ e

fazer $p_N \leftarrow W_{x_N}(T_0, T_f, z_N^0)$.

Para $i = N-1, \ldots, 0$ *(decrescente)*

  Calcular $dE/d\xi = dE/d\xi + \Delta t_i \alpha_{M-1} f_\xi(z_i^{M-1})^T p_{i+1}$,

  calcular $dE/du_i = \Delta t_i \alpha_{M-1} f_{u_i}(z_i^{M-1})^T p_{i+1}$,

  calcular $v = \Delta t_i \alpha_{M-1} f_{x_i^{M-1}}(z_i^{M-1})^T p_{i+1}$ e

  calcular $p_i = p_{i+1} + v$.

  Para $j = M-2, \ldots, 0$ *(decrescente, definido só para $M \geq 2$)*

    Calcular $y = \alpha_j p_{i+1} + \beta_j v$,

    calcular $dE/d\xi = dE/d\xi + \Delta t_i \alpha_j f_\xi(z_i^j)^T y$,

```
        calcular dE/du_i = dE/du_i + Δt_i α_j f_{u_i}(z_i^j)^T y,
        calcular v = Δt_i f_{x_i^j}(z_i^j)^T y  e
        calcular p_i = p_i + v.
    fimpara
fimpara
```

Na Subrotina 2.1, $y, v \in \mathbb{R}^{n_x}$ são vetores auxiliares. Observemos que não é necessário armazenar os valores adjuntos p_i^j das variáveis intermediárias x_i^j quando $j \neq 0$. Por essa razão, utilizamos a notação p_i para os valores adjuntos p_i^0 das variáveis x_i^0. Este tipo de implementação é uma **estratégia mista** que trata de achar um equilíbrio entre custo computacional e requerimentos de memória para armazenamento. É fácil ver que, como W e suas derivadas parciais, W_ξ, W_{u_N} e W_{x_N}, estão sendo calculadas juntas, poderíamos chamar uma única subrotina para calcular todas elas utilizando o modo reverso. Este não é o caso da função f e suas derivadas, que estão sendo calculadas em diferentes passos da Subrotina 2.1. Por esta razão, não é possível, trivialmente, tirar vantagem de expressões comuns entre f e f_{x_i}, f_{u_i} e f_ξ. Chamamos essa estratégia de FAD **híbrida**. Nesta implementação estamos sacrificando tempo computacional para poupar memória para armazenamento.

Uma classe particular e importante de problemas de controle é aquela na qual o objetivo é minimizar a duração do processo. Uma estratégia possível para tratar esta situação é introduzir um novo parâmetro de desenho ξ^{n_ξ}, definir a função objetivo do problema (6.6) como

$$W(T_0, T_f, z_N^0) = (T_f - T_0)\xi^{n_\xi}, \tag{6.20}$$

e substituir (6.1) por

$$\frac{dx(t)}{dt} = \bar{f}(x, u, \xi) = \xi^{n_\xi} f(x, u, \xi), \quad T_0 \leq t \leq T_f, \tag{6.21}$$

$$0 \leq \xi^{n_\xi} < +\infty. \tag{6.22}$$

Se utilizarmos a Subrotina 2.1 para calcular o gradiente de (6.20), f será calculada duas vezes no mesmo ponto: $\bar{f}(x, u, \xi) = \xi^{n_\xi} f(x, u, \xi)$ e $\bar{f}_{\xi^{n_\xi}}(x, u, \xi) = f(x, u, \xi)$. Para superar este problema, deveríamos utilizar, no lugar do vetor auxiliar $y \in \mathbb{R}^{n_x}$, um vetor tridimensional $y \in \mathbb{R}^{n_x \times N \times M}$ (ou NM vetores $y_i^j \in \mathbb{R}^{n_x}$). Desta forma, os valores de $f(z_i^j)$ serão armazenados em y_i^j e depois utilizados no cálculo de $\bar{f}_{\xi^{n_\xi}}(z_i^j)$. Esta modificação surge a partir da análise da Subrotina 2.1 ou

da aplicação das fórmulas canônicas à seguinte reformulação dos métodos da família Runge-Kutta:

$$\begin{cases} f_i^j = f(z_i^j), \quad j = 0, \ldots, M-1, \\ x_{i+1}^0 = F(X_{i+1}, U_{i+1}, \xi) = x_i^0 + \Delta t_i \sum_{j=0}^{M-1} [\alpha_j \xi^{n_\xi} f_i^j], \\ x_i^{j+1} = x_i^0 + \beta_j \Delta t_i \xi^{n_\xi} f_i^j, \quad j = 0, \ldots, M-2, \end{cases} \tag{6.23}$$

para $i = 0, \ldots, N-1$. Esta é uma modificação *ad-hoc* para um caso em particular. Neste novo enfoque estamos dobrando os requerimentos de memória para armazenamento para poupar o custo computacional de avaliar f duas vezes. Para mostrar como utilizar a metodologia, vamos a aplicar a Subrotina 2.1 para todos os problemas apresentados na Seção 6.4.

6.3 Métodos não-monótonos de gradientes espectrais projetados

Os algoritmos de gradientes espectrais projetados não-monótonos foram introduzidos no Capítulo 5. Estes métodos combinam o gradiente projetado clássico com dois ingredientes recentemente desenvolvidos em otimização: (i) os esquemas de busca linear não-monótona desenvolvidos por Grippo, Lampariello e Lucidi [70] para os métodos do tipo Newton e (ii) o comprimento de passo espectral, introduzido por Barzilai e Borwein [5] e analisado por Raydan [100, 101]. Nesta seção reproduzimos a definição das duas versões dos métodos de gradientes espectrais projetados. O primeiro (SPG1 daqui em diante) utiliza o gradiente projetado clássico como um caminho curvilíneo de busca. O segundo (SPG2 daqui em diante) utiliza o gradiente espectral projetado factível como direção de busca para evitar projeções adicionais durante o processo de busca unidimensional. Estes métodos aplicam-se a problemas do tipo

$$\begin{aligned} &\text{Minimizar } \varphi(x) \\ &\text{sujeito a } x \in S, \end{aligned} \tag{6.24}$$

onde S é um conjunto convexo e fechado em $\mathbb{R}^n$. Ao longo desta seção supomos que φ está definida e tem derivadas parciais contínuas num conjunto aberto que contém S.

Dado $z \in \mathbb{R}^n$, definimos $P(z)$ como a projeção ortogonal de z em S. Denotamos $g(x) = \nabla\varphi(x)$. Os algoritmos utilizam um inteiro $K \geq 1$, um parâmetro pequeno $\lambda_{min} > 0$, um parâmetro grande $\lambda_{max} > \lambda_{min}$, um parâmetro de decréscimo suficiente $\gamma \in (0,1)$ e parâmetros de salvaguardas $0 < \sigma_1 < \sigma_2 < 1$. Dados $\lambda_0 \in [\lambda_{min}, \lambda_{max}]$ e $x_0 \in S$, os Algoritmos 3.1 e 3.2 abaixo, descrevem como obter x_* tal que $\|P(x_* - g(x_*)) - x_*\|_2 = 0$.

Algoritmo 3.1

Iniciar $k \leftarrow 0$.

Enquanto $(\|P(x_k - g(x_k)) - x_k\|_2 \neq 0)$

Calcular $x_+ = P(x_k - \lambda_k g(x_k))$ e

fazer $\alpha \leftarrow \lambda_k$.

Enquanto $(\varphi(x_+) > \max_{0 \leq j \leq \min\{k, K-1\}}\{\varphi(x_{k-j})\} + \gamma\langle x_+ - x_k, g(x_k)\rangle)$

definir $\alpha \in [\sigma_1\alpha, \sigma_2\alpha]$ e

calcular $x_+ = P(x_k - \alpha g(x_k))$.

fimenquanto

Fazer $x_{k+1} \leftarrow x_+$ e

calcular $s_k = x_{k+1} - x_k$, $y_k = g(x_{k+1}) - g(x_k)$ and $b_k = \langle s_k, y_k\rangle$.

Se $(b_k \leq 0)$ fazer $\lambda_{k+1} \leftarrow \lambda_{max}$

senão calcular $\lambda_{k+1} = \min\{\lambda_{max}, \max\{\lambda_{min}, \frac{\langle s_k, s_k\rangle}{b_k}\}\}$.

Fazer $k \leftarrow k + 1$.

fimenquanto

Fazer $x_* \leftarrow x_k$.

Algoritmo 3.2

Iniciar $k \leftarrow 0$.

Enquanto $(\|P(x_k - g(x_k)) - x_k\|_2 \neq 0)$

Calcular $d_k = P(x_k - \lambda_k g(x_k)) - x_k$,

calcular $x_+ = x_k + d_k$ e

fazer $\alpha \leftarrow 1$.

Enquanto $(\varphi(x_{k+1}) > \max_{0 \leq j \leq \min\{k, K-1\}}\{\varphi(x_{k-j})\} + \gamma\alpha\langle d_k, g(x_k)\rangle)$

definir $\alpha \in [\sigma_1\alpha, \sigma_2\alpha]$ e

calcular $x_+ = x_k + \alpha d_k$.

fimenquanto

Fazer $x_{k+1} \leftarrow x_+$ e

calcular $s_k = x_{k+1} - x_k$, $y_k = g(x_{k+1}) - g(x_k)$ and $b_k = \langle s_k, y_k\rangle$.

Se $(b_k \leq 0)$ fazer $\lambda_{k+1} \leftarrow \lambda_{max}$

senão calcular $\lambda_{k+1} = \min\{\lambda_{max}, \max\{\lambda_{min}, \frac{\langle s_k, s_k\rangle}{b_k}\}\}$.

Fazer $k \leftarrow k + 1$.

fimenquanto

Fazer $x_* \leftarrow x_k$.

O cálculo de $\alpha \in [\sigma_1\alpha, \sigma_2\alpha]$ utiliza uma interpolação quadrática unidimensional.

6.4 Resultados numéricos

Nossos códigos requerem cinco subrotinas implementadas em C/C++. A primeira, para calcular a função objetivo W e suas derivadas parciais W_{x_N}, W_{u_N} e W_ξ, deveria utilizar o modo reverso. A segunda é utilizada para calcular as restrições de igualdade h e h_{x_i}, h_{u_i} e h_ξ; a terceira para calcular as restrições de desigualdade q e suas derivadas parciais; a quarta para calcular f; finalmente, a última para calcular f_{x_i}, f_{u_i} e f_ξ. Como nos problemas-teste as funções (f e W) e as restrições (h e q) são suficientemente simples, decidimos escrever seus códigos à mão. No caso geral, pacotes de diferenciação automática como ADOL-C [69] ou ADIC++ (ver Capítulo 2) deveriam ser utilizados. Alguns parâmetros estão conectados com os métodos Runge-Kutta e o passo de integração. Os parâmetros M, α e β servem para escolher o método de Runge-Kutta (veja a Tabela 6.1). Os métodos listados nessa tabela são só exemplos e qualquer outro método na família Runge-Kutta pode ser utilizado. O tamanho do passo de integração Δt pode ser fixo ou, como definido em (6.3), diferentes tamanhos de passo de integração Δt_i podem ser considerados.

Os problemas-teste são os reportados em [62]. Dividimos os 11 problemas em três grupos de acordo com sua complexidade. O primeiro grupo inclui os problemas de controle mais simples (que não tem restrições nas variáveis do estado final). Problemas com restrições nas variáveis do estado final pertencem ao segundo grupo. Finalmente, o terceiro grupo contém os problemas nos quais o objetivo é minimizar a duração do processo e há restrições no estado final. Para permitir a reproduzibilidade dos resultados, e como a referência original está em russo, todas nossas formulações contínuas dos problemas-teste são listadas embaixo. Uma descrição completa dos seus significados físicos e dos controles e trajetórias ótimas pode ser encontrada em [62].

É necessário mencionar que neste tipo de problemas de otimização que derivam de problemas de controle ótimo, a complexidade da função objetivo depende não só da dimensão da solução, $n = n_u(N+1) + n_\xi$, mas também depende da dimensão das variáveis de estado. Uma boa medida da complexidade da função objetivo é $n_{ocp} = (n_x + n_u)(N+1) + n_\xi$. A Tabela 6.2 apresenta os grupos e as variáveis n and n_{ocp}.

Problema 1.a: $(n_x = 3, n_u = 1, n_\xi = 0)$

$$f(x(t), u(t), \xi)^T = [x^1(t), (1 - x^0(t)^2)x^1(t) - x^0(t) + u^0(t), x^0(t)^2 + x^1(t)^2 + u^0(t)^2], \quad T_0 \le t \le T_f,$$

$$x(T_0)^T = [3, 0, 0], \quad T_0 = 0, \quad T_f = 10,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f), \quad -10^{10} \le u^0(\cdot) \le 10^{10},$$

e controle inicial $u(\cdot) \equiv 0$.

Problema 1.b: $(n_x = 3, n_u = 1, n_\xi = 0)$

$$f(x(t), u(t), \xi)^T = [x^1(t), (1 - x^0(t)^2)x^1(t) - x^0(t) + u^0(t), x^0(t)^2 + x^1(t)^2 + u^0(t)^2], \quad T_0 \le t \le T_f,$$

$$x(T_0)^T = [0, 1, 0], \quad T_0 = 0, \quad T_f = 5,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f), \quad -10^{10} \le u(\cdot) \le 10^{10},$$

e controle inicial $u(\cdot) \equiv 0$.

Problema 1.c: $(n_x = 3, n_u = 1, n_\xi = 0)$

$$f(x(t), u(t), \xi)^T = [x^1(t), (1 - x^0(t)^2)x^1(t) - x^0(t) + u^0(t), x^0(t)^2 + x^1(t)^2 + u^0(t)^2], \quad T_0 \le t \le T_f,$$

$$x(T_0)^T = [0, 1, 0], \quad T_0 = 0, \quad T_f = 5,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f), \quad -0.8 \le u(\cdot) \le 0.8,$$

e controle inicial $u(\cdot) \equiv 0$.

Problema 2: Uso de catálise para duas reações químicas sucessivas $(n_x = 2, n_u = 1, n_\xi = 0)$

$$f(x(t), u(t), \xi)^T = [u^0(t)(10x^1(t) - x^0(t)), -u^0(t)(10x^1(t) - x^0(t)) - (1 - u^0(t))x^1(t)], \quad T_0 \le t \le T_f,$$

$$x(T_0)^T = [1, 0], \quad T_0 = 0, \quad T_f = 1,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^0(T_f) + x^1(T_f), \quad 0 \le u(\cdot) \le 1$$

e controle inicial $u(\cdot) \equiv 0.5$.

Problema 3: Reações químicas entre gases $(n_x = 2, n_u = 1, n_\xi = 0)$

$$f(x(t), u(t), \xi)^T = [f^0(x(t), u(t), \xi), f^1(x(t), u(t), \xi)],$$

$$f^0(x(t), u(t), \xi) = -2c^1 x^0(t) u^0(t)/(2c^0 + x^1(t)),$$

$$f^1(x(t), u(t), \xi) = -2f^1(x(t), u(t), \xi) - c^2 u^0(t)^2 (2x^1(t)/(2c^0 + x^1(t)))^2, \quad T_0 \le t \le T_f,$$

$$c^0 = 1.475 \times 10^{-2}, \quad c^1 = 1.8725688803 \times 10^7, \quad c^2 = 1.2162426427 \times 10^{14},$$

$$x(T_0)^T = [0.0105, 0.0085], \quad T_0 = 0, \quad T_f = 8,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = -100x^1(T_f), \quad 0 \leq u(\cdot) \leq 1$$

e controle inicial $u(\cdot) \equiv 0$.

Problema 4: Problema da Dolichobrachistochrone ($n_x = 2, n_u = 1, n_\xi = 0$)

$$f(x(t), u(t), \xi)^T = [u^0(t), \sqrt{(1 + u^0(t)^2)/x^0(t)}], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [3, 0], \quad T_0 = 0, \quad T_f = 2,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^1(T_f), \quad x^0(T_f) = 10, \quad -10^{10} \leq u(\cdot) \leq 10^{10}$$

e controle inicial $u(\cdot) \equiv 5$.

Problema 5: Rotação de uma máquina elétrica ($n_x = 3, n_u = 1, n_\xi = 0$)

$$f(x(t), u(t), \xi)^T = [x^1(t), -(4 + 0.8x^1(t)) + u^0(t), u^0(t)^2], \quad T_0 \leq t \leq T_f,$$

$$x(T_0) = 0_{n_x}, \quad T_0 = 0, \quad T_f = 2,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = x^2(T_f),$$

$$x^0(T_f) = 1, \quad x^1(T_f) = 0, \quad -10^{10} \leq u(\cdot) \leq 10^{10}$$

e controle inicial $u(\cdot) \equiv 0$.

Problema 6: Movimento de um foguete ($n_x = 5, n_u = 2, n_\xi = 0$)

$$f(x(t), u(t), \xi)^T = [x^1(t), u^0(t), x^3(t), u^1(t) - 9.81, u^0(t)^2 + u^1(t)^2], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [-1, 0, 0, 0, 0], \quad T_0 = 0, \quad T_f = 4,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = \sqrt{x^4(T_f)},$$

$$x^0(T_f) = x^1(T_f) = x^2(T_f) = x^3(T_f) = 0, \quad -10^{10}_{n_u} \leq u(\cdot) \leq 10^{10}_{n_u}$$

e controle inicial $u(\cdot)^T \equiv (0, 1)$.

Problema 7: ($n_x = 3, n_u = 1, n_\xi = 1$)

$$f(x(t), u(t), \xi)^T = \xi^0[x^1(t), x^2(t), u^0(t)], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [16, 0, 0], \quad T_0 = 0, \quad T_f = 1,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad -1 \leq u(\cdot) \leq 1, \quad 0 \leq \xi^0 \leq 10^{10},$$

controle inicial $u(\cdot) \equiv 0$ e parâmetro de desenho inicial $\xi = 1$.

Problema 8: Controle de movimento de aeronaves ($n_x = 4, n_u = 2, n_\xi = 1$)

$$f(x(t), u(t), \xi)^T = \xi^0[x^2(t), x^3(t), -x^3(t) + u^0(t)sin(u^1(t)), x^2(t) + u^0(t)cos(u^1(t))], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [10, 0, 0, 0], \quad T_0 = 0, \quad T_f = 5,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi,$$

$$x(T_f) = 0_{n_x}, \quad 0 \leq u^0(\cdot) \leq 1, \quad -3.14 \leq u^1(\cdot) \leq 3.14, \quad 0 \leq \xi^0 \leq 10^{10},$$

controle inicial $u(\cdot) \equiv 0_{n_u}$ e parâmetro de desenho inicial $\xi = 1$.

Problema 9: Dinâmica de foguetes com duração mínima ($n_x = 5, n_u = 2, n_\xi = 1$)

$$f(x(t), u(t), \xi)^T = \xi^0[x^1(t), u^0(t), x^3(t), u^1(t) - 9.81, u^0(t)^2 + u^1(t)^2], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [-1, 0, 1, 0, 0], \quad T_0 = 0, \quad T_f = 4,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi,$$

$$x^0(T_f) = x^1(T_f) = x^2(T_f) = x^3(T_f) = 0, \sqrt{x^4(T_f)} - 13 = 0,$$

$$-10^{10}_{n_u} \leq u(\cdot) \leq 10^{10}_{n_u}, \quad 0 \leq \xi^0 \leq 10^{10}$$

controle inicial $u(\cdot)^T \equiv [0, 5]$ e parâmetro de desenho inicial $\xi = 1$.

Problema 10: Lançamento de satélites ($n_x = 3, n_u = 1, n_\xi = 1$)

$$f(x(t), u(t), \xi)^T = \xi^0[x^1(t), -x^0(t) + x^2(t), u^0(t)], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [0.052, 0, -0.145], \quad T_0 = 0, \quad T_f = 6.28,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad -0.3 \leq u(\cdot) \leq 0.3, \quad 0 \leq \xi^0 \leq 10^{10}$$

controle inicial $u(\cdot) \equiv 0$ e parâmetro de desenho inicial $\xi = 1$.

Problema 11: Parada de um corpo em rotação ($n_x = 2, n_u = 1, n_\xi = 1$)

$$f(x(t), u(t), \xi)^T = \xi^0[x^1(t), u^0(t)], \quad T_0 \leq t \leq T_f,$$

$$x(T_0)^T = [2, 1], \quad T_0 = 0, \quad T_f = 8,$$

$$W(T_0, T_f, x(T_f), u(T_f), \xi) = (T_f - T_0)\xi, \quad x(T_f) = 0_{n_x}, \quad -1 \leq u(\cdot) \leq 1, \quad 0 \leq \xi^0 \leq 10^{10}$$

controle inicial $u(\cdot) \equiv 0$ e parâmetro de desenho inicial $\xi = 1$.

Na descrição dos problemas, v^i é a i-ésima componente escalar do vetor $v = [v^0, v^1, \ldots, v^{m-1}]^T \in \mathbb{R}^m$ e c_m é um vetor constante com todas as suas entradas iguais a $c \in \mathbb{R}$.

Todos os problemas-teste só têm restrições de igualdade nas variáveis de estado final x_N e restrições de caixa do tipo $a \leq u \leq b$, com $a, b \in \mathbb{R}^{n_u(N+1)}$. Para satisfazer as restrições de igualdade $h(x_N) = 0$, aplicamos a estratégia clássica de penalização da função de perda quadrática: somamos o termo $\rho\|h(x_N)\|_2^2$ à função objetivo de (6.6) e resolvemos os subproblemas

$$\text{Minimizar } W(T_0, T_f, x_N, u_N, \xi) + \rho\|h(x_N)\|_2^2,$$

Grupo	Problema	n	n_{ocp}
1	1.a	201	804
	1.b	101	404
	1.c	101	404
	2	31	124
	3	161	483
2	4	41	123
	5	41	164
	6	162	567
3	7	21	85
	8	203	607
	9	163	568
	10	127	509
	11	161	484

Tabela 6.2: Grupos de problemas-teste

com restrições de caixa, para valores crescentes de $\rho \in \{10, 100, 100, \ldots\}$ até obter $\|h(x)\|_2 \leq 10^{-5}$. Neste caso, o cálculo de h e suas derivadas parciais h_{x_N} é incluído na subrotina que calcula a função objetivo penalizada. Como os limitantes para as variáveis de controle são considerados explicitamente no processo de minimização, a função para o cálculo de q, q_{x_i}, q_{u_i} e q_ξ não é necessária. Os problemas do Grupo 3 foram tratados como sugere-se em (6.20)–(6.22). Desta forma, aumentamos as restrições de caixa (6.22) a estes problemas. Às vezes, para evitar minimizadores locais do novo parâmetro de desenho ξ^{n_ξ}, com $\xi^{n_\xi} = 0$ (excluindo o caso particular no qual o estado inicial satisfaz as restrições no estado final), é necessário introduzir uma constante pequena $\delta \in \mathbb{R}$ e substituir (6.22) por

$$0 < \delta \leq \xi^{n_\xi} < +\infty. \tag{6.25}$$

Se $\xi^{n_\xi} = \delta$ na solução, diminuímos δ e reiniciamos o processo de minimização. Reportamos os resultados de aplicar esta estratégia uma vez só, com $\delta = 10^{-16}$ no Problema 9, e com $\delta = 0.2$ nos Problemas 10 e 11.

Para discretizar os problemas, utilizamos o método de Runge-Kutta de ordem 4 com tamanho do passo de integração $\Delta t = 0.05$, exceto para o Problema 2 onde utilizamos $\Delta t = 0.03$. O critério de parada foi $\|g_P(x_k)\|_2 \leq \epsilon$ com $\epsilon = 10^{-6}$, $\epsilon = 10^{-5}$ e $\epsilon = 10^{-3}$ para os Grupos 1, 2 e 3 respectivamente.

Todos os experimentos foram rodados numa SPARCstation Sun Ultra 1, com um processador UltraSPARC de 64 bits, *clock* de 167-MHz e 128-MBytes de memória RAM. Os códigos do SPG

Problema	ρ	IT	FGE	LS	Tempo	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
4	10^1	14	15	0	0.02	2.967166e+00	2.2062e−04	8.4623e−06
	10^2	1	4	1	0.00	2.971137e+00	2.2043e−06	8.0205e−06
	10^3	1	5	1	0.00	2.971534e+00	2.2041e−08	8.0924e−06
	10^4	1	6	1	0.00	2.971573e+00	2.2040e−10	8.1007e−06
	10^5	1	7	1	0.00	2.971577e+00	2.2040e−12	8.1015e−06
5	10^1	5	6	0	0.00	3.464720e+01	2.7065e−01	8.1510e−06
	10^2	7	8	0	0.00	3.965662e+01	3.1239e−03	1.6879e−07
	10^3	6	8	1	0.02	4.022061e+01	3.1725e−05	2.5631e−07
	10^4	6	9	1	0.00	4.027773e+01	3.1775e−07	3.6015e−07
	10^5	6	10	1	0.00	4.028345e+01	3.1780e−09	4.1471e−07
	10^6	6	11	1	0.00	4.028403e+01	3.1780e−11	4.7385e−07
6	10^1	9	10	0	0.05	1.961230e+01	6.2346e−04	6.3842e−07
	10^2	12	16	2	0.07	1.962353e+01	6.2567e−06	6.7946e−06
	10^3	12	17	2	0.07	1.962465e+01	6.2589e−08	7.2625e−06
	10^4	12	18	2	0.05	1.962477e+01	6.2592e−10	7.5989e−06
	10^5	12	19	2	0.08	1.962478e+01	6.2593e−12	8.0433e−06

Tabela 6.3: Soluções intermediárias do SPG1 para os problemas do Grupo 2.

estão na linguagem C/C++ e foram compilados com o compilador g++ (GNU project C and C++ compiler v2.7) utilizando a opção de otimização do compilador -O4. Para os métodos SPG, utilizamos $\gamma = 10^{-4}$, $\lambda_{min} = 10^{-16}$, $\lambda_{max} = 10^{16}$ (exceto para o Problema 1 onde utilizamos $\lambda_{max} = 10$), $\sigma_1 = 0.1$, $\sigma_2 = 0.9$ e $\lambda_0 = 1/\|g_P(x_0)\|_2$. Para escolher o parâmetro K, testamos SPG2 no Problema 8, com K variando de 10 a 15. Escolhemos $K = 13$ porque produziu, para o Problema 8, a solução com o menor $\|h(x_N)\|_2$. Porém, vários testes mostraram que as soluções encontradas e o tempo computacional não mudam substancialmente para diversos valores de K maiores do que 5.

As Tabelas 6.3 e 6.4 mostram, para os problemas nos Grupos 2 e 3 respectivamente, o comportamento de SPG1 na resolução de cada subproblema. As Tabelas 6.5 e 6.6 mostram os mesmos resultados para SPG2. Reportamos o número de iterações (IT), o número de avaliações de função e gradiente (FGE), o número de buscas lineares (LS), o tempo de CPU em segundos (Tempo), o melhor valor de função ($\varphi(x)$), a perda quadrática ($\|h(x_N)\|_2^2$) e a norma euclideana do gradiente contínuo projetado ($\|g_P(x)\|_2$). Finalmente, as Tabelas 6.7 e 6.8 resumem o comportamento dos métodos SPG para o conjunto completo de problemas.

Em todos os casos, exceto nos Problemas 3 e 9, as soluções encontradas coincidem, com ínfimas diferenças, com as soluções reportadas em [62]. Atribuímos a diferença na solução do Problema 3

Problema	ρ	IT	FGE	LS	Tempo	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
7	10^1	12092	16004	2715	8.58	7.941602e+00	2.9067e−03	7.8372e−04
	10^2	5404	7279	1163	3.85	7.994073e+00	2.9703e−05	9.5701e−04
	10^3	6800	9256	1521	4.97	7.999407e+00	2.9748e−07	7.8681e−04
	10^4	4123	5590	992	3.07	7.999964e+00	4.0338e−09	9.0854e−04
	10^5	2828	3880	746	2.08	8.000033e+00	4.3174e−11	9.9281e−04
8	10^1	341	456	60	3.72	1.020345e+01	4.2866e−03	5.7982e−04
	10^2	224	298	35	2.35	1.028079e+01	4.3355e−05	9.6581e−04
	10^3	53	63	7	0.50	1.028859e+01	4.3425e−07	9.9601e−04
	10^4	381	570	68	4.50	1.028937e+01	4.3430e−09	9.9511e−04
	10^5	48	62	7	0.53	1.028945e+01	4.3432e−11	9.9849e−04
9	10^1	252	488	59	1.85	1.697346e+00	4.2441e−04	7.1244e−04
	10^2	197	322	52	1.28	1.704994e+00	4.3243e−06	2.8364e−04
	10^3	77	124	21	0.45	1.705763e+00	4.4307e−08	8.5985e−04
	10^4	168	269	42	1.03	1.705841e+00	4.4559e−10	7.3905e−04
	10^5	86	139	19	0.52	1.705849e+00	4.2509e−12	9.3798e−04
10	10^1	31	32	0	0.10	1.256000e+00	5.5674e−04	9.0142e−04
	10^2	20	33	5	0.12	1.256000e+00	5.3111e−04	8.8188e−04
	10^3	710	1265	198	3.80	1.418484e+00	3.3044e−05	9.9777e−04
	10^4	3439	6563	903	19.60	1.476615e+00	2.9792e−07	9.7732e−04
	10^5	2136	4221	577	12.55	1.481943e+00	2.9576e−09	9.6172e−04
	10^6	2682	5231	719	15.95	1.482473e+00	2.9473e−11	9.0517e−04
11	10^1	557	1046	150	2.80	4.095464e+00	3.2712e−03	8.4463e−04
	10^2	1005	1992	270	5.40	4.155792e+00	3.4935e−05	9.8595e−04
	10^3	928	1867	254	5.12	4.162059e+00	3.4956e−07	9.8441e−04
	10^4	998	1964	277	5.30	4.162686e+00	3.5802e−09	9.5445e−04
	10^5	475	910	130	2.48	4.162751e+00	3.6215e−11	9.9809e−04

Tabela 6.4: Soluções intermediárias do SPG1 para os problemas do Grupo 3.

Problema	ρ	IT	FGE	LS	Tempo	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
4	10^1	14	15	0	0.02	2.967166e+00	2.2062e−04	8.4623e−06
	10^2	1	4	1	0.00	2.971137e+00	2.2043e−06	8.0205e−06
	10^3	1	5	1	0.02	2.971534e+00	2.2041e−08	8.0924e−06
	10^4	1	6	1	0.00	2.971573e+00	2.2040e−10	8.1007e−06
	10^5	1	7	1	0.02	2.971577e+00	2.2040e−12	8.1015e−06
5	10^1	5	6	0	0.00	3.464720e+01	2.7065e−01	8.1510e−06
	10^2	7	8	0	0.00	3.965662e+01	3.1239e−03	1.6879e−07
	10^3	6	8	1	0.00	4.022061e+01	3.1725e−05	2.5631e−07
	10^4	6	9	1	0.00	4.027773e+01	3.1775e−07	3.6016e−07
	10^5	6	10	1	0.00	4.028345e+01	3.1780e−09	4.1471e−07
	10^6	6	11	1	0.00	4.028403e+01	3.1780e−11	4.7385e−07
6	10^1	9	10	0	0.03	1.961230e+01	6.2346e−04	6.3842e−07
	10^2	12	16	2	0.07	1.962353e+01	6.2567e−06	6.7946e−06
	10^3	12	17	2	0.07	1.962465e+01	6.2589e−08	7.2625e−06
	10^4	12	18	2	0.08	1.962477e+01	6.2592e−10	7.5989e−06
	10^5	12	19	2	0.07	1.962478e+01	6.2593e−12	8.0433e−06

Tabela 6.5: Soluções intermediárias do SPG2 para os problemas do Grupo 2.

a um possível erro de impressão em [62]. Outra possibilidade para estas diferenças é a escolha de diferentes métodos de integração e tamanhos de passo. Só com uma exceção, os métodos SPG acharam soluções com ínfimas diferenças. No Problema 9, SPG2 achou a melhor solução com $\varphi(x) = 0.609295$, enquanto SPG1 achou um ponto estacionário com $\varphi(x) = 1.705849$ (o mesmo valor reportado em [62]). Ambos métodos satisfizeram o critério de parada em todos os casos e não houve uma diferença significativa nos seus comportamentos.

6.5 Conclusões

Neste trabalho mostramos como aplicar a metodologia introduzida em [47] a problemas de controle utilizando métodos da família Runge-Kutta. Um enfoque equivalente pode ser aplicado associado a outros métodos de integração como, por exemplo, Newton-Cotes e Adams-Moulton (ver [108] e suas numerosas referências [23, 48, 49, 50, 105]). A mesma ressalva deve ser feita com respeito às técnicas de otimização. No lugar dos métodos SPG, a Subrotina 2.1 pode ser combinada com outros algoritmos de otimização como TNBOX [52] e LANCELOT [34], baseados em regiões de confiança e um enfoque Quase-Newton. Veja [40] para uma comparação entre estes dois códigos para minimização com restrições de caixa e o Capítulo 5 para uma comparação entre

Problema	ρ	IT	FGE	LS	Tempo	$\varphi(x)$	$\|h(x)\|_2^2$	$\|g_P(x)\|_2$
7	10^1	10278	13437	2508	7.17	7.941438e+00	2.9232e−03	8.4046e−04
	10^2	8467	10812	1820	5.77	7.994078e+00	2.9648e−05	5.4158e−04
	10^3	8979	11478	2000	6.10	7.999407e+00	2.9698e−07	9.0763e−04
	10^4	4403	5711	1052	3.12	7.999974e+00	3.9434e−09	9.5398e−04
	10^5	2828	3725	724	1.97	8.000042e+00	4.4236e−11	7.8326e−04
8	10^1	578	748	128	5.87	1.020343e+01	4.2887e−03	4.8833e−04
	10^2	386	505	82	4.00	1.028079e+01	4.3383e−05	9.7696e−04
	10^3	177	230	35	1.82	1.028859e+01	4.3396e−07	4.4132e−04
	10^4	46	58	8	0.45	1.028937e+01	4.3394e−09	6.6745e−04
	10^5	44	59	8	0.45	1.028944e+01	4.3424e−11	5.9739e−04
9	10^1	13836	26253	3500	101.20	6.024155e−01	3.4902e−04	9.7981e−04
	10^2	178	338	51	1.37	6.086081e−01	3.4235e−06	8.3238e−04
	10^3	105	200	30	0.82	6.092257e−01	3.5731e−08	8.9670e−04
	10^4	169	330	56	1.33	6.092885e−01	3.7623e−10	9.6193e−04
	10^5	76	159	26	0.65	6.092949e−01	3.8523e−12	9.9513e−04
10	10^1	31	32	0	0.10	1.256000e+00	5.5674e−04	9.0142e−04
	10^2	18	23	4	0.07	1.256000e+00	5.3200e−04	8.8235e−04
	10^3	842	1338	238	4.03	1.418450e+00	3.3106e−05	9.9166e−04
	10^4	3628	6227	1049	18.82	1.476621e+00	2.9804e−07	9.2118e−04
	10^5	2733	4749	791	14.40	1.481954e+00	2.9200e−09	9.4879e−04
	10^6	2050	3598	589	11.05	1.482480e+00	2.9356e−11	9.6731e−04
11	10^1	1102	1915	347	4.93	4.095515e+00	3.2884e−03	7.6420e−04
	10^2	1345	2560	449	6.53	4.155787e+00	3.4923e−05	9.8835e−04
	10^3	1144	2204	343	5.67	4.162045e+00	3.5268e−07	9.5838e−04
	10^4	1277	2567	415	6.55	4.162674e+00	3.5396e−09	9.7592e−04
	10^5	1420	2825	433	7.20	4.162737e+00	3.5053e−11	9.8648e−04

Tabela 6.6: Soluções intermediárias do SPG2 para os problemas do Grupo 3.

Problema	IT	FGE	LS	Tempo	$\varphi(x)$	$\|g_P(x)\|_2$
1.a	434	558	77	2.62	2.141775e+01	2.3322e−07
1.b	87	95	6	0.20	2.621363e+00	6.5584e−07
1.c	43	44	0	0.12	4.340875e+00	4.1008e−07
2	48	50	1	0.02	9.519459e−01	7.6128e−07
3	1	19	1	0.08	−1.958250e+00	4.5696e−08
4	18	37	4	0.02	2.971577e+00	8.1015e−06
5	36	52	4	0.02	4.028403e+01	4.7385e−07
6	57	80	8	0.32	1.962478e+01	8.0433e−06
7	31249	42009	7137	22.55	8.000033e+00	9.9281e−04
8	1047	1449	177	11.60	1.028945e+01	9.9849e−04
9	780	1342	193	5.13	1.705849e+00	9.3798e−04
10	9018	17345	2402	52.12	1.482473e+00	9.0517e−04
11	3963	7779	1081	21.10	4.162757e+00	9.9867e−04

Tabela 6.7: Comportamento de SPG1.

Problema	IT	FGE	LS	Tempo	$\varphi(x)$	$\|g_P(x)\|_2$
1.a	500	654	96	3.43	2.141775e+01	3.2025e−07
1.b	87	95	6	0.22	2.621363e+00	6.5576e−07
1.c	43	44	0	0.12	4.340875e+00	4.1008e−07
2	49	52	2	0.03	9.519459e−01	6.3637e−07
3	1	12	1	0.08	−1.997609e+00	4.4563e−08
4	18	37	4	0.06	2.971577e+00	8.1015e−06
5	36	52	4	0.00	4.028403e+01	4.7385e−07
6	57	80	8	0.32	1.962478e+01	8.0433e−06
7	34955	45163	8104	24.13	8.000042e+00	7.8326e−04
8	1231	1600	261	12.59	1.028944e+01	5.9739e−04
9	14364	27280	3663	105.37	6.092949e−01	9.9513e−04
10	9302	15967	2671	48.47	1.482480e+00	9.6731e−04
11	6288	12071	1987	30.88	4.162737e+00	9.8129e−04

Tabela 6.8: Comportamento de SPG2.

TNBOX e os métodos SPG. Veja também [101] para uma comparação entre o Gradiente Espectral e Gradientes Conjugados para minimização irrestrita. Mais ainda, no lugar de considerar métodos de penalização, muitos outros métodos de programação não-linear podem ser utilizados (Lagrangeano aumentado, linearização, métodos do tipo Newton, técnicas de pontos interiores, etc.). Há muitas formas de considerar as restrições (6.5). Se utilizarmos técnicas de minimização seqüencial (como métodos de penalização), parte das restrições (como por exemplo restrições de caixa) podem ser consideradas explicitamente no processo de minimização enquanto outras restrições podem ser penalizadas. É importante mencionar que, em todos os casos, a função auxiliar (6.7) e as fórmulas canônicas (6.8)–(6.11) podem ser utilizadas para calcular as derivadas totais e derivadas de ordem maior.

Testamos o comportamento dos métodos SPG para a resolução de problemas de controle ótimo porque, como mencionamos no capítulo anterior, "resulta um pouco surpreendente que uma ferramenta tão simples seja competitiva com algoritmos elaborados que utilizam subrotinas e procedimentos numéricos amplamente testados". Como nossos experimentos numéricos mostraram, os métodos de gradientes espectrais projetados combinados com diferenciação automática são, de fato, ferramentas muito úteis para resolver problemas de controle ótimo.

Finalmente, apresentamos um conjunto de problemas e informação detalhada do comportamento dos métodos SPG para a sua resolução. Esta informação pode ser utilizada para futuras comparações no desenvolvimento de novos *softwares* para resolver problemas de controle ótimo.

Capítulo 7

Um método de Gradientes Conjugados Espectrais para Minimização sem Restrições

7.1 Introdução

Num trabalho recente [101], Raydan introduziu o método de gradientes espectrais (SGM) para a minimização de problemas de grande porte sem restrições. As principais características deste método são que só direções de gradiente são utilizadas em cada busca linear e que uma estratégia não-monótona garante convergência global. Surpreendentemente, este método supera algoritmos sofisticados de gradientes conjugados em muitos problemas. Os resultados numéricos obtidos em [56, 82, 101] e no Capítulo 4, dentre outros, nos sugeriram que as idéias de gradientes espectrais e gradientes conjugados poderiam ser combinadas obtendo, assim, algoritmos ainda mais eficientes.

Suponhamos que $f : \mathbb{R}^n \to \mathbb{R}$ tem derivadas parciais contínuas e denotemos $g(x) = \nabla f(x)$. No método de minimização considerado neste capítulo, os iterandos são obtidos por meio de

$$x_{k+1} = x_k + \alpha_k d_k,$$

e

$$d_{k+1} = -\theta_k g_{k+1} + \beta_k s_k \tag{7.1}$$

para $k = 0, 1, 2, \ldots$, onde g_k denota $g(x_k)$, $x_0 \in \mathbb{R}^n$ é arbitrário e

$$d_0 = -\theta_0 g_0.$$

Suponhamos que x_k e x_{k+1} são dois pontos consecutivos e denotemos $s_k = x_{k+1} - x_k = \alpha_k d_k$ e $y_k = g_{k+1} - g_k$. Suponhamos por um momento que f é quadrática e $H \equiv \nabla^2 f(x)$ é definida positiva. Isto implica que $y_k \neq 0$. Portanto, o verdadeiro minimizador x_* satisfaz

$$x_* = x_{k+1} + d_*,$$

onde

$$Hd_* = -g_{k+1}.$$

Pré-multiplicando por s_k^T, obtemos

$$s_k^T H d_* = -s_k^T g_{k+1},$$

o qual implica que

$$y_k^T d_* = -s_k^T g_{k+1}.$$

Portanto, o hiperplano

$$\mathcal{H}_k \equiv \{d \in \mathbb{R}^n \mid y_k^T d = -s_k^T g_{k+1}\}$$

contém o incremento ótimo d_* tal que $x_* = x_{k+1} + d_*$. Observemos que a direção nula $d = 0$ pertence a $\mathcal{H}$ só se $s_k^T g_{k+1} = 0$, o que não é de modo algum nossa hipótese.

Assim, é natural impor que a direção de busca d_{k+1}, calculada por um algoritmo idealizado para minimizar f, satisfaça

$$d_{k+1} \in \mathcal{H}_k. \tag{7.2}$$

Por (7.1),

$$\beta_k = \frac{(\theta_k y_k - s_k)^T g_{k+1}}{s_k^T y_k}. \tag{7.3}$$

Para $\theta_k = 1$ esta fórmula foi introduzida por Perry em [96]. Supondo que $s_j^T g_{j+1} = 0$, para $j = 0, 1, \ldots, k$, obtemos

$$\beta_k = \frac{\theta_k y_k^T g_{k+1}}{\alpha_k \theta_{k-1} g_k^T g_k}. \tag{7.4}$$

Se $\theta_k = \theta_{k-1} = 1$ esta é a clássica fórmula de Polak-Ribière. Finalmente, se também supusermos que os gradientes sucessivos são ortogonais, obtemos a generalização da fórmula de Fletcher-Reeves:

$$\beta_k = \frac{\theta_k g_{k+1}^T g_{k+1}}{\alpha_k \theta_{k-1} g_k^T g_k}. \tag{7.5}$$

Neste trabalho, motivados pelo sucesso do método dos gradientes espectrais, decidimos comparar a escolha clássica $\theta_k = 1$ com a escolha espectral:

$$\theta_k = s_k^T s_k / s_k^T y_k. \tag{7.6}$$

De fato, as direções $d_k = -\theta_k g_k$ são as utilizadas por Raydan no seu método de gradientes espectrais. O parâmetro θ_k dado por (7.6) e a inversa do quociente de Rayleigh

$$s_k^T [\int_0^1 \nabla^2 f(x_k + t s_k) dt] s_k / s_k^T s_k$$

o qual, obviamente, está entre o maior e o menor autovalor da Hessiana média $\int_0^1 \nabla^2 f(x_k + t s_k) dt$.

Mais ainda, depois de algumas experimentações numéricas, observamos que a primeira tentativa para a escolha do comprimento do passo α_k é também um parâmetro muito importante que influencia o comportamento do algoritmo. Portanto, decidimos testar duas alternativas diferentes para esta escolha.

Este capítulo está organizado como se segue. Na Seção 7.2, apresentamos o algoritmo modelo, introduzindo todas as características essenciais de sua implementação. Na Seção 7.3, utilizamos o conjunto de problemas teste dado em [101] para responder as seguintes questões:

1. A escolha (7.6) é melhor que $\theta_k \equiv 1$?
2. Qual é a melhor escolha para β_k, dentre (7.3), (7.4) e (7.5)?
3. Qual é a melhor escolha inicial para o comprimento do passo?

Na Seção 7.4, comparamos o novo algoritmo com CONMIN (um código popular de gradientes conjugados baseado em [106, 103]) e com o método de gradientes espectrais de Raydan, utilizando as mesmas funções teste da Seção 7.3. Na Seção 7.5, comparamos o novo algoritmo com o método de gradientes espectrais utilizando o problema de estimação em ótica descrito no Capítulo 4. As conclusões são apresentadas na Seção 7.6.

7.2 O algoritmo

Tendo em mente as definições de g_k, s_k e y_k dadas na Introdução, definimos o Método dos Gradientes Conjugados Espectrais (SCG) como se segue:

Algoritmo SCG

Seja $x_0 \in \mathbb{R}^n$ e sejam $0 < \sigma < \gamma < 1$.

Passo 0: Definir $d_0 = -g_0$ e iniciar $k \leftarrow 0$.

Passo 1: Se $g_k = 0$, terminar a execução do algoritmo.

Passo 2: Calcular (tentando primeiro $\alpha = \bar{\alpha}(k, d_k, d_{k-1}, \alpha_{k-1})$) $\alpha > 0$ tal que

$$f(x_k + \alpha d_k) \leq f(x_k) + \sigma \alpha g_k^T d_k \tag{7.7}$$

e

$$g(x_k + \alpha d_k)^T d_k \geq \gamma g_k^T d_k. \tag{7.8}$$

Definir $\alpha_k = \alpha$ e

$$x_{k+1} = x_k + \alpha_k d_k.$$

Passo 3: Calcular θ_k por (7.6) (ou $\theta_k = 1$) e β_k por (7.3), (7.4) ou (7.5).

Definir

$$d = -\theta_k g_{k+1} + \beta_k s_k. \tag{7.9}$$

Se

$$d^T g_{k+1} \leq -10^{-3} \|d\| \|g_{k+1}\| \tag{7.10}$$

definir $d_{k+1} = d$. De outra forma, definir

$$d_{k+1} = -\theta_k g_{k+1}.$$

Passo 4: Fazer $k \leftarrow k + 1$ e ir ao Passo 2.

É bem conhecido (ver [39, 51]) que um comprimento de passo α satisfazendo (7.7) e (7.8) sempre existe se f é limitada inferiormente ao longo da direção d_k. Assumimos que temos um algoritmo que calcula α com aquelas condições ou detecta que f não é limitada inferiormente. Nesse caso, dissemos que SCG para na iteração k. Na prática, adotamos a busca linear unidimensional utilizada em CONMIN (ver [106]) para calcular α.

A direção de busca d calculada por (7.9) pode não ser uma direção de descida. Essa é a razão que motivou várias modificações da fórmula de Perry em [106]. Quando o ângulo entre d e $-g_{k+1}$ não é suficientemente agudo "reiniciamos" o algoritmo com a direção do gradiente espectral $-\theta_k g_{k+1}$. Razões mais sofisticadas para reiniciar têm sido propostas na literatura, mas nós estamos interessados no comportamento de um algoritmo que utilize este critério "ingênuo", associado como a escolha espectral para a reinicialização. Claramente, o coeficiente θ_k é sempre positivo e está sempre bem definido pois (7.8) implica que $s_k^T y_k > 0$.

7.3 Discussão das alternativas

Nesta seção utilizamos os problemas teste considerados em [101] para responder as questões formuladas na Introdução. Com este propósito, consideramos o algoritmo SGC com $\sigma = 10^{-4}$ e $\gamma = 0.5$.

Assim, para cada escolha de β_k (dentre Perry (7.3), Polak-Ribière (7.4) e Fletcher-Reeves (7.5)) temos quatro métodos:

M1: θ_k é calculado por (7.6) e a escolha inicial de α é

$$\bar{\alpha}(k, d_k, d_{k-1}, \alpha_{k-1}) = \begin{cases} 1, & \text{se } k = 0 \\ \alpha_{k-1}\|d_{k-1}\|_2/\|d_k\|_2, & \text{caso contrário;} \end{cases} \tag{7.11}$$

M2: θ_k é calculado por (7.6) e $\bar{\alpha}(k, d_k, d_{k-1}, \alpha_{k-1}) \equiv 1$;

M3: $\theta_k \equiv 1$ e o α inicial é calculado como em (7.11);

M4: $\theta_k \equiv 1$ e $\bar{\alpha}(k, d_k, d_{k-1}, \alpha_{k-1}) \equiv 1$.

Nas tabelas seguintes, mostramos o comportamento dos métodos acima descritos. Para cada algoritmo declaramos o número de avaliações de função e gradiente (FGE) e o valor da função objetivo na solução encontrada ($f(x)$). Para terminar as simulações, utilizamos, como em [101], o critério

$$\|g(x_k)\|_2 \leq 10^{-6} \text{ máx}\{1, |f(x_k)|\}.$$

Todos os experimentos foram rodados numa SPARCstation Sun Ultra 1, com um processador UltraSPARC de 64 bits, *clock* de 167-MHz e 128-MBytes de memória RAM. Os códigos de SGM e CONMIN estão em Fortran e foram compilados com o compilador f77 (SC 1.0 Fortran v1.4). Os outros algoritmos estão na linguagem C/C++ e foram compilados com o compilador g++ (GNU project C and C++ compiler v2.7). Em todos os casos utilizamos a opção de otimização do compilação -O4.

A Tabela 7.1 corresponde às quatro alternativas da fórmula (7.3). As Tabelas 7.2 e 7.3 correspondem as fórmulas (7.4) e (7.5) respectivamente. Mostramos, para cada problema, o número de avaliações de função e gradiente e o melhor valor da função objetivo encontrado. Seja f_i o melhor valor de f encontrado pelo método M_i e f_j o melhor valor de f encontrado por M_j. Dizemos que, para um problema em particular, o comportamento de M_i foi melhor que o comportamento de M_j se $f_i \leq f_j - 10^{-3}$ ou se $|f_i - f_j| < 10^{-3}$ e o número de avaliações de função

e gradiente de M_i foi menor que o número de avaliações de função e gradiente de M_j. Dizemos que "M_i venceu M_j $k_1 - k_2$" se o comportamento de M_i foi melhor que o comportamento de M_j em k_1 problemas enquanto que o comportamento de M_j foi melhor que o comportamento de M_i em k_2 problemas.

As conclusões dos experimentos mostrados acima são:

Para a fórmula de Perry's (Tabela 7.1)

- M1 venceu M3 $21 - 12$;
- M1 venceu M2 $26 - 14$;
- M1 venceu M4 $31 - 9$;
- M3 venceu M2 $23 - 17$;
- M3 venceu M4 $31 - 9$;
- M2 venceu M4 $27 - 8$.

Para a fórmula de Polak-Ribière (Tabela 7.2)

- M3 venceu M1 $9 - 8$;
- M3 venceu M2 $26 - 14$;
- M3 venceu M4 $32 - 8$;
- M1 venceu M2 $27 - 13$;
- M1 venceu M4 $33 - 7$;
- M2 venceu M4 $31 - 8$.

Para a fórmula de Fletcher-Reeves (Tabela 7.3)

- M3 venceu M1 $10 - 7$;
- M3 venceu M2 $24 - 16$;
- M3 venceu M4 $29 - 8$;
- M1 venceu M2 $24 - 16$;

		M1		M2		M3		M4	
Problema		FGE	$f(x)$	FGE	$f(x)$	FGE	$f(x)$	FGE	$f(x)$
1	100	11	0.0000E+00	8	0.0000E+00	11	0.0000E+00	8	0.0000E+00
1	1000	11	0.0000E+00	8	0.0000E+00	11	0.0000E+00	8	0.0000E+00
1	10000	11	0.0000E+00	60	0.0000E+00	11	0.0000E+00	60	0.0000E+00
2	100	63	5.0500E+02	79	5.0500E+02	63	5.0500E+02	83	5.0500E+02
2	500	85	1.2525E+04	124	1.2525E+04	95	1.2525E+04	140	1.2525E+04
2	1000	96	5.0050E+04	140	5.0050E+04	90	5.0050E+04	186	5.0050E+04
3	100	11	8.7540E−22	7	1.4665E−24	11	8.7540E−22	7	7.8758E−25
3	1000	9	5.2302E−20	6	2.6191E−22	9	5.2302E−20	6	5.4108E−20
3	10000	43	0.0000E+00	18	0.0000E+00	43	0.0000E+00	13	0.0000E+00
4	100	94	1.8410E−06	117	1.8410E−06	98	1.8410E−06	83	1.8410E−06
4	1000	84	2.3338E−07	121	2.1479E−07	86	2.4019E−07	79	2.4705E−07
4	10000	88	2.2553E−08	114	2.2104E−08	94	2.2561E−08	81	2.2369E−08
5	100	55	3.0248E−15	41	4.7261E−15	56	9.0436E−16	99	7.7355E−15
5	1000	106	1.4078E+00	140	7.1253E−01	70	4.8690E−15	190	3.9707E−01
5	3000	95	3.9707E−01	54	8.0458E−15	113	3.9707E−01	193	3.9707E−01
6	100	59	1.2882E−10	93	2.1347E−10	120	1.7172E−10	385	2.7329E−10
6	1000	221	9.7949E−11	449	2.2576E−10	306	6.3862E−10	1794	7.1612E−10
6	10000	753	1.5824E−10	2669	2.4077E−10	765	3.1394E−10	7879	1.4202E−10
7	100	54	7.1299E−24	110	3.7756E−25	49	1.7519E−20	118	3.8932E−15
7	1000	58	4.2731E−18	144	5.6455E−25	50	1.1695E−16	113	2.8596E−27
7	10000	61	2.2113E−21	91	1.2007E−16	53	2.1468E−17	134	6.4250E−23
8	100	151	9.0249E−04	166	9.0249E−04	110	9.0249E−04	255	9.0249E−04
8	1000	107	9.6868E−03	110	9.6862E−03	81	9.6862E−03	579	9.6862E−03
8	10000	96	9.9002E−02	79	9.9002E−02	98	9.9002E−02	650	9.9002E−02
9	100	180	2.6633E−15	272	2.5811E−15	188	1.5503E−15	278	2.7837E−15
9	1000	659	9.1087E−15	927	1.1960E−15	681	1.2204E−15	934	1.1820E−15
10	100	29	1.0583E−19	22	6.2419E−21	63	5.6617E−14	291	5.7001E−19
10	1000	83	1.6030E−15	204	9.3560E−19	132	1.4792E−15	107	4.8044E+93
11	100	168	3.8005E−10	691	4.5490E−11	131	1.6455E−09	263	2.8000E−09
11	1000	366	1.9973E−09	190	2.0085E−09	117	5.6132E−09	826	2.4485E−10
12	100	727	1.0000E+00	707	1.0000E+00	694	1.0000E+00	3950	1.0000E+00
12	500	1899	1.0000E+00	3414	1.0000E+00	2081	1.0000E+00	17602	1.0000E+00
13	100	32	1.0909E+02	27	1.0909E+02	39	1.0909E+02	45	1.0909E+02
13	1000	31	1.1082E+03	22	1.1082E+03	34	1.1082E+03	48	1.1082E+03
13	10000	23	1.1099E+04	19	1.1099E+04	36	1.1099E+04	49	1.1099E+04
14	100	85	1.1965E+04	129	1.1965E+04	65	1.1965E+04	361	1.1965E+04
14	1000	43	1.2147E+05	77	1.2147E+05	121	1.2147E+05	236	1.2147E+05
14	10000	41	1.2165E+06	96	1.2165E+06	38	1.2165E+06	525	1.2165E+06
15	100	116	6.6990E−16	76	3.7810E+02	87	3.7810E+02	339	7.8770E+00
15	1000	106	3.1328E−15	229	7.0812E−15	77	3.9306E+03	362	3.9228E+03

Tabela 7.1: Comportamento do método de Perry.

		M1		M2		M3		M4	
Problema		FGE	$f(x)$	FGE	$f(x)$	FGE	$f(x)$	FGE	$f(x)$
1	100	13	0.0000E+00	11	0.0000E+00	13	0.0000E+00	11	0.0000E+00
1	1000	13	0.0000E+00	11	0.0000E+00	13	0.0000E+00	166	0.0000E+00
1	10000	13	0.0000E+00	218	0.0000E+00	13	0.0000E+00	166	0.0000E+00
2	100	68	5.0500E+02	79	5.0500E+02	68	5.0500E+02	91	5.0500E+02
2	500	108	1.2525E+04	123	1.2525E+04	108	1.2525E+04	163	1.2525E+04
2	1000	121	5.0050E+04	140	5.0050E+04	122	5.0050E+04	185	5.0050E+04
3	100	11	8.7540E−22	8	4.7974E−19	11	8.7540E−22	95	1.3644E−19
3	1000	9	5.2302E−20	8	3.5315E−21	9	5.2302E−20	89	1.3304E−22
3	10000	43	0.0000E+00	45	0.0000E+00	43	0.0000E+00	216	0.0000E+00
4	100	114	2.4054E−06	123	1.8410E−06	111	2.4054E−06	97	1.8410E−06
4	1000	98	2.3339E−07	108	2.2664E−07	98	2.3339E−07	111	2.1427E−07
4	10000	110	2.2265E−08	122	2.1983E−08	107	2.2265E−08	104	2.2680E−08
5	100	51	9.3293E−15	52	1.1363E−14	51	9.3293E−15	116	5.4871E−15
5	1000	117	7.1253E−01	99	7.1253E−01	115	7.1253E−01	231	7.1253E−01
5	3000	110	3.9707E−01	64	3.8591E−15	112	3.9707E−01	193	3.9707E−01
6	100	75	8.1586E−11	109	4.0205E−10	75	8.1586E−11	361	1.8057E−10
6	1000	271	2.7962E−10	522	3.9881E−10	268	5.1372E−10	1714	8.0629E−11
6	10000	1192	8.5903E−11	3159	5.7281E−10	1064	1.8114E−10	8308	7.1085E−10
7	100	59	2.7563E−16	117	2.0270E−16	59	2.7563E−16	121	1.1097E−22
7	1000	82	3.1513E−15	105	4.5191E−17	79	5.5793E−18	108	1.3089E−15
7	10000	52	8.7075E−17	89	1.6665E−25	52	8.7057E−17	129	4.2441E−18
8	100	183	9.0249E−04	160	9.0249E−04	193	9.0249E−04	359	9.0249E−04
8	1000	155	9.6862E−03	157	9.6862E−03	150	9.6862E−03	626	9.6862E−03
8	10000	100	9.9002E−02	158	9.9002E−02	104	9.9002E−02	840	9.9002E−02
9	100	224	5.5834E−15	261	1.9083E−15	223	1.0253E−14	242	3.7251E−15
9	1000	823	5.3509E−15	936	1.1286E−15	976	9.5998E−15	824	1.6607E−15
10	100	29	1.0512E−19	47	4.4518E−21	29	1.0511E−19	969	1.4175E−25
10	1000	43	2.1838E−23	485	4.9804E−22	43	2.3554E−23	107	1.2014E+76
11	100	329	5.3346E−10	436	4.1324E−09	394	8.2489E−11	310	4.2044E−10
11	1000	1089	4.7035E−09	346	2.4502E−09	607	5.1556E−10	320	2.3313E−09
12	100	840	1.0000E+00	1087	1.0000E+00	914	1.0000E+00	5145	1.0000E+00
12	500	3004	1.0000E+00	4031	1.0000E+00	3062	1.0000E+00	24685	1.0000E+00
13	100	39	1.0909E+02	24	1.0909E+02	39	1.0909E+02	34	1.0909E+02
13	1000	36	1.1082E+03	22	1.1082E+03	36	1.1082E+03	47	1.1082E+03
13	10000	30	1.1099E+04	28	1.1099E+04	30	1.1099E+04	32	1.1099E+04
14	100	76	1.1965E+04	171	1.1965E+04	76	1.1965E+04	261	1.1965E+04
14	1000	62	1.2147E+05	83	1.2147E+05	62	1.2147E+05	376	1.2147E+05
14	10000	62	1.2165E+06	85	1.2165E+06	62	1.2165E+06	296	1.2165E+06
15	100	65	3.8597E+02	80	3.7810E+02	65	3.8597E+02	366	3.9379E+00
15	1000	77	3.9267E+03	70	3.9267E+03	77	3.9267E+03	334	3.9228E+03

Tabela 7.2: Comportamento do método de Polak-Ribière.

Problema		M1		M2		M3		M4	
		FGE	$f(x)$	FGE	$f(x)$	FGE	$f(x)$	FGE	$f(x)$
1	100	13	1.4211E−14	12	0.0000E+00	13	1.4211E−14	13	2.8422E−13
1	1000	65	0.0000E+00	64	0.0000E+00	65	0.0000E+00	65	0.0000E+00
1	10000	117	0.0000E+00	167	0.0000E+00	117	0.0000E+00	116	0.0000E+00
2	100	87	5.0500E+02	129	5.0500E+02	87	5.0500E+02	104	5.0500E+02
2	500	135	1.2525E+04	203	1.2525E+04	135	1.2525E+04	169	1.2525E+04
2	1000	150	5.0050E+04	235	5.0050E+04	150	5.0050E+04	326	5.0050E+04
3	100	11	8.7540E−22	9	1.6231E−19	11	8.7540E−22	99	1.9409E−17
3	1000	9	5.2302E−20	8	3.1504E−20	9	5.2302E−20	89	1.3690E−22
3	10000	43	0.0000E+00	34	0.0000E+00	43	0.0000E+00	317	0.0000E+00
4	100	9237	2.0511E−06	514	1.8410E−06	9064	2.0431E−06	706	1.8410E−06
4	1000	546	2.3349E−07	478	2.2725E−07	490	2.3349E−07	1066	2.2725E−07
4	10000	260	2.1433E−08	474	1.4611E−08	225	2.1434E−08	1026	2.1369E−08
5	100	94	6.1146E−15	88	6.5000E−15	94	6.1146E−15	112	7.3569E−15
5	1000	355	3.9707E−01	483	3.9707E−01	349	3.9707E−01	2482	3.9707E−01
5	3000	361	1.4132E−14	332	1.4085E−14	368	1.3015E−14	243	3.9707E−01
6	100	72	7.3343E−11	83	3.9906E−10	72	7.3343E−11	335	2.8331E−10
6	1000	181	1.5526E−10	611	7.1882E−11	181	1.5532E−10	1764	1.6422E−10
6	10000	712	7.5201E−11	4835	5.7198E−11	754	6.4120E−11	12168	4.3333E−11
7	100	226	3.4249E−14	308	4.7522E−13	210	2.7708E−13	2549	6.3509E−14
7	1000	150	1.6903E−16	2203	8.5476E−14	166	4.9431E−13	4796	2.2730E−25
7	10000	175	1.2559E−14	3254	3.0481E−13	169	2.1257E−14	2305	6.8998E−13
8	100	1651	9.0249E−04	413	9.0249E−04	1636	9.0249E−04	1480	9.0249E−04
8	1000	738	9.6862E−03	246	9.6862E−03	751	9.6862E−03	582	9.6862E−03
8	10000	524	9.9002E−02	3027	9.9002E−02	478	9.9002E−02	1913	9.9002E−02
9	100	531	2.5973E−15	14001	1.2942E−03	531	2.5971E−15	14492	2.5580E−01
9	1000	5050	5.8067E−16	14001	7.2011E+00	4410	4.6703E−16	14217	1.8909E+01
10	100	29	1.0511E−19	55	5.3174E−19	29	1.0512E−19	887	5.0363E−21
10	1000	43	9.7570E−24	503	1.6185E−22	43	1.2717E−23	107	7.1871E+93
11	100	299	4.0649E−10	497	9.4581E−11	423	5.3053E−10	372	5.3573E−10
11	1000	522	1.3232E−09	4286	5.7282E−10	562	3.0749E−09	1550	5.0101E−10
12	100	9584	1.0065E+02	20633	1.0000E+00	9584	1.0065E+02	135737	1.0000E+00
12	500	9991	2.9167E+02	60913	9.0475E+01	9991	2.9167E+02	135008	2.0857E+02
13	100	50	1.0909E+02	42	1.0909E+02	50	1.0909E+02	50	1.0909E+02
13	1000	40	1.1082E+03	52	1.1082E+03	40	1.1082E+03	28	1.1082E+03
13	10000	33	1.1099E+04	19	1.1099E+04	33	1.1099E+04	64	1.1099E+04
14	100	65	1.1965E+04	5430	1.1965E+04	65	1.1965E+04	3115	1.1965E+04
14	1000	22	1.2147E+05	2676	1.2147E+05	22	1.2147E+05	552	1.2147E+05
14	10000	31	1.2165E+06	419	1.2165E+06	31	1.2165E+06	352	1.2165E+06
15	100	5551	3.9379E+00	794	3.7810E+02	9042	3.9379E+00	379	7.8770E+00
15	1000	207	7.8770E+00	709	3.9391E+00	86	7.8770E+00	2677	7.8770E+00

Tabela 7.3: Comportamento do método de Fletcher-Reeves.

- M1 venceu M4 $30 - 7$;
- M2 venceu M4 $28 - 12$.

Agora, comparando as melhores alternativas para cada fórmula de gradientes conjugados, concluímos que:

- Perry (M1) venceu Polak-Ribière (M3) $30 - 6$.
- Perry (M1) venceu Fletcher-Reeves (M3) $29 - 7$.
- Polak-Ribière (M3) venceu Fletcher-Reeves (M3) $23 - 11$.

7.4 Comparação com CONMIN e SGM

Os experimentos da Seção 7.3 parecem indicar que a melhor fórmula de gradientes conjugados espectrais é a fórmula de Perry (7.3) com a escolha espectral (7.6) de θ_k e a escolha inicial (7.11) do comprimento do passo. Logo, comparamos este método com CONMIN [106] e o método do gradiente espectral de Raydan. Utilizamos os códigos originais (em Fortran) de SGM e CONMIN. SGM foi utilizado com os parâmetros recomendados por Raydan [101].

Os resultados são apresentados na Tabela 7.4. Utilizando os mesmos critérios descritos acima, vemos que Perry M1 venceu CONMIN $20 - 19$. A comparação com o SGM de Raydan deve levar em consideração que este método, às vezes, avalia a função em pontos onde o gradiente não é avaliado. Contando só avaliações de gradiente, Perry M1 foi melhor que SGM 20 vezes e SGM foi melhor que Perry M1 no mesmo número de problemas. Contando também avaliações de função, Perry M1 venceu SGM $21 - 19$.

7.5 Um problema de estimação de parâmetros em Ótica

Em trabalhos recentes, o método dos gradientes espectrais tem sido utilizado com sucesso para um difícil problema inverso que consiste na estimação de parâmetros óticos de um filme fino utilizando só dados de transmissão. Ver [28] e o Capítulo 4.

A transmissão T de um filme fino absorvente num substrato transparente é dada por

$$T = \frac{Ax}{B - Cx + Dx^2}, \tag{7.12}$$

onde

$$A = 16s(n^2 + \kappa^2), \tag{7.13}$$

		SGM				CONMIN			Perry(M1)		
Problema		IT	FE	GE	$f(x)$	IT	FGE	$f(x)$	IT	FGE	$f(x)$
1	100	8	8	9	0.0000E+00	15	38	1.6058E−11	5	11	0.0000e+00
1	1000	8	8	9	-1.1369E−13	15	38	1.5154E−10	5	11	0.0000e+00
1	10000	8	8	9	1.8190E−12	15	38	1.4734E−09	5	11	0.0000e+00
2	100	52	57	53	5.0500E+02	40	81	5.0500E+02	45	63	5.0500e+02
2	500	74	80	75	1.2525E+04	63	127	1.2525E+04	67	85	1.2525e+04
2	1000	82	91	83	5.0050E+04	71	145	5.0050E+04	75	96	5.0050e+04
3	100	3	3	4	1.8795E−23	3	7	1.4066E−07	5	11	8.7540e−22
3	1000	4	4	5	1.3346E−23	15	38	8.1381E−18	4	9	5.2302e−20
3	10000	53	56	54	0.0000E+00	14	38	4.6837E−20	5	43	0.0000e+00
4	100	76	80	77	2.4054E−06	51	108	1.8410E−06	62	94	1.8410e−06
4	1000	91	104	92	2.2558E−07	53	112	2.2664E−07	56	84	2.3338e−07
4	10000	89	99	90	2.1659E−08	59	126	2.2674E−08	60	88	2.2553e−08
5	100	34	34	35	1.1369E−14	33	67	3.0081E−14	29	55	3.0248e−15
5	1000	40	40	41	4.3612E−15	81	169	3.9707E−01	72	106	1.4078e+00
5	3000	44	45	45	2.0021E−14	35	71	1.8684E−14	62	95	3.9707e−01
6	100	106	111	107	5.5889E−10	49	99	6.7141E−10	42	59	1.2882e−10
6	1000	296	364	297	1.4567E−09	158	320	1.3921E−10	169	221	9.7949e−11
6	10000	1351	1751	1352	1.0295E−09	464	937	5.3219E−11	565	753	1.5824e−10
7	100	69	91	70	3.4615E−17	19	47	2.9286E−12	29	54	7.1299e−24
7	1000	93	118	94	1.4427E−20	30	73	1.4110E−15	28	58	4.2731e−18
7	10000	70	92	71	1.9663E−17	28	69	1.4479E−14	28	61	2.2113e−21
8	100	48	49	49	9.0249E−04	27	65	9.0249E−04	73	151	9.0249e−04
8	1000	57	57	58	9.6862E−03	25	55	9.6862E−03	47	107	9.6862e−03
8	10000	70	70	71	9.9001E−02	1	3	1.1114E+23	37	96	9.9002e−02
9	100	167	191	168	2.4820E−16	80	161	4.9988E−15	141	180	2.6633e−15
9	1000	878	1152	879	1.6416E−14	306	613	6.1288E−16	520	659	9.1087e−15
10	100	38	38	39	3.1061E−29	13	29	2.8874E−18	11	29	1.0583e−19
10	1000	66	68	67	1.6362E−25	27	62	1.4308E−20	20	83	1.6030e−15
11	100	740	988	741	1.1325E−09	47	95	1.0019E−09	98	168	3.8005e−10
11	1000	1345	1851	1346	7.9283E−09	43	87	2.0417E−09	196	366	1.9973e−09
12	100	1429	1886	1430	1.0000E+00	254	516	1.0000E+00	536	727	1.0000e+00
12	500	4452	5896	4453	1.0000E+00	1082	2180	1.0000E+00	1522	1899	1.0000e+00
13	100	26	26	27	1.0909E+02	13	27	1.0909E+02	16	32	1.0909e+02
13	1000	23	23	24	1.1082E+03	11	23	1.1082E+03	16	31	1.1082e+03
13	10000	21	21	22	1.1099E+04	9	19	1.1099E+04	11	23	1.1099e+04
14	100	438	587	439	1.1965E+04	13	27	1.1965E+04	48	85	1.1965e+04
14	1000	288	391	289	1.2147E+05	12	25	1.2147E+05	22	43	1.2147e+05
14	10000	119	154	120	1.2165E+06	11	23	1.2165E+06	21	41	1.2165e+06
15	100	81	84	82	3.8597E+02	25	53	3.7810E+02	64	116	6.6990e−16
15	1000	80	87	81	7.8770E+00	34	69	3.9267E+03	59	106	3.1328e−15

Tabela 7.4: Comportamento de SGM, CONMIN e Perry M1.

$$B = [(n+1)^2 + \kappa^2][(n+1)(n+s^2) + \kappa^2], \tag{7.14}$$

$$\begin{aligned} C = {} & [(n^2 - 1 + \kappa^2)(n^2 - s^2 + \kappa^2) - 2\kappa^2(s^2+1)]2\cos\varphi \\ & -\kappa[2(n^2 - s^2 + \kappa^2) + (s^2+1)(n^2 - 1 + \kappa^2)]2\sin\varphi, \end{aligned} \tag{7.15}$$

$$D = [(n-1)^2 + \kappa^2][(n-1)(n-s^2) + \kappa^2], \tag{7.16}$$

$$\varphi = 4\pi n d/\lambda, \quad x = exp(-\alpha d), \quad \alpha = 4\pi\kappa/\lambda. \tag{7.17}$$

Nas fórmulas (7.13)–(7.17) a seguinte notação é utilizada:

(a) λ é o comprimento de onda;

(b) $s \equiv s(\lambda)$ é o índice de refração do substrato;

(c) $r \equiv r(\lambda)$ é o índice de refração do filme;

(d) $\kappa \equiv \kappa(\lambda)$ é o coeficiente de atenuação do filme;

(e) d é a espessura do filme.

Assumimos que $s(\lambda)$ e d são conhecidos, um conjunto de dados experimentais $(\lambda_i, T^{obs}(\lambda_i))$, para $i = 1, \ldots, N$, onde $(\lambda_{min} \leq \lambda_i < \lambda_{i+1} \leq \lambda_{max}$ para todo $i = 1, \ldots, N-1)$, é dado e desejamos estimar $r(\lambda)$ e $\kappa(\lambda)$. Numa primeira olhada, este problema parece ser indeterminado. De fato, dado λ, a seguinte equação deve ser satisfeita:

$$T(\lambda, s(\lambda), d, r(\lambda), \kappa(\lambda)) = T^{obs}(\lambda). \tag{7.18}$$

A equação (7.18) tem duas incógnitas $r(\lambda)$ e $\kappa(\lambda)$ e, portanto, em geral, o seu conjunto de soluções é uma curva no espaço bidimensional $(r(\lambda), \kappa(\lambda))$. Porém, restrições físicas reduzem drasticamente o conjunto viável das incógnitas $r(\lambda)$ e $\kappa(\lambda)$. Para a classe de problemas estudada em [28], essas restrições físicas são:

$$r(\lambda_{max}) \geq 1, \quad \kappa(\lambda_{max}) \geq 0, \tag{7.19}$$

$$r'(\lambda_{max}) \leq 0, \quad \kappa'(\lambda_{max}) \leq 0, \tag{7.20}$$

$$r''(\lambda) \geq 0 \quad \text{para todo } \lambda \in [\lambda_{min}, \lambda_{max}], \tag{7.21}$$

$$\kappa''(\lambda) \geq 0 \quad \text{para todo } \lambda \in [\lambda_{min}, \lambda_{max}]. \tag{7.22}$$

Assim, a solução de quadrados mínimos contínuos do problema de estimação é a solução $(r(\lambda), \kappa(\lambda))$ de

$$\text{Minimizar} \int_{\lambda_{min}}^{\lambda_{max}} |T(\lambda, s(\lambda), d, r(\lambda), \kappa(\lambda)) - T^{obs}(\lambda)|^2 d\lambda \tag{7.23}$$

sujeito às restrições (7.19)–(7.22).

No Capítulo 4 definimos

$$r(\lambda_{max}) = 1 + u^2, \quad \kappa(\lambda_{max}) = v^2, \tag{7.24}$$

$$r'(\lambda_{max}) = -u_1^2, \quad \kappa'(\lambda_{max}) = -v_1^2, \tag{7.25}$$

$$r''(\lambda) = w(\lambda)^2 \quad \text{para todo } \lambda \in [\lambda_{min}, \lambda_{max}], \tag{7.26}$$

$$\kappa''(\lambda) = z(\lambda)^2 \quad \text{para todo } \lambda \in [\lambda_{min}, \lambda_{max}]. \tag{7.27}$$

Em situações da vida real, nas quais os dados estão representados por um conjunto de N pontos igualmente espaçados no intervalo $[\lambda_{min}, \lambda_{max}]$, definimos

$$h = (\lambda_{max} - \lambda_{min})/(N-1)$$

e

$$\lambda_i = \lambda_{min} + (i-1)h, \quad i = 1, \ldots, N.$$

O valor observado da transmissão em λ_i será chamado T_i^{obs}. Além disso, utilizamos a notação r_i, κ_i, w_i, z_i da forma óbvia:

$$r_i = r(\lambda_i), \quad \kappa_i = \kappa(\lambda_i),$$

$$w_i = w(\lambda_{i+1}), \quad z_i = z(\lambda_{i+1}),$$

para $i = 1, \ldots, N$. A discretização de (7.24-7.27) nos fornece:

$$r_N = 1 + u^2, \quad v_N = v^2, \tag{7.28}$$

$$r_{N-1} = r_N + u_1^2 h, \quad \kappa_{N-1} = \kappa_N + v_1^2 h, \tag{7.29}$$

$$r_i = w_i^2 h^2 + 2r_{i+1} - r_{i+2}, \quad i = 1, \ldots, N-2, \tag{7.30}$$

$$\kappa_i = z_i^2 h^2 + 2\kappa_{i+1} - \kappa_{i+2} \quad i = 1, \ldots, N-2. \tag{7.31}$$

Finalmente, a função objetivo de (7.23) é aproximada por uma soma de quadrados, resultando o problema de otimização

$$\text{Minimizar} \sum_{i=1}^{N} [T(\lambda_i, s(\lambda_i), d, r_i, \kappa_i) - T_i^{obs}]^2. \tag{7.32}$$

Como r_i e κ_i dependem de u, u_1, v, v_1, w, z através de (7.28–7.31), o problema (7.32) toma a forma

$$\text{Minimizar } f(u, u_1, v, v_1, w_1, \ldots, w_{N-2}, z_1, \ldots, z_{N-2}). \tag{7.33}$$

Nos experimentos, utilizamos os filmes considerados no Capítulo 4, com 100 pontos observados:

Filme 1: Simulação de um filme fino de germânio amorfo depositado num substrato de vidro com $d = 118$nm. $\lambda_{min} = 600$nm e $\lambda_{max} = 2000$nm.

Filme 2: Idêntico ao Filme 1 com $d = 782$nm. $\lambda_{min} = 1000$nm e $\lambda_{max} = 2000$nm.

Filme 3: Simulação de um filme fino de germânio amorfo depositado em um substrato cristalino de silício com $d = 147$nm. $\lambda_{min} = 1250$nm e $\lambda_{max} = 2500$nm.

Filme 4: Idêntico ao Filme 3 com $d = 640$nm. $\lambda_{min} = 640$nm e $\lambda_{max} = 1250$nm.

Filme 5: Simulação de um filme fino silício amorfo hidrogenado depositado em vidro com $d = 624$nm. $\lambda_{min} = 600$nm e $\lambda_{max} = 1600$nm.

Como estimativa inicial de $\kappa(\lambda)$ utilizamos uma função linear por partes cujos valores são 0.1 no menor comprimento de onda do espectro, 0.01 em $\lambda_{min} + 0.2(\lambda_{max} - \lambda_{min})$, e 10^{-10} em λ_{max}. A estimativa inicial de $r(\lambda)$ foi uma função linear variando entre 5 (em λ_{min}) e 3 (em λ_{max}).

Os resultados fisicamente aceitáveis do procedimento de estimação foram obtidos no Capítulo 4 utilizando 30000 iterações do método de gradientes espectrais de Raydan. Aqui, utilizamos com critério de parada para SCG (Perry M1) a desigualdade $f(x_k) < f_{Raydan}$, onde f_{Raydan} e o mínimo valor encontrado utilizando SGM. Na Tabela 7.5 mostramos os resultados. Reportamos o número de iterações (IT), o número de avaliações de função (FE), as avaliações de função e gradiente (FGE) e o tempo de CPU em segundos (Tempo). Observe que SCG atinge a solução de SGM utilizando entre um terço e a metade do tempo computacional utilizado pelo método de gradientes espectrais.

7.6 Conclusões

No trabalho clássico [106], a idéia básica de Perry foi modificada com o intuito de superar o fato de a matriz que, implicitamente, define a direção de busca, não ser definida positiva. Como resultado, a estrutura algorítmica do CONMIN foi obtida. Neste trabalho seguimos uma direção

	SGM				SCG			
Problem	IT	FE	Tempo	f_{Raydan}	IT	FGE	Tempo	$f(x_k)$
1	30000	35825	45.0	6.929605E−07	3605	6184	7.7	6.926210E−07
2	30000	35568	45.8	2.203053E−07	6798	11092	14.1	2.201913E−07
3	30000	38113	47.9	6.224862E−06	7344	13471	17.5	6.224860E−06
4	30000	35687	44.6	1.365270E−06	10356	17938	22.3	1.365184E−06
5	30000	36290	46.3	2.120976E−07	7611	13205	16.5	2.066100E−07

Tabela 7.5: Problemas de Ótica.

diferente, motivados pela necessidade de preservar as agradáveis propriedades geométricas da direção de Perry. Por um lado, observamos que o escalamento do gradiente utilizando o parâmetro espectral de [101] é benéfico, por outro lado, detectamos que a escolha inicial do comprimento do passo é um parâmetro crucial que influencia o comportamento prático do método. Desta forma, o algoritmo de Perry claramente supera a Polak-Ribière e Fletcher-Reeves e é competitivo com CONMIN e o método de Raydan [101].

Mais ainda, como observado por Raydan, o método dos gradientes espectrais certamente depende de uma forma mas dramática do precondicionamento em problemas mal condicionados do que o método de gradientes conjugados. Essa é a razão pela qual, no difícil problema inverso estudado na Seção 7.5, SGM foi superado pela versão M1 do método de Perry.

Capítulo 8

Conclusões

Neste trabalho abordamos as principais técnicas de diferenciação automática e desenvolvemos uma ferramenta-teste que implementa as idéias básicas. No Capítulo 2 descrevemos os conceitos básicos e realizamos alguns testes para verificar as propriedades teóricas de cada método. No Capítulo 3 testamos a diferenciação automática combinada com um método de otimização, o método de Newton. Basicamente, utilizamos ADIC++ para calcular as linhas de Jacobianos esparsos pelo modo reverso de DA. Assim, reproduzimos parte do trabalho desenvolvido em [60] com uma única diferença: não calculamos uma única derivada à mão. Poderíamos ter utilizado qualquer outro esquema, no lugar de diferenças centrais, para discretizar a formulação do problema. Por exemplo, uma possibilidade poderia ter sido aplicar técnicas de *upwinding* para escolher, em cada ponto da malha de discretização, o esquema de aproximação apropriado. E isso não teria complicado em nada o cálculo das derivadas.

Porém, ADIC++ é uma ferramenta-teste e as suas possibilidades são limitadas. Principalmente no formato de entrada e na utilização das estruturas de dados com memória dinâmica. Assim, convencidos das grandes vantagens da utilização da diferenciação automática em otimização, apontamos com um dos trabalhos que darão continuidade a este, o desenvolvimento de uma ferramenta de diferenciação automática. Esta é uma tarefa laboriosa e o desenvolvimento de uma ferramenta eficiente envolve cientistas de várias áreas, tais como: Otimização Discreta, para a elaboração de heurísticas para o aproveitamento de expressões comuns; Computação Científica, para a manipulação de dados (uma linguagem para expressar as funções e estruturas de dados eficientes para a diferenciação); e Matemática Aplicada, para o aperfeiçoamento das técnicas de diferenciação automática existentes e sua aplicação a problemas práticos.

Nos Capítulos 5 e 7 desenvolvemos o Método do Gradiente Espectral Projetado (SPG) e o

Método de Gradientes Conjugados Espectrais (SCG) que, como seus nomes indicam, só utilizam a informação dos gradientes para gerar direções de descida. Nos Capítulos 4 e 7 aplicamos o Método do Gradiente Espectral [101] e o SCG para resolver um problema de estimação de constantes óticas. No Capítulo 6 utilizamos o SPG para resolver problemas de controle ótimo. Em todos os casos, utilizamos o modo reverso de diferenciação automática para o cálculo dos gradientes.

A utilização da diferenciação automática com o método de Newton no Capítulo 3 foi só um exercício. As aplicações aos problemas de Ótica e de Controle Ótimo foram diferentes. Num problema como o dos filmes, o cálculo manual das derivadas pode levar várias horas e, no fim, quem garante que esteja certo? Assim é preciso fazer um programa simples para testar com diferenças finitas, corrigir o código se for necessário e assim por diante. Mais ainda, dificilmente estarão sendo reaproveitados todos os cálculos possíveis. Neste caso em particular, testamos o código que utiliza o modo reverso de DA contra uma subrotina gerada à mão por um programador "muito experiente". O código de DA foi dez vezes mais rápido e permitiu encontrar um erro quase imperceptível no cálculo manual.

Já no caso dos problemas de controle ótimo foi mais interessante ainda. Em geral, as funções objetivo dependem das variáveis de controle e das variáveis de estado no tempo final. Um exemplo típico deste tipo de problemas é minimizar o combustível necessário para levar um foguete á Lua, onde as variáveis de controle representam o combustível e a posição da Lua impõe as restrições no estado final. Concentremos-nos na dependência das variáveis de estado no ponto final. Estas variáveis de estado, por sua vez dependem das variáveis de controle e do estado inicial. Em geral, esta dependência está representada por uma equação diferencial. Na prática, integramos essa equação diferencial por um método numérico como, por exemplo, um método da família Runge-Kutta. Assim, podemos pensar nossa função objetivo como a aplicação sucessiva de um método de integração numérica que tem como entrada os controles e o estado inicial. No Capítulo 6, calculamos, utilizando as fórmulas canônicas introduzidas por Evtushenko [47] e uma estratégia mista dos modos diretos e reverso, o gradiente para uma função objetivo geral que utilize qualquer método de integração na família dos métodos Runge-Kutta. Nos atrevemos a dizer que realizar esta tarefa à mão e em forma eficiente é praticamente impossível.

Em [4] são apresentadas algumas conclusões interessantes a respeito da implementação de ferramentas de diferenciação automática. Dentre elas destacam-se:

1. Foi observado que a eficiência dos pacotes de diferenciação automática é sensível a detalhes de implementação.

2. Durante muitos anos foi assumido (justificadamente) que o custo computacional era dominado pela aritmética e, por essa razão, toda a atenção foi dedicada ao desenvolvimento de *hardware* eficiente para operações em ponto flutuante.

3. Na atualidade, o *overhead* introduzido pelo tratamento de memória dinâmica e a sobrecarga de operadores, deverá se tornar um tópico de estudo que aponte para o desenvolvimento de *hardware* eficiente nestas tarefas.

Muitas novas idéias estão sendo testadas para o aperfeiçoamento das técnicas de diferenciação computacional. O problema é que os testes, em geral, são inconclusivos. As plataformas computacionais e os diferentes modos de implementar as idéias podem modificar uma conclusão. O investimento na área de *hardware* deverá criar um *standard* eficiente para o manejo de memória dinâmica que viabilize e torne efetivas as metodologias que precisem dela.

Ainda há muitíssimos problemas nos quais a diferenciação automática pode ser aplicada. O cálculo eficiente de derivadas abre as portas para o desenvolvimento de novos métodos e novas formulações de problemas já existentes. Pessoalmente, achamos que neste campo, onde a Matemática, em particular a Otimização Contínua, mistura-se com a Computação, há muita coisa para ser feita. A diferenciação automática, possibilitando o cálculo eficiente de derivadas, revolucionará áreas tais como Análise Numérica e Otimização, convertendo-se numa ferramenta amplamente utilizada. Pacotes acadêmicos como LANCELOT [36] e BOX [52] e *softwares* comerciais como Matlab e Mathematica também incorporarão técnicas de diferenciação automática que substituirão, quando possível, as técnicas de minimização sem derivadas e os métodos de diferenciação simbólica e aproximação por diferenças finitas. As linguagens de programação mais difundidas na área de Métodos Numéricos, como Fortran e C, também incorporarão técnicas de diferenciação automática que transformarão subrotinas que avaliem funções em subrotinas que avaliem as funções junto com suas derivadas. Esta transformação será efetuada durante a compilação dos algoritmos e em forma transparente para o usuário ou programador.

Bibliografia

[1] J. Abate, C. H. Bischof, L. Roh and A. Carle, *Algorithms and design for a second-order automatic differentiation module*, em *Proceedings of the International Symposium on Symbolic and Algebraic Computing (ISSAC) '97*, Association of Computing Machinery, New York, 1997, pp. 149–155.

[2] B. M. Averick, R. G. Carter, J. J. Moré and G. L. Xue, *The MINPACK-2 test problem collection*, Preprint MCS-P153-0692, Argonne National Laboratory, Argonne, Illinois, 1992.

[3] B. M. Averick, J. J. Moré, C. H. Bischof, A. Carle and A. Griewank, *Computing large sparse Jacobian matrices using automatic differentiation*, SIAM Journal in Scientific Computing, 15 (1994), pp. 285–294.

[4] M. C. Bartholomew-Biggs, L. Bartholomew-Biggs and B. Christianson, *Optimization & automatic differentiation in ADA: some practical experience*, Optimization Methods and Software, 4 (1994), pp. 47–73.

[5] J. Barzilai and J. M. Borwein, *Two point step size gradient methods*, IMA Journal of Numerical Analysis, 8 (1988), pp. 141–148.

[6] D. P. Bertsekas, *On the Goldstein-Levitin-Polyak gradient projection method*, IEEE Transactions on Automatic Control, 21 (1976), pp. 174–184.

[7] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont, MA, 1995.

[8] M. Berz, C. Bischof, G. Corliss and A. Griewank, *Computational Differentiation: Techniques, Applications, and Tools*, editado por M. Berz, C. Bischof, G. Corliss and A. Griewank, SIAM, Philadelphia, 1996.

[9] R. H. BIELSCHOWSKY, A. FRIEDLANDER, F. M. GOMES, J. M. MARTÍNEZ AND M. RAYDAN, *An adaptive algorithm for bound constrained quadratic minimization*, por aparecer em Investigación Operativa.

[10] E. G. BIRGIN, J. M. MARTÍNEZ AND M. RAYDAN, *Nonmonotone spectral projected gradient methods on convex sets*, Technical Report 92/97, Departamento de Matemática, IMECC-UNICAMP, Campinas, SP, Brazil, 1997.

[11] C. H. BISCHOF, A. CARLE, G. CORLISS, A. GRIEWANK AND P. HOVLAND, *ADIFOR: Generating derivative codes from Fortran programs*, Scientific Programming, 1 (1992), pp. 1–29.

[12] C. H. BISCHOF AND A. GRIEWANK, *ADIFOR: A Fortran system for portable automatic differentiation*, Preprint MCS-P317-0792, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1992.

[13] C. H. BISCHOF, A. CARLE, P. M. KHADEMI AND G. PUSCH, *Automatic differentiation: obtaining fast and reliable derivatives - fast*, em *Control Problems in Industry*, editado por I. Lasiecka and B. Morton, Birkhäuser, Boston, 1995, pp. 1–16.

[14] C. H. BISCHOF, A. CARLE, P. H. KHADEMI, A. MAUER AND P. HOVLAND, *ADIFOR 2.0 user's guide (Revision C)*, Preprint MCS-TM-192, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1995.

[15] C. H. BISCHOF AND A. GRIEWANK, *Computational differentiation and multidisciplinary design*, em *Inverse Problems and Optimal Design in Industry*, editado por H. Engl and J. McLaughlin, Teubner Verlag, Stuttgart, Germany, 1994, pp. 187–211.

[16] C. H. BISCHOF, L. GREEN, K. HAIGLER AND T. KNAUFF, *Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation*, em *Proceedings of the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, 1994, pp. 73–84.

[17] C. H. BISCHOF, A. CARLE AND P. M. KHADEMI, *Fortran 77 interface specification to the SparsLinC library*, Preprint ANL/MCS-TM-196, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1995.

[18] C. H. BISCHOF, P. M. KHADEMI, A. BOUARICHA AND A. CARLE, *Efficient computation of gradients and Jacobians by dynamic explotation of sparsity in automatic differentiation*, Optimization Methods and Software, 7 (1996), pp. 1–39.

[19] C. H. BISCHOF, A. BOUARICHA, P. M. KHADEMI AND J. J. MORÉ, *Computing gradients in large-scale optimization using automatic differentiation*, Preprint MCS-P488-0195, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1996.

[20] C. H. BISCHOF AND M. HAGHIGHAT, *Hierachical approaches to automatic differentiation*, Preprint MCS-P571-0396, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1996.

[21] C. H. BISCHOF, A. CARLE, P. M. KHADEMI AND A. J. MAUER-OATS, *ADIFOR 2.0: Automatic differentiation of Fortran 77 programs*, IEEE Computational Science & Engineering, 3 (1996), pp. 18–32.

[22] C. H. BISCHOF, L. ROH, AND A. J. MAUER-OATS, *ADIC: an extensible automatic differentiation tool for ANSI-C*, Software Practical Experience, 27 (1997), pp. 1425–1456.

[23] J. C. BUTCHER, *On Runge-Kutta processes of high order*, Journal of Australian Mathematic Society, 4 (1964), pp. 179–194.

[24] M. BORN AND E. WOLF, *Principles of Optics*, Pergamon, London, 1980.

[25] R. P. BRENT, *Some efficient algorithms for solving systems of nonlinear equations*, SIAM Journal of Numerical Analysis, 10 (1973), pp. 327–344.

[26] K. M. BROWN, *A quadratically convergent Newton-like method based on Gaussian elimination*, SIAM Journal of Numerical Analysis, 6 (1969), pp. 560–569.

[27] P. H. CALAMAI AND J. J. MORÉ, *Projected gradient methods for linearly constrained problems*, Mathematical Programming, 39 (1987), pp. 93–116.

[28] I. CHAMBOULEYRON, J. M. MARTÍNEZ, A. C. MORETTI AND M. MULATO, *Retrieval of optical constants and thickness of thin films from transmission spectra*, Applied Optics, 36 (1997), pp. 8238–8247.

[29] I. CHAMBOULEYRON, J. M. MARTÍNEZ, A. C. MORETTI AND M. MULATO, *Optical constants of thin films by means of a pointwise constrained optimization approach*, Thin Solid Films, 317 (1998), pp. 133–136.

[30] D. B. CHRISTIANSON, A. J. DAVIES, L. C. W. DIXON, R. ROY AND P. VAN DER ZEE, *Giving reverse differentiation a helping hand*, Optimization Methods and Software, 8 (1997), pp. 53–67.

[31] T. F. COLEMAN, B. S. GARBOW AND J. J. MORÉ, *Software for estimating sparse Jacobian matrices*, ACM Transactions on Mathematical Software, 10 (1984), pp. 329–345.

[32] T. F. COLEMAN, B. S. GARBOW AND J. J. MORÉ, *Fortran subroutines for estimating sparse Jacobian matrices*, ACM Transactions on Mathematical Software, 10 (1984), pp. 346–347.

[33] T. F. COLEMAN AND J. J. MORÉ, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 1876–209.

[34] A. R. CONN, N. I. M. GOULD AND PH. L. TOINT, *Global convergence of a class of trust region algorithms for optimization with simple bounds*, SIAM Journal on Numerical Analysis, 25 (1988), pp. 433–460. Veja também SIAM Journal on Numerical Analysis, 26 (1989), pp. 764–767.

[35] A. R. CONN, N. I. M. GOULD AND PH. L. TOINT, *A globally convergent augmented Lagrangean algorithm for optimization with general constraints and simple bounds*, SIAM Journal on Numerical Analysis, 28 (1991), pp. 545–572.

[36] A. R. CONN, N. I. M. GOULD AND PH. L. TOINT, *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A), Springer Series in Computational Mathematics 17*, Springer-Verlag, New York, Berlin and Heidelberg, 1992.

[37] J. H. CONWAY AND N. J. C. SLOANE, *Sphere Packings, Lattices and Groups*, Springer-Verlag, New York, 1988.

[38] A. R. CURTIS, M. J. D. POWELL AND J. K. REID, *On the estimation of sparse Jacobian matrices*, Journal of the Institute of Mathematics and Applications, 13 (1974), pp. 117–119.

[39] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New York, 1983.

[40] M. A. DINIZ-EHRHARDT, M. A. GOMES-RUGGIERO AND S. A. SANTOS, *Comparing the numerical performance of two trust-region algorithms for large-scale bound-constrained minimization*, por aparecer em Investigación Operativa.

[41] L. C. W. DIXON, Z. A. MAANY AND M. MOHSENINIA, *Automatic differentiation of large sparse systems*, Journal of Economic Dynamics and Control, 19 (1990), pp. 299–311.

[42] L. C. W. DIXON, *Use of automatic differentiation for calculating Hessians and Newton steps*, em *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, editado por A. Griewank and G. F. Corliss, SIAM, Philadelphia, 1991, pp. 114–125.

[43] J. C. DUNN, *Global and asymptotic convergence rate estimates for a class of projected gradient processes*, SIAM Journal on Control and Optimization, 19 (1981), pp. 368–400.

[44] J. C. DUNN, *Gradient related constrained minimization algorithms in function spaces: convergence properties and computational implications*, em *Large Scale Optimization: State of the Art*, editado por W. W. Hager, D. W. Hearn and P.M. Pardalos, Kluwer, 1994.

[45] Y. G. EVTUSHENKO, *Numerical Optimization Techniques*, Optimization Software Inc., Publications Division, New York, 1985.

[46] Y. G. EVTUSHENKO, *Automatic differentiation viewed from optimal control theory*, em *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, editado por A. Griewank and G. F. Corliss, SIAM, Philadelphia, 1991, pp. 25–30.

[47] Y. G. EVTUSHENKO, *Computation of exact gradients in distributed dynamic systems*, Optimization, Methods and Software, 9 (1998), pp. 45–75.

[48] E. FEHLBERG, *New high-order Runge-Kutta formulas with stepsize control for systems of first and second-order differential equations*, ZAMM, 44 (1964), pp. 17–29.

[49] E. FEHLBERG, *New high-order Runge-Kutta formulas with an arbitrary small truncation error*, ZAMM, 46 (1964), pp. 1–16.

[50] E. FEHLBERG, *Klassische Runge-Kutta formeln fünfter und siebenter ordnung mit schrittweiten-kontrolle*, Computing, 4 (1964), pp. 93–106.

[51] R. FLETCHER, *Practical Methods of Optimization*, (2nd edition), John Wiley and Sons, Chichester, New York, Brisbane, Toronto and Singapore, 1987.

[52] A. FRIEDLANDER, J. M. MARTÍNEZ AND S. A. SANTOS, *A new trust region algorithm for bound constrained minimization*, Applied Mathematics and Optimization, 30 (1994), pp. 235–266.

[53] A. FRIEDLANDER AND J. M. MARTÍNEZ, *On the maximization of a concave quadratic function with box constraints*, SIAM Journal on Optimization, 4 (1994), pp. 177–192.

[54] A. GEORGE AND E. NG, *Symbolic factorization for sparse Gaussian elimination with partial pivoting*, SIAM Journal of Science and Statistical Computations, 8 (1987), pp. 877–898.

[55] P. E. GILL, W. MURRAY AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London, 1981.

[56] W. GLUNT, T. L. HAYDEN AND M. RAYDAN, *Molecular conformations from distance matrices*, Journal of Computational Chemistry, 14 (1993), pp. 114–120.

[57] A. A. GOLDSTEIN, *Convex Programming in Hilbert Space*, Bulletin of the American Mathematical Society, 70 (1964), pp. 709–710.

[58] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations (3rd edition)*, The Johns Hopkins University Press, Baltimore and London, 1996.

[59] M. A. GOMES-RUGGIERO, J. M. MARTÍNEZ AND A. C. MORETTI, *Comparing algorithms for solving sparse nonlinear systems of equations*, SIAM Journal of Scientific and Statistical Computations, 13 (1992), pp. 459–483.

[60] M. A. GOMES-RUGGIERO, D. N. KOZAKEVICH AND J. M. MARTÍNEZ, *A numerical study on large-scale nonlinear solvers*, Computers and Mathematics with Applications, 32 (1996), pp. 1–13.

[61] N. I. GRACHEV AND Y. G. EVTUSHENKO, *A library of programs for solving optimal control problems*, USSR Computational Mathematics and Mathematical Physics, 19 (1980), pp. 99–119.

[62] N. I. GRACHEV AND A. N. FILKOV, *Solution of optimal control problems in system DIOS*, Computing Center of Russian Academy of Sciences, Moscow, 1986 (em russo).

[63] A. GRIEWANK AND PH. L. TOINT, *On the uncostrained optimization of partially separable objective functions*, em *Nonlinear Optimization 1981*, editado por M. J. D. Powell, Academic Press, London, 1982, pp. 301–312.

[64] A. GRIEWANK, *On automatic differentiation*, em *Mathematical Programming: Recent Developments and Applications*, editado por M. Iri and K. Tanabe, Kluwer Academic Publishers, 1989, pp. 83–108.

[65] A. GRIEWANK AND G. F. CORLISS, *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, editado por A. Griewank and G. F. Corliss, SIAM, Philadelphia, 1991.

[66] A. GRIEWANK AND S. REESE, *On the calculation of Jacobian matrices by the Markowitz rule for vertex elimination*, em *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, editado por A. Griewank and G. F. Corliss, SIAM, Philadelphia, 1991, pp. 126–135.

[67] A. GRIEWANK, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Preprint MCS-P228-0491, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1991.

[68] A. GRIEWANK, *Some bounds on the complexity of gradients, Jacobians, and Hessians*, em *Complexity in Numerical Optimization*, editado por P. M. Pardalos, World Scientific Publishing Co., 1993.

[69] A. GRIEWANK, *ADOL-C. A package for the automatic differentiation of algorithms written in C/C++*, ACM TOMS, 22 (1996), pp. 131–167.

[70] L. GRIPPO, F. LAMPARIELLO AND S. LUCIDI, *A nonmonotone line search technique for Newton's method*, SIAM Journal on Numerical Analysis, 23 (1986), pp. 707–716.

[71] O. S. HEAVENS, *Optical Properties of Thin Films*, Dover, New York, 1991.

[72] M. HEINKENSCHLOSS, *Mesh independence for nonlinear least squares problems with norm constraints*, SIAM Journal on Optimization, 3 (1993), pp. 81–117.

[73] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[74] K. E. HILLSTROM, *JAKE-F - a portable symbolic differentiator of functions given by algorithms*, Technical Report 82-48, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1982.

[75] M. IRI, *Simultaneous computation of functions, partial derivatives and estimates of rounding errors - Complexity and practicality*, Japan Journal of Applied Mathematics, 1 (1984), pp. 223–252.

[76] M. IRI AND K. KUBOTA, *Methods of fast automatic differentiation and applications*, em *Research memorandum RMI 87-02*, Department of Mathematical Engineering and Instrumental Physics, Faculty of Engineering, University of Tokyo, 1987, pp. 87–102.

[77] M. IRI, *History of automatic differentiation and rounding error estimation*, em *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, editado por A. Griewank and G. F. Corliss, SIAM, Philadelphia, 1991, pp. 3–16.

[78] D. W. JUEDES *A taxonomy of automatic differentiation tools*, em *Automatic Differentiation of Algorithms: Theory, Implementation and Applications*, editado por A. Griewank and F. Corliss, SIAM, Philadelphia, 1991, pp. 315–329.

[79] L. V. KANTOROVICH, *Ob odnoĭ matematicheskoĭ simvolike, udobnoĭ pri provedenii vychisleniĭ na mashinakh*, Doklady Akademii Nauk SSSR, 113 (1957), pp. 738–741.

[80] E. S. LEVITIN AND B. T. POLYAK, *Constrained Minimization Problems*, USSR Computational Mathematics and Mathematical Physics, 6 (1966), pp. 1–50.

[81] S. LUCIDI, L. PALAGI AND M. ROMA, *On some properties of quadratic programs with a convex quadratic constraint*, SIAM Journal on Optimization, 8 (1998), pp. 105–122.

[82] F. LUENGO, M. RAYDAN, W. GLUNT AND T. L. HAYDEN, *Preconditioned spectral gradient method for unconstrained optimization problems*, Technical Report 96-08, Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela, 47002 Caracas 1041-A, Venezuela, 1996.

[83] J. C. MANIFACIER, J. GASIOT, AND J. P. FILLARD, *A simple method for the determination of the optical constants n, k and the thickness of a weakly absorbin film*, J. Phys. E: Sci. Instrum, 9 (1976), pp. 1002.

[84] J. M. MARTÍNEZ, *Generalization of the methods of Brent and Brown for solving nonlinear simultaneous equations*, SIAM Journal of Numerical Analysis, 16 (1979), pp. 434–448.

[85] J. M. MARTÍNEZ, *Solving nonlinear simultaneous equations with a generalization of Brent's method*, BIT, 20 (1980), pp. 501–510.

[86] J. M. MARTÍNEZ AND S. A. SANTOS, *Métodos Computacionais de Otimização*, 20º Colóquio Brasileiro de Matemática, editado por J. M. Martínez e S. A. Santos, IMPA, Rio de Janeiro, Brasil, 1995.

[87] J. M. MARTÍNEZ AND S. A. SANTOS, *A trust region strategy for minimization on arbitrary domains*, Mathematical Programming, 68 (1995), pp. 267–302.

[88] J. M. MARTÍNEZ AND S. A. SANTOS, *Convergence results on an algorithm for norm constrained regularization and related problems*, RAIRO Operations Research, 31 (1997), pp. 269–294.

[89] J. M. MARTÍNEZ, *Augmented Lagrangians and sphere packing problems*, por aparecer em International Journal of Computer Mathematics.

[90] G. P. MCCORMICK AND R. A. TAPIA, *The gradient projection method under mild differentiability conditions*, SIAM Journal on Control, 10 (1972), pp. 93–98.

[91] B. A. MURTAGH AND M. A. SAUNDERS, *MINOS User's Guide*, Report SOL 77-9, Department of Operations Research, Stanford University, California, 1977.

[92] B. A. MURTAGH AND M. A. SAUNDERS, *Large-scale linearly constrained optimization*, Mathematical Programming, 14 (1978), pp. 41–72.

[93] S. G. NASH, *Preconditioning of truncated-Newton methods*, SIAM Journal on Scientific and Statistical Computing, 6 (1985), pp. 599–616.

[94] G. N. NEWSAM AND J. D. RAMSDELL, *Estimation of sparse Jacobian matrices*, SIAM Journal on Algorithms and Discrete Methods, 4 (1983), pp. 404–417.

[95] J. NOCEDAL, *Large scale unconstrained optimization*, em *The State of the Art in Numerical Analysis*, editado por A. Watson and I. Duff, Oxford University Press, Oxford, 1997.

[96] A. PERRY, *A modified conjugate gradient algorithm*, Operations Research, 26 (1978), pp. 1073–1078.

[97] PHAM DINH TAO AND LE THI HOAI AN, *A D.C. Optimization algorithm for solving the trust-region subproblem*, SIAM Journal on Optimization, 8 (1998), pp. 476–505.

[98] E. POLAK, *Computation Methods in Optimization*, Academic Press, New York and London, 1971.

[99] F. RENDL AND H. WOLKOWICZ, *A semidefinite framework to trust region subproblems with application to large scale minimization*, por aparecer em SIAM Journal on Optimization.

[100] M. RAYDAN, *On the Barzilai and Borwein choice of steplength for the gradient method*, IMA Journal of Numerical Analysis, 13 (1993), pp. 321–326.

[101] M. RAYDAN, *The Barzilain and Borwein gradient method for the large scale unconstrained minimization problem*, SIAM Journal on Optimization, 7 (1997), pp. 26–33.

[102] S. A. SANTOS AND D. C. SORENSEN, *A new matrix-free algorithm for the large-scale trust-region subproblem*, Technical Report 95-20, Department of Computational and Applied Mathematics, Rice University, 1995.

[103] A. SCHWARTZ AND E. POLAK, *Family of projected descent methods for optimization problems with simple bounds*, Journal of Optimization Theory and Applications, 92 (1997), pp. 1–31.

[104] V. E. SHAMANSKII, *A modification of Newton's method*, Ukrainskii Mathematicheskii Zhurnal, 19 (1967), pp. 133–138.

[105] E. B. SHANKS, *Solution of differential equations by evaluation of functions*, Mathematical Computation, 20 (1966), pp. 21–38.

[106] D. F. SHANNO AND K. H. PHUA, *Minimization of unconstrained multivariate functions*, ACM Transactions on Mathematical Software, 2 (1976), pp. 87–94.

[107] D. C. SORENSEN, *Minimization of a large scale quadratic function subject to a spherical constraint*, SIAM Journal on Optimization, 7 (1997), pp. 141–161.

[108] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag Inc., Berlin, Heidelberg and New York, 1972.

[109] R. SWANEPOEL, *Determination of the thickness and optical constants of amorphous silicon*, J. Phys. E: Sci. Instrum., 16 (1983), pp. 1214–1222.

[110] R. SWANEPOEL, *Determination of surface roughness and optical constants of inhomogeneous amorphous silicon films*, J. Phys. E: Sci. Instrum., 17 (1984), pp. 896–903.

[111] K. L. TEO AND Z. S. WU, *Computation Methods for Optimizing Distributed Systems*, Academic Press, Orlando, 1984.

[112] K. L. TEO, C. J. GOH AND K. H. WONG, *A Unified Computational Approach to Optimal Control Problems*, Longman Scientific & Technical, England, 1991.

[113] C. R. VOGEL, *A constrained least-squares regularization method for nonlinear ill-posed problems*, SIAM Journal on Control and Optimization, 28 (1990), pp. 34–49.

[114] D. S. WATKINS, *Fundamentals os Matrix Computations*, John Wiley & Sons Inc., New York, Chichester, Brisbane, Toronto and Singapore, 1991.

[115] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

[116] T. J. YPMA, *Historical development of the Newton-Raphson method*, SIAM Review, 37 (1995), pp. 531–551.