Métodos intervalares em otimização global

Tiago de Morais Montanher

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

Programa: Matemática Aplicada
Orientador: Prof. Dr. Walter Figueiredo Mascarenhas

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, setembro de 2009

Agradecimentos

Agradeço a meu orientador, Walter Mascarenhas, que soube dar bons conselhos ao longo dos anos em que trabalhamos juntos e sempre foi paciente, até em meus momentos de maior teimosia. Agradeços aos professores Eduardo Colli, Nelson Khul e Alexandre Roma pois participaram ativamente de minha formação matemática e souberam dar puxões de orelha nos momentos em que eu os mereci. Agradeço a meus colegas do IME que me motivaram e inspiraram durante minha graduação e mestrado. Em especial agradeço a todos os usuários do laboratório do MAP que sempre estiveram dispostos a me ajudar e discutir idéias importantes para este trabalho.

No âmbito pessoal agradeço aos amigos que sempre me incentivaram e tiveram papel fundamental em minha formação. Em especial destaco os irmãos do chá, Phalkon, Adalberto, Alexandre Vicentini, José Mario e especialmente ao Bruno Ribeiro pela amizade de mais de 20 anos.

Agradeço a meus pais, Antonio e Sonia, que sempre me deram liberdade para tomar decisões e a meu irmão Danilo por sua sinceridade e companheirismo. Finalmente agradeço à Jéssica Martins Camargo pelo amor e paciência que me dedicou nos últimos dois anos e meio.

Resumo

Neste trabalho discutimos problemas de otimização sob a ótica global. Desenvolvemos algoritmos capazes de limitar todas as raízes de um sistema de equações não lineares, todos os mínimos globais de uma função irrestrita e todos os mínimos globais de uma função com restrições de igualdade. A aritmética intervalar é a ferramenta que permite a construção de nossos algoritmos e nós a apresentamos comparando-a com ponto flutuante. Mostramos que a aritmética intervalar permite cálculos verificados e dá informações globais sobre o comportamento de uma função em um intervalo. O preço pago por essas vantagens é a eficiência pois ela é cerca de 10 vezes mais lenta que a aritmética de ponto flutuante.

Nosso trabalho resultou em um pacote escrito em Matlab para resolver problemas de dimensão baixa. Nosso enfoque é computacional e todos os algoritmos foram testados em problemas clássicos de otimização global. Os resultados são descritos e comparados com o obtido por métodos de convexificação. O leitor pode repetir nossos experimentos pois disponibilizamos os códigos do pacote e de testes para download.

Palavras-chave: Otimização Global, Aritmética Intervalar, Método de Newton Intervalar.

Abstract

This work deals with optimization problems from a global viewpoint. We develop algorithms to bound all roots of nonlinear systems of equations, all global minima of unconstrained functions and all global minima of functions subject to equality constraints. Interval arithmetic is the tool that made possible our algorithms and we present this arithimetic comparing it with the floating point arithmetic. We show that interval arithmetic allows the verified calculus and give us global informations about the behavior of a function in an interval. On the other hand, we show that floating point arithmetic is about 10 times faster than interval arithmetic.

We wrote a Matlab toolbox to solve low dimensional problems. Our approach is mainly computational and our algorithms were tested in classical global optimization problems. The results are described and compared with results obtained by convexification methods. The reader can repeat our experiments since our toolbox and test set problems are available online.

Keywords: Global Optimization, Interval Arithmetic, Interval Newton Method.

Sumário

Lista de Abreviaturas xv

Lista de Símbolos xvii

Lista de Figuras xix

1 Introdução 1

2 Aritmética Intervalar 5

2.1 Introdução 5

2.2 INTLAB 6

2.3 Aritmética Intervalar 7

2.3.1 Intervalos 7

2.3.2 Operações Aritméticas 9

2.3.3 Potências Intervalares 11

2.3.4 Cálculos Verificados 11

2.3.5 Matrizes e Vetores 13

2.4 Teoremas de Inclusão 14

2.4.1 Propriedades Algébricas 14

2.4.2 Inclusão Monotônica 16

2.5 Implementações da Aritmética Intervalar 18

2.6 Extensões Intervalares 20

2.7 Divisão Estendida 22

2.8 Comparação de Eficiência 24

3 Sistemas de Equações Não Lineares 27

3.1 Introdução 27

3.2 Notação 28

3.3 Newton Intervalar 28

3.3.1 Teorema do Valor Médio 29

3.3.2 O Método de Newton 29

3.3.3 Método de Gauss Seidel 31

3.4 Método de Hansen e Greenberg 32

3.4.1 Visão Geral 32

3.4.2 Passo de Escolha 32

3.4.3 Passo de Eliminação 33

3.4.4 Passo de Redução 33

3.4.5 Passo de Divisão 34

3.4.6 Tomando Uma Caixa Como Solução 34

3.4.7 O Algoritmo 34

3.4.8 Convergência do Algoritmo 36

3.5 A Função intsolve 36

3.6 Teoremas de Moore 38

3.7 Método de Krawczyk 40

3.8 A Função intksolve 42

3.9 Estudos Computacionais 43

3.9.1 Himmelblau 43

3.9.2 Brown Almost Linear 44

3.9.3 Bullard e Biegler 45

3.9.4 Ferraris e Tronconi 46

3.9.5 Equilíbrio de Combustão 47

3.9.6 Kubicek 48
3.9.7 Smith 49
3.9.8 Equações Trigonométricas 50

4 Otimização Irrestrita **53**

4.1 Introdução 53
4.2 Notação 54
4.3 Visão Geral do Método 54
4.4 Descrição do Algoritmo 55
4.4.1 Passo de Escolha 55
4.4.2 Análise do Gradiente 56
4.4.3 Passo da Hessiana 56
4.4.4 Passo de Atualização 56
4.4.5 Passo de Redução 57
4.4.6 Passo de Divisão 57
4.4.7 Tomando Uma Caixa Como Candidata 58
4.4.8 O Algoritmo 58
4.4.9 Convergência do Algoritmo 61
4.5 A Função intmininc 61
4.6 Estudos Computacionais 63
4.6.1 Rosenbrock 64
4.6.2 Rastringin 65
4.6.3 Levy 66
4.6.4 Griewank 68
4.6.5 Goldstein e Price 69
4.6.6 Three Hump Camel 70
4.6.7 Branin 72
4.6.8 Easom 73

4.6.9 Shubert 74
4.6.10 Michalewics 75

5 Otimização Com Restrições de Igualdade **77**

5.1 Introdução 77
5.2 Notação 78
5.3 Condições de Lagrange 78
5.4 Visão Geral do Método 79
5.5 Descrição do Algoritmo 81
5.5.1 Passo de Escolha 81
5.5.2 Passo das Restrições 81
5.5.3 Limitando os Lagrangeanos 81
5.5.4 Passo de Atualização 82
5.5.5 Passo de Redução 85
5.5.6 Passo de Divisão 85
5.5.7 Tomando Uma Caixa Como Candidata 86
5.5.8 O Algoritmo 86
5.6 A função intmincon 89
5.7 Estudos Computacionais 91
5.7.1 Runarsson 91
5.7.2 Runarsson 2 92
5.7.3 Runarsson 3 93

6 Conclusões **95**

6.1 Considerações Finais 95

A Diferenciação Automática **97**

Referências Bibliográficas **99**

Índice Remissivo **102**

Lista de Abreviaturas

NaN	Not a Number.
$RMAX$	Maior número real representável em nosso computador, por volta de 10^{350}.
$IEEE754$	Padrão para aritmética de ponto flutuante.

Lista de Símbolos

M(t)	Conjunto dos números de máquina em nosso computador com tipo t.
M	Conjunto dos números de máquina em nosso computador com tipo *double*.
I(R)	Conjunto dos intervalos reais.
I(M)	Conjunto dos intervalos formados por M
$w(a)$	Comprimento do intervalo a.
$\mathbb{W}(A)$	Comprimento do vetor intervar A.
$\|a\|$	Módulo do intervalo a.
$mid(a)$	Ponto médio do intervalo a.
$rad(a)$	Raio do intervalo a.
$\|\|A\|\|$	Norma do vetor ou matriz intervalar A.
$\downarrow(.)$	Modo de arredondamento para baixo.
$\uparrow(.)$	Modo de arredondamento para cima.
$\bullet(.)$	Modo de arredondamento para o mais próximo.
x^I	Avaliação intervalar de x.

Lista de Figuras

2.1 Comparação de performance entre as aritméticas de ponto flutuante e intervalar. Os gráficos dão a razão entre o tempo de execução com intervalos e o tempo de execução com ponto flutuante em escala logarítmica. As tarefas foram executadas 10000 vezes e os tempos médios de execução usados no cálculo da razão. 25

4.1 Função de Rosenbrock no domínio $-5 \leq x_1, x_2 \leq 10$ 64

4.2 Função de Rastringin no domínio $-5.12 \leq x_1, x_2 \leq 5.12$ 66

4.3 Função de Levy no domínio $-10 \leq x_1, x_2 \leq 10$ 67

4.4 Função de Griewank no domínio $-6 \leq x_1, x_2 \leq 6$ 69

4.5 Função de Goldstein e Price no domínio $-2 \leq x_1, x_2 \leq 2$ 70

4.6 Função Three Hump Camel no domínio $-5 \leq x_1, x_2 \leq 5$ 71

4.7 Função de Branin no domínio $-5 \leq x_1 \leq 10$, $0 \leq x_2 \leq 15$ 72

4.8 Função de Easom no domínio $-30 \leq x_1, \leq x_2 \leq 30$ 73

4.9 Função de Shubert no domínio $-10 \leq x_1, x_2 \leq 10$ 74

4.10 Função de Michalewicz no domínio $0 \leq x_1, x_2 \leq \pi$ 76

Capítulo 1

Introdução

Este trabalho discute métodos numéricos de otimização global. Nosso objetivo é estudar e implementar algoritmos que resolvem os problemas abaixo

Problema 1 *Determinar* ***todas*** *as raízes de um sistema de equações não lineares* $f(x) = 0$.

Problema 2 *Determinar* ***todos*** *os mínimos* ***globais*** *de uma função* f *sem restrições.*

Problema 3 *Determinar* ***todos*** *os mínimos* ***globais*** *de uma função* f *sujeitos a um conjunto de restrições de igualdade* $c(x) = 0$.

Estes problemas são difíceis de resolver usando métodos tradicionais de otimização como os apresentados por Nocedal e Wright (2006) ou Luenberger (1989). Destacamos duas razões que justificam esta dificuldade:

A primeira é a natureza local da teoria desenvolvida pela literatura de otimização desde os tempos de Newton e Lagrange. Para constatar este fato basta notar que algoritmos como o método de Newton e o BFGS partem da idéia de procurar um ponto onde o gradiente da função que desejamos minimizar se anula. Esta condição é local e está presente em muitos algoritmos de otimização não linear.

A segunda é a aritmética de ponto flutuante, usada na maioria dos algoritmos de programação não linear. Esta aritmética introduz erros em nossos cálculos que são dificeis de estimar quando avaliamos expressões complexas. Além disso ao avaliar uma função em um ponto usando aritmética de ponto flutuante não ganhamos qualquer informação sobre o comportamento da função em uma proximidade deste ponto. Neste sentido a aritmética de ponto flutuante é local.

Para resolver problemas de otimização global precisamos lidar com estes pontos de alguma forma. Como a teoria de otimização global não é desenvolvida como a local e não conhecemos nenhum resultado que possa substituir as condições de otimalidade de primeira e segunda ordem

mantendo a garantia de encontrar todos os mínimos globais de uma função, vamos explorar meios de lidar com as dificuldades provenientes da aritmética de ponto flutuante.

A aritmética intervalar criada por Moore (1979) é capaz de lidar com os problemas da aritmética de ponto flutuante com facilidade. Com ela é facil conhecer erros de arredondamento obtidos em cálculos complexos e também podemos avaliar o comportamento global de uma função em um intervalo ou caixa. A principal desvantagem desta aritmética com relação à de ponto flutuante é a eficiência pois, como veremos, a aritmética de ponto flutuante é cerca de 10 vezes mais rápida que a intervalar.

Os algoritmos apresentados em nosso trabalho utilizam as condições de otimalidade local e a aritmética intervalar para resolver os problemas 1 - 3 com garantias. Isto quer dizer que os métodos nunca falham ao buscar os mínimos globais de uma função. Por outro lado, estes métodos são lentos e seu uso em problemas de tamanho médio e grande ainda é proibitivo.

Um dos principais resultados do trabalho é o *INTSOLVER*, um pacote escrito em Matlab que implementa nossas idéias e resolve os problemas 1 - 3 em casos gerais de dimensões baixas. Nosso pacote está disponível em

http://www.mathworks.com/matlabcentral/fileexchange/25211

As principais características do *INTSOLVER* estão relacionadas a seguir

1. Resolve os problemas 1 - 3 limitando as soluções por caixas de tamanho pré-fixado.
2. Permite ao usuário controlar o número máximo de iterações, avaliações de função, gradiente e hessiana em cada método.
3. Permite que o usuário opte por iterar o método até encontrar todas as soluções de um problema. Este modo pode ser demorado e cabe ao usuário usá-lo com coerência.
4. Todos os cálculos de derivada e hessiana são feitos por diferenciação automática.
5. Simples de usar. Usuários de Matlab não terão problemas em entender nosso pacote.

Com relação ao texto, os problemas 1 - 3 são tratados em capítulos diferentes. Cada capítulo é independente e o leitor pode ir direto ao tópico de interesse. A exceção é o capítulo 2 que traz a teoria necessária para os demais e a seção 3.3 que apresenta o método de Newton intervalar. O leitor com conhecimentos básicos sobre aritmética intervalar pode pular estes pontos em uma primeira leitura e voltar neles quando precisar esclarecer conceitos e notações. Dividimos o texto assim:

- No capítulo 2 apresentamos a aritmética intervalar, suas principais propriedades e uma comparação dela com a aritmética de ponto flutuante.

- No capítulo 3 discutimos métodos para resolver o problema de determinar todas as raízes de f. Introduzimos o método de Newton intervalar e apresentamos o método de Krawczyk.
- Nos capítulos 4 e 5 tratamos os problemas de otimização irrestrita e com restrições de igualdade. Utilizamos o método de Newton intervalar para derivar algoritmos que limitam todas as soluções destes problemas.
- No capítulo 6 discutimos os resultados mais importantes deste trabalho e apontamos tópicos para trabalhos futuros.
- No apêndice A introduzimos a diferenciação automática, um método seguro e eficiente para avaliar derivadas de funções sem a necessidade de escrever suas expressões analíticas. Em nosso pacote todas as avaliações de derivada são feitas automaticamente usando o pacote disponível em Rump (1999).

A última seção de cada capítulo traz estudos computacionais. A literatura de métodos intervalares em otimização global é pequena e alguns autores ao apresentar novos métodos não se preocupam em testa-los em problemas relevantes. Apresentamos problemas de otimização global e avaliamos o desempenho de nossos métodos ao resolvê-los. Todos os estudos computacionais feitos em nosso trabalho podem ser repetidos pelo leitor. Os arquivos de teste estão disponíveis em

http://www.mathworks.com/matlabcentral/fileexchange/25211

Capítulo 2

Aritmética Intervalar

2.1 Introdução

Neste capítulo apresentamos a aritmética intervalar. Ela substitui a aritmética de ponto flutuante em nossas aplicações e têm papel central no desenvolvimento de nossos algoritmos. A definição de uma aritmética é um trabalho tedioso e que foge de nossos objetivos práticos. Por essa razão esse esforço deve ser acompanhado de uma boa justificativa. Em nosso caso temos duas propriedades da aritmética que serão úteis em nossos algoritmos e que não estão presentes na aritmética de ponto flutuante. Essas propriedades são

- Permite cálculos verificados de expressões complexas.

Isto quer dizer que, ao contrário do que acontece na aritmética de ponto flutuante, é fácil estimar os erros decorrentes de arredondamento em nossos cálculos. Ilustramos esta propriedade na seção 2.3.

- Dá informações globais sobre o comportamento das funções.

Ao contrário da aritmética de ponto flutuante, que tem caráter puramente local, a aritmética intervalar traz informações precisas sobre a variação de uma função em determinado intervalo. Veremos um exemplo desta propriedade na seção 2.4.

Este capítulo é um resumo dos trabalhos de Moore (1979), Alefeld e Herzberger (1983) e Hansen e Walster (2004). Todos os conceitos discutidos são ilustrados utilizando o Matlab. Nossos objetivos são:

1. Apresentar a aritmética intervalar e suas propriedades fundamentais.
2. Apresentar o *INTLAB*, um pacote aberto para aritmética intervalar.
3. Comparar as aritméticas de ponto flutuante e intervalar.

Na seção 2.2 apresentamos o *INTLAB*, um pacote escrito em Matlab por Rump (1999) para lidar com intervalos. Ilustramos o uso do pacote ao longo do capítulo. Nas seções 2.3 - 2.7 apresentamos a aritmética intervalar. Descrevemos suas principais propriedades e mostramos que, além de algumas definições, só precisamos da capacidade de alterar o modo de arredondamento em nosso computador para implementá-la. Os resultados apresentados aqui são simples e o leitor não deveria ter dificuldades em obtê-los sozinho. Eles são expostos para tornar nosso trabalho mais completo e por permitirem a construção rigorosa de nossos algoritmos.

Na seção 2.8 comparamos a eficiência das aritméticas de ponto flutuante e intervalar. Mostramos que as vantagens da aritmética intervalar tem um custo: Ela é cerca de 10 vezes mais lenta que a aritmética de ponto flutuante.

O leitor com conhecimentos básicos da aritmética intervalar pode pular e voltar nele quando necessário. O leitor interessado apenas no *INTLAB* pode procurar as partes do texto com o simbolo

>>

que trazem os exemplos de uso da biblioteca.

2.2 INTLAB

A teoria intervalar tem forte apelo computacional. Todo material desenvolvido neste capítulo tem como motivação responder perguntas difíceis para a aritmética de ponto flutuante. Portanto, tão importante quanto estabelecer relações teóricas entre intervalos, é dispor de uma implementação simples e que rode em computadores pessoais.

As primeiras implementações da aritmética intervalar foram escritas para ALGOL e FORTRAN. Atualmente há bibliotecas para lidar com intervalos em C/C++, Java, Python, FORTRAN, Maple, Mupad e Matlab. Dentre as implementações disponíveis destacamos a *INTLIB* escrita para FORTRAN77 (KEARFOTT et al., 1994) e FORTRAN90 (KEARFOTT, 1996a), a *Interval* escrita para C/C++ no projeto *Boost* e o *INTLAB* escrito para Matlab por Rump (1999).

Neste trabalho usamos o *INTLAB* para nossos algoritmos. Essa escolha se deve a três fatores. O primeiro é a facilidade de uso e eficiência do *INTLAB*. O segundo é a facilidade de se programar e modificar funções em Matlab e o terceiro é a portabilidade do pacote. O Matlab está disponível nas plataformas Windows, Linux e Mac, o que facilita o uso de nossas funções.

O *INTLAB* é livre para fins não comerciais e foi criado pelo matemático alemão Rump (1999). Ele é escrito em Matlab e depende apenas deste software e do padrão *IEEE*754 para funcionar. O *IEEE*754 é largamente usado em computadores pessoais, laptops e mainframes o que aumenta a portabilidade do pacote. O *INTLAB* está disponível no endereço

http://www.ti3.tu-harburg.de/rump/intlab/

e sua instalação é simples.

Há três formas de definir intervalos no *INTLAB*. Com o comando intval escrevemos intervalos rigorosos para valores que não são representados pela máquina

```
>> x = intval(0.1)
intval x =
[0.0999, 0.1001]
```

com o comando infsup escrevemos intervalos com extremos conhecidos

```
>> x = infsup(1, 2)
intval x =
[1.0000, 2.0000]
```

e com o comando midrad definimos um intervalo através de seu ponto médio e raio

```
>> x = midrad(0, 1)
intval x =
[-1.0000, 1.0000].
```

Ao longo das demais seções apresentamos o *INTLAB*. Com isto ilustramos conceitos importantes da teoria intervalar e permitimos ao leitor se familiarizar com funções que serão úteis no restante do trabalho.

2.3 Aritmética Intervalar

Nesta seção introduzimos as relações fundamentais entre intervalos. Esta introdução é elementar e para um tratamento completo do assunto o leitor deve consultar (ALEFELD; HERZBERGER, 1983) e (MOORE, 1979).

2.3.1 Intervalos

Definição 1 *Chamamos de intervalo real o conjunto*

$$a := [a_1, a_2] := \{x \in \mathbb{R} | a_1 \leq x \leq a_2, -\infty \leq a_1 \leq a_2 \leq \infty\}. \tag{2.1}$$

e denotamos por $\mathbb{I}(\mathbb{R})$ *o conjunto dos intervalos reais.*

Assim como no caso real, desejamos comparar os elementos de $\mathbb{I}(\mathbb{R})$. As definições abaixo servem a este propósito.

Definição 2 *Dois intervalos* $a = [a_1, a_2]$ *e* $b = [b_1, b_2]$ *são iguais se e somente se* $a_1 = b_1$ *e* $a_2 = b_2$. *Neste caso escrevemos* $a = b$.

Definição 3 *Um intervalo* $a = [a_1, a_2]$ *está contido em* $b = [b_1, b_2]$ *se e somente se* $b_1 \leq a_1$ *e* $a_2 \leq b_2$. *Neste caso escrevemos* $A \subseteq B$.

Dados dois intervalos a e b, o *INTLAB* é capaz de verificar as relações $=$ e $\subseteq$.

```
>> a = infsup(0,1)
intval a =
[0.0000,1.0000]
>> b = infsup(-1,2)
intval b =
[-1.0000,2.0000]
>> a == b
ans =
     0
>> in(a,b)
ans =
     1
```

Definição 4 *A intersecção entre dois intervalos* a *e* b *é vazia* $a \cap b = \varnothing$ *se* $a_1 > b_2$ *ou* $b_1 > a_2$. *Caso contrário a intersecção entre* a *e* b *é o intervalo*

$$a \cap b = [max(a_1, b_1), min(a_2, b_2)]. \tag{2.2}$$

A função *intersect* do *INTLAB* devolve a intersecção de dois intervalos não disjuntos. No caso de intervalos disjuntos esta função devolve o intervalo $[NaN, NaN]$. Onde NaN é a abreviação de Not a Number.

```
>> a = infsup(0,2)
intval a =
[0.0000,2.0000]
>> b = infsup(1,5)
intval b =
[1.0000,5.0000]
>> intersect(a,b)
intval ans =
[1.0000,2.0000]
```

Definição 5 *Dado um intervalo* $a = [a_1, a_2]$ *seu comprimento, módulo, ponto médio e raio são dados respectivamente por*

$$w(a) = a_2 - a_1 \tag{2.3}$$

$$|a| = max(|a_1|, |a_2|) \tag{2.4}$$

$$mid(a) = \frac{a_1 + a_2}{2} \tag{2.5}$$

$$rad(a) = \frac{a_2 - a_1}{2}. \tag{2.6}$$

Estas funções estão definidas no *INTLAB* como segue,

```
>> a = infsup(1,5)
intval a =
[1.0000,5.0000]
>> diam(a)
ans =
     4
>> abss(a)
ans =
     5
>> mid(a)
ans =
     3
>> rad(a)
ans =
     2
```

2.3.2 Operações Aritméticas

Definimos agora as operações aritméticas intervalares seguindo o trabalho de Moore (1979). Na seção 2.5 discutimos pontos relevantes sobre a implementação destas operações em computadores.

Definição 6 *Seja* $\otimes \in \{+, -, *, /\}$ *a operação binária tradicional no conjunto dos números reais. Se* a *e* b *pertencem a* $\mathbb{I}(\mathbb{R})$ *então*

$$a \otimes b := \{c = a^* \otimes b^* | a^* \in a, b^* \in b\} \tag{2.7}$$

Segue daí que dados a *e* b *em* $\mathbb{I}(\mathbb{R})$ *temos*

$$a + b = [a_1 + b_1, a_2 + b_2]. \tag{2.8}$$

$$a - b = [a_1 - b_2, a_2 - b_1]. \tag{2.9}$$

$$a * b = [min\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}, max\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}]. \tag{2.10}$$

Se além disso $0 \notin b$ *temos*

$$\frac{a}{b} = a * \left[\frac{1}{b_2}, \frac{1}{b_1}\right]. \tag{2.11}$$

Estas operações estão definidas no *INTLAB*. A chamada das funções que implementam estas operações aritméticas é idêntica aos casos com tipos double, float e complex.

```
>> a = infsup(0,2)
intval a =
[0.0000,2.0000]
>> b = infsup(1,5)
intval b =
[1.0000,5.0000]
>> a + b
intval ans =
[1.0000,7.0000]
>> a - b
intval ans =
[-5.0000,1.0000]
>> a * b
intval ans =
[0.0000,10.0000]
>> a / b
intval ans =
[0.0000,2.0000]
```

No caso de uma das variáveis ser do tipo intervalar e a outra do tipo double, float ou complex o cálculo é automaticamente convertido para o tipo intervalar.

2.3.3 Potências Intervalares

Além das operações fundamentais descritas acima definimos potências intervalares como segue

Definição 7 *Dado n natural e $x = [x_1, x_2]$ temos*

$$x^n = \begin{cases} [x_1^n, x_2^n] & \text{se } x_1 > 0 \text{ ou } n \text{ é impar} \\ [x_2^n, x_1^n] & \text{se } x_1 < 0 \text{ e } n \text{ é par} \\ [0, |x|^n] & \text{se } 0 \in x \text{ e } n \text{ é par} \end{cases} \tag{2.12}$$

Veremos nas próximas seções que na aritmética intervalar não vale a propriedade $x^n = x \dots x$.

2.3.4 Cálculos Verificados

Nesta seção argumentamos que a aritmética intervalar permite estimar os erros de arredondamento com facilidade. Esta propriedade, que chamamos de cálculo verificado, justifica o uso da aritmética intervalar. O exemplo abaixo ilustra o cálculo verificado e na seção 2.5 discutimos detalhes de implementação que garantem a propriedade.

Exemplo 1 *Vamos usar o método de Newton para encontrar a raiz real de $f(x) = x^3 - 2x - 5$ a partir do ponto $x_0 = \pi$ e desejamos estimar estimar os erros cometidos a cada iteração do método. Para que a propriedade de cálculo verificado fique clara, vamos discutir o problema usando as aritméticas de ponto flutuante e intervalar. A comparação torna óbvia a vantagem da aritmética intervalar no cálculo de erros.*

Inicialmente note que a quantidade π não é representável em nosso computador e teremos que aproxima-la e isso leva a erros de arredondamento. Usando aritmética de ponto flutuante temos

```
>> pi
ans =
    3.1416
```

e o erro cometido na aproximação é fácil de calcular mas não vêm dado explicitamente. Por outro lado, usando aritmética intervalar obtemos

```
>> intval(pi)
intval ans =
[3.14159265358979,3.14159265358980]
```

e o resultado não é uma aproximação de π mas sim um intervalo que seguramente contém o valor. Sendo assim o erro de aproximação cometido é dado explicitamente se fizermos

```
>> diam(intval(pi))
ans =
     1.021405182655144e-14.
```

Agora que x_0 está representado podemos aplicar o método de Newton. Escrevemos

```
>> fnewton = @(x) x - (x^3 - 2*x - 5)/(3*x^2 - 2)
fnewton =
    @(x)x-(x^3-2*x-5)/(3*x^2-2)
```

e utilizando aritmética de ponto flutuante temos

```
>> fnewton(pi)
ans =
    2.4272.
```

Neste ponto estimar os erros de arredondamento devidos ao passo de Newton é uma tarefa complicada. Para isso devemos considerar o erro acumulado da aproximação anterior e recorrer a fórmulas que estimam os erros de arredondamento em operações fundamentais de ponto flutuante. Este trabalho é árduo e se repetirá a cada nova iteração do método. Usando aritmética intervalar teremos

```
>> fnewton(intval(pi))
intval ans =
[   2.42721600770102,   2.42721600770103]
```

e o resultado é novamente um intervalo que contém o verdadeiro valor do passo de Newton. Para estimar o erro de arredondamento que cometemos ao tomar um ponto representável qualquer no intervalo dado acima fazemos

```
>> diam(fnewton(intval(pi)))
ans =
     8.881784197001252e-16.
```

Com este exemplo procuramos convencer o leitor da facilidade de obter estimativas do erro de arredondamento através da aritmética intervalar. Ressaltamos que repetidamente garantimos obter estimativas seguras deste erro. Os detalhes que garantem essa segurança serão discutidos na seção 2.5.

2.3.5 Matrizes e Vetores

Ao longo de nosso trabalho não faremos distinção entre vetores intervalares e caixas. Matrizes e vetores intervalares são definidos como no caso real. As operações entre estes elementos também não sofre alterações. Uma vez definidos a soma e o produto entre intervalos basta substituir as operações tradicionais pelas intervalares e obter a nova operação.

O *INTLAB* permite criar e operar vetores e matrizes intervalares com a mesma facilidade dos tipos double, float e complex.

```
>> A = [infsup(1,2), infsup(-1,0); infsup(0,1), infsup(2,3)];
>> B = [infsup(0,1); infsup(2,3)];
>> A*B
intval ans =
[-3.5000,2.5000]
[3.0000,10.0000]

>> A = [infsup(1,2), infsup(-1,0); infsup(0,1), infsup(2,3)]
intval A =
[1.0000,2.0000] [-1.0000,0.0000]
[0.0000,1.0000] [2.0000,3.0000]
>> B = intval(rand(2))
intval B =
[0.8936,0.8937] [0.3528,0.3529]
[0.0578,0.0579] [0.8131,0.8132]
>> A * B
intval ans =
[0.8357,1.7873] [-0.4603,0.7058]
[0.1157,1.0674] [1.6263,2.7924]
```

Definição 8 *Sejam V e M um vetor e uma matriz intervalar respectivamente. Temos então*

$$\mathbb{W}(V) = max(w(V_1), \ldots, w(V_n)) \tag{2.13}$$

$$||V|| = max(|V_1|, \ldots, |V_n|) \tag{2.14}$$

$$||M|| = max_i \sum_j |M_{ij}|. \tag{2.15}$$

As funções acima não estão definidas no *INTLAB* porém podemos obtê-las da seguinte forma,

```
>> V = [infsup(0,1), infsup(2,3)];

>> max(diam(V))
ans =
     1

>> max(abss(V))
ans =
     3

>> M = [infsup(1,2), infsup(-1,0); infsup(0,1), infsup(2,3)];
>> max(sum(abss(M),1))
ans =
     4
```

2.4 Teoremas de Inclusão

Nesta seção mostramos que algumas propriedades comuns à aritmética real não são válidas no caso intervalar e apresentamos o principal resultado desta aritmética, o teorema fundamental da aritmética intervalar. O material desta seção é baseado nos trabalhos de Hansen e Walster (2004), Moore (1979) e Ratz (1996).

2.4.1 Propriedades Algébricas

As propriedades abaixo seguem imediatamente das definições de soma e produto de intervalos.

Proposição 1 *Sejam a, b e c intervalos em $\mathbb{I}(\mathbb{R})$ então*

$$a + (b + c) = (a + b) + c. \tag{2.16}$$

$$a + b = b + a. \tag{2.17}$$

$$a(bc) = (ab)c. \tag{2.18}$$

$$ab = ba. \tag{2.19}$$

$$a + b = b + c \Rightarrow a = c. \tag{2.20}$$

Sejam $\mathbf{0} = [0,0]$ *e* $\mathbf{1} = [1,1]$ *então* $\mathbf{0}$ *e* $\mathbf{1}$ *são os únicos intervalos tais que,*

$$\mathbf{0} + a = a. \tag{2.21}$$

$$\mathbf{0}a = 0. \tag{2.22}$$

$$\mathbf{1}a = a. \tag{2.23}$$

Os próximos exemplos mostram que algumas propriedades da aritmética real não são válidas no caso intervalar.

Exemplo 2 *Considere o intervalo*

```
>> a
intval a =
[2.0000,3.0000]
```

Segue das definições 2.9 e 2.11 que

```
>> a - a
intval ans =
[-1.0000,1.0000]
```

e

```
>> a / a
intval ans =
[0.6666,1.5000].
```

Em geral, temos que $a - a = \mathbf{0}$ *e* $\frac{a}{a} = \mathbf{1}$ *(se* $0 \notin a$*) se e somente se* $a = [a_1, a_1]$.

Exemplo 3 *Considere os intervalos*

```
>> a
intval a =
[1.0000,2.0000]

>> b = infsup(1,1)
intval b =
    1.0000
```

Note que na aritmética intervalar não vale a propriedade distributiva pois

```
>> a * (b - b)
intval ans =
    0.0000
```

ao passo que

```
>> a * b - a * b
intval ans =
[-1.0000,1.0000]
```

No entanto vale a propriedade subdistributiva que decorre imediatamente da aplicação das definições de soma e produto.

Proposição 2 *Sejam a, b e c intervalos em $\mathbb{I}(\mathbb{R})$ então*

$$a(b+c) \subseteq ab + ac. \tag{2.24}$$

De fato, no exemplo acima temos

```
>>in(a*(b + b), a*b + a*b)
ans =
     1
```

Lembrando que *in* é a função definida pelo *INTLAB* para testar se um intervalo está ou não contido em outro.

2.4.2 Inclusão Monotônica

Definimos agora a propriedade de inclusão monotônica. Ela permite conhecer o comportamento de uma função em todo um intervalo. Esta propriedade dá informações globais sobre uma função e é de grande importância em nossa escolha pela aritmética intervalar. Considere o seguinte exemplo

Exemplo 4 *Desejamos estudar o comportamento da função $f(x) = x^2$ no intervalo $[2,4]$. Se utilizamos aritmética de ponto flutuante obtemos apenas informações locais sobre f. Ao tomar $x = 3$ temos $f'(3) = 6$ e concluímos que este não é um ponto crítico da função. Não podemos dizer nada sobre outros pontos do intervalo a partir desta avaliação de função.*

Considerando aritmética intervalar obtemos informações sobre o comportamento de f em todo o intervalo. Com uma avaliação de função temos $f'([2,4]) = [4,8]$ e sabemos que não há pontos críticos da função nesse intervalo.

O restante da seção dedica-se a generalizar este exemplo.

Definição 9 *Seja $F : \mathbb{I}(\mathbb{R})^n \to \mathbb{I}(\mathbb{R})^m$ uma função intervalar. Dizemos que F tem a propriedade de inclusão monotônica se*

$$y \subseteq x \Rightarrow F(y) \subseteq F(x) \tag{2.25}$$

Exemplo 5 *A função intervalar $f(X) = mid(X) + \frac{1}{2}X$ **não** tem propriedade de inclusão como vemos abaixo com o INTLAB.*

```
>> f = @(x) mid(x) + 0.5 * x
f =
    @(x)mid(x)+0.5*x

>> y = infsup(1,3)
intval y =
[1.0000, 3.0000]

>> z = infsup(2,3)
intval z =
[2.0000, 3.0000]

>> f(y)
intval ans =
[2.5000, 3.5000]

>> f(z)
intval ans =
[3.5000, 4.0000].
```

Será comum nos referirmos à propriedade de inclusão monotônica pelo nome de propriedade de inclusão. O próximo teorema nos diz que funções racionais intervalares tem propriedade de inclusão. Lembramos que uma função racional é aquela que pode ser expressa como razão entre dois polinômios.

Teorema 1 *A função racional intervalar* $F = \frac{P}{Q}$ *onde* Q *não se anula tem a propriedade de inclusão.*

Demonstração: Sejam x, y, u e v intervalos tais que $x \subseteq y$ e $u \subseteq v$. Segue das definições 2.8 – 2.11 que

$$x + u \subseteq y + v$$
$$x - u \subseteq y - v$$
$$x * u \subseteq y * v$$
$$x/u \subseteq y/v$$

O resultado segue da transitividade da relação $\subseteq$ e do fato das funções racionais intervalares serem formadas apenas por essas operações.□

Este resultado é útil pois muitos problemas importantes na prática podem ser escritos como funções racionais. O resultado é mais geral e se estende as funções potência, raiz, trigonométricas, exponencial e logaritmo. No entanto a demonstração nestes casos é mais complicada e foge ao escopo de nosso texto. O leitor interessado pode consultar o trabalho de Moore (1979).

Mostramos agora que a composição de funções intervalares com propriedade de inclusão ainda tem esta propriedade. Este resultado permite a construção de funções intervalares complexas com propriedade de inclusão.

Teorema 2 *Se* f *e* g *são funções com a propriedade de inclusão então a composição* $f(g(x))$ *tem a propriedade de inclusão.*

Demonstração: Sejam X_1 e X_2 intervalos tais que $X_1 \subseteq X_2$. Como g tem a propriedade de inclusão sabemos que $g(X_1) \subseteq g(X_2)$. Uma vez que f também tem a propriedade de inclusão é imediato que $f(g(X_1)) \subseteq f(g(X_2))$.□

2.5 Implementações da Aritmética Intervalar

Nas seções anteriores vários conceitos da aritmética intervalar foram definidos sobre o espaço $\mathbb{I}(\mathbb{R})$. Na prática, desejamos usar esta aritmética em computadores onde o conjunto $\mathbb{R}$ não está definido. Nesta seção argumentamos que a capacidade de alterar o modo de arredondamento de nosso computador garante que podemos representar intervalos e operações de $\mathbb{I}(\mathbb{R})$ em $\mathbb{I}(\mathbb{M})$. Para maiores detalhes sobre os assuntos tratados aqui consulte (754, 1985), (HARGREAVES, 2002), (RUMP, 2001) e (RUMP, 1999).

Denotamos por $\mathbb{M}(t)$ o conjunto finito de números representáveis em nosso computador com o tipo t. Em geral t é do tipo double ou float. Para facilitar nossa análise consideramos apenas o caso $\mathbb{M} = \mathbb{M}(double)$. Todas os resultados obtidos valem para outros tipos de dados.

Assumimos que nosso computador segue o padrão $IEEE754$ (1985). Esta hipótese não deve ser restritiva pois um número grande de computadores o seguem. O leitor interessado em implementações de bibliotecas intervalares sem a necessidade do padrão $IEEE754$ pode consultar o trabalho de Rump (2001).

Dado um ponto $x \notin \mathbb{M}$, desejamos representá-lo neste conjunto. Isto é feito através de arredondamento. O $IEEE754$ permite mudar o modo de arredondamento, para baixo, para cima e para o mais próximo sendo este último o valor padrão. O próximo exemplo mostra que se $x = [x_1, x_2]$ e $x_1 \notin \mathbb{M}$ então o intervalo $x^* = [x_1^*, x_2] \in \mathbb{I}(\mathbb{M})$ pode não conter x se o modo de arredondamento é para o mais próximo.

Exemplo 6 *Suponha que desejamos representar* $x = [0.1, 1]$ *no conjunto* $\mathbb{I}(\mathbb{M})$. *Suponha ainda que* 0.1001 *é o ponto mais próximo de* 0.1 *em* $\mathbb{M}$. *Arredondando para o mais próximo teremos*

$$x^* = [0.1001, 1].$$

e $x \not\subseteq x^*$. *No entanto se alterarmos o modo de arredondamento para baixo teremos*

$$x^* = [0.999, 1].$$

e garantimos que $x_1^* < x_1$ *e* $x \subseteq x^*$.

A capacidade de alterar o modo de arredondamento também garante o cálculo verificado das operações aritméticas como mostramos abaixo. Para simplificar a notação, utilizamos $\downarrow(.)$, $\uparrow(.)$ e $\bullet(.)$ para indicar que a operação $(.)$ foi realizada com o modo de arredondamento para baixo, para cima e para o mais próximo respectivamente.

Definição 10 *Se* a *e* b *são elementos de* $\mathbb{I}(\mathbb{M})$ *então*

$$a + b = [\downarrow (a_1 + b_1), \uparrow (a_2 + b_2)]. \tag{2.26}$$

$$a - b = [\downarrow (a_1 - b_2), \uparrow (a_2 - b_1)]. \tag{2.27}$$

$$a * b = [min\{\downarrow (a_1 b_1), \downarrow (a_1 b_2), \downarrow (a_2 b_1), \downarrow (a_2 b_2)\}, \tag{2.28}$$

$$max\{\uparrow (a_1 b_1), \uparrow (a_1 b_2), \uparrow (a_2 b_1), \uparrow (a_2 b_2)\}]. \tag{2.29}$$

Se além disso $0 \notin b$ *temos*

$$\frac{a}{b} = \left[min\left\{\downarrow \left(\frac{a_1}{b_2}\right), \downarrow \left(\frac{a_1}{b_1}\right), \downarrow \left(\frac{a_2}{b_2}\right), \downarrow \left(\frac{a_2}{b_1}\right)\right\}, \right. \tag{2.30}$$

$$\left. max\left\{\uparrow \left(\frac{a_1}{b_2}\right), \uparrow \left(\frac{a_1}{b_1}\right), \uparrow \left(\frac{a_2}{b_2}\right), \uparrow \left(\frac{a_2}{b_1}\right)\right\}\right]. \tag{2.31}$$

As definições acima garantem o cálculo rigoroso das operações aritméticas intervalares e consequentemente de funções racionais. O cálculo das funções potência, raiz, trigonométricas, exponencial e logaritmo também é verificado. O leitor pode consultar o trabalho de Rump (2001) para mais detalhes.

2.6 Extensões Intervalares

Até aqui discutimos funções intervalares e suas propriedades. Nesta seção mostramos como relacionar funções reais e funções intervalares. Esta associação é importante pois na prática temos funções reais e gostariamos de encontrar seus mínimos ou raízes a partir da aritmética intervalar. O material desta seção é baseado nos trabalhos de Hargreaves (2002), Moore (1979) e Rump (2001).

Sejam M_1 e M_2 conjuntos arbitrários e $g : M_1 \to M_2$ uma função arbitrária. Denote por $S(M_1)$ e $S(M_2)$ a família de todos os subconjuntos de M_1 e M_2 respectivamente. Considere a função $\overline{g} : S(M_1) \to S(M_2)$ dada por,

$$\overline{g}(X) = \{g(x); x \in X, X \in S(M_1)\}.$$

Isto é, dada a função g e um intervalo $X \in \mathbb{I}(\mathbb{R})$, $\overline{g}(X)$ é o conjunto de todos os valores que g assume ao percorrer X. Podemos então definir extensões intervalares.

Definição 11 *Seja* $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$. *Chamamos de extensão intervalar de* f *uma função* $F : \mathbb{I}(\mathbb{M})^n \to \mathbb{I}(\mathbb{M})$ *que tem propriedade de inclusão e tal que para qualquer* $X^* \subseteq X$ *e satisfaz*

$$\overline{f}(X^*) \subseteq F(X).$$

Os algoritmos desenvolvidos nos próximos capítulos admitem que conhecemos extensões inter-

valares das funções que desejamos minimizar ou encontrar raízes. Esta hipótese não é restritiva pois funções racionais, trigonométricas, exponencial, logaritmo, raiz e potência tem extensão intervalar. Para obtê-las basta substituir as variáveis e operações aritméticas reais por suas respectivas variáveis e operações intervalares. Isto permite escrever um grande número de funções importantes em matemática aplicada.

Em geral funções reais admitem mais de uma extensão intervalar. No próximo exemplo consideramos duas extensões de uma mesma função e discutimos alguns aspectos importantes em sua escolha.

Exemplo 7 *Seja* $f(x) = \frac{1}{x^2+1}$ *e considere o intervalo* $X = [-2, 2]$. *Esta função é continua no compacto* X *e portanto admite um valor máximo e mínimo, que desejamos limitar. Considere a extensão de* f

$$F(X) = \frac{1}{X * X + 1}$$

Temos então

```
>> F = @(x) 1 / (x*x + 1)
F =
    @(x)1/(x*x+1)

>> F(infsup(-2,2))
intval ans =
[- Inf,Inf]
```

Isto é, não ganhamos qualquer informação sobre os limitantes de f *em* X. *Considere agora outra extensão de* f

$$G(X) = \frac{1}{X^2 + 1}$$

Temos então

```
>> G = @(x) 1 / (x^2 + 1)
G =
    @(x)1/(x^2+1)

>> G(infsup(-2,2))
intval ans =
[0.1999,1.0000]
```

Segue daí que 0.1999 *e* 1 *são limitantes para o mínimo e o máximo de* f *em* X. *Note que com o mesmo esforço computacional ganhamos mais informações utilizando a extensão* G *ao invés de* F.

Este exemplo mostra que algumas extensões intervalares podem trazer mais informações do que outras. Em geral é difícil saber se a extensão que consideramos é a melhor possível. Porém em alguns casos conhecemos regras que permitem descartar extensões pouco informativas.

Proposição 3 *Se* $f(x) = a_n x^n + \ldots + a_0$ *então a extensão dada pelo método de Horner*

$$F(X) = A_0 + X(A_1 + X(A_2 + \ldots + X(A_n))) \tag{2.32}$$

nunca é pior do que a obtida pela soma de potências $F^* = A_n x * x \ldots * x + \ldots + A_0$.

Este resultado segue diretamente da propriedade subdistributiva e não será demonstrado. O exemplo sugere ainda que o uso de potências não nos dá intervalos maiores do que quando consideramos o produto $x \ldots x$. Isto quer dizer que potências são sempre mais informativas do que este produto.

2.7 Divisão Estendida

Nos próximos capítulos encontramos situações onde nossa definição de divisão não se aplica. Por exemplo ao calcular

$$N = r - \frac{a}{b} \tag{2.33}$$

onde r, a e b são intervalos e $0 \in b$.

Nesta seção definimos uma divisão que estende a de 2.3. Diversos autores discutiram o assunto, dentre eles Ratschek e Rokne (1988), Hansen e Walster (2004), Hammer et al. (1993) e Ratz (1996). Em nosso trabalho adotamos a definição de Ratz. Em nossa definição usamos os operadores especiais $-\infty$, ∞ e NaN definidos no $IEEE754$. Embora o padrão simplifique as idéias ele não é essencial para a construção da extensão de divisão. Qualquer computador que permita alterar o modo de arredondamento e defina operadores equivalentes aos usados aqui pode estender a divisão intervalar.

Definição 12 *Se* a *e* b *pertencem a* $\mathbb{I}(\mathbb{M})$ *então*

$$\frac{a}{b} = \begin{cases} [\downarrow(\frac{a_2}{b_1}), \infty] & se\ a_2 < 0\ e\ b_1 < b_2 = 0 \\ [-\infty, \uparrow(\frac{a_2}{b_2})] \cup [\downarrow(\frac{a_2}{b_1}), \infty] & se\ a_2 < 0\ e\ b_1 < 0 < b_2 \\ [-\infty, \uparrow(\frac{a_2}{b_2})] & se\ a_2 < 0\ e\ 0 = b_1 < b_2 \\ [-\infty, \infty] & se\ a_1 < 0 < a_2\ e\ b_1 < 0 < b_2 \\ [-\infty, \uparrow(\frac{a_1}{b_1})] & se\ a_1 > 0\ e\ b_1 < b_2 = 0 \\ [-\infty, \uparrow(\frac{a_1}{b_1})] \cup [\downarrow(\frac{a_1}{b_2}), \infty] & se\ a_1 > 0\ e\ b_1 < 0 < b_2 \\ [\downarrow(\frac{a_1}{b_2}), \infty] & se\ a_1 > 0\ e\ 0 = b_1 < b_2 \\ \text{NaN} & se\ 0 \notin A\ e\ B = [0, 0] \end{cases} \tag{2.34}$$

A divisão estendida não está definida no *INTLAB*. A função *intdiv* do *INTSOLVER* estende a divisão segundo a definição acima. Ilustramos seu uso abaixo.

```
>> a
intval a =
[1.0000,2.0000]
>> b
intval b =
[-1.0000,5.0000]
>> c = intdiv(a,b)
intval c =
[-Inf,-1.0000] [0.2000,Inf]
>> a
intval a =
[-1.0000,2.0000]
>> b
intval b =
[-3.0000,1.0000]
>> c = intdiv(a,b)
intval c =
[-Inf,Inf]
>> a
intval a =
[1.0000,2.0000]
>> b
intval b =
[0.0000,1.0000]
>> c = intdiv(a,b)
intval c =
[1.0000,Inf]
```

2.8 Comparação de Eficiência

Nesta seção comparamos a eficiência das aritméticas de ponto flutuante e intervalar. Os testes foram feitos em Matlab com o *INTLAB*. Os resultados devem variar com a linguagem de programação e implementação intervalar usadas. Em nossos experimentos utilizamos um AMD Sempron© de $800MHZ$ e $1GB$ de memória RAM com sistema operacional Linux.

Operações de álgebra linear são responsáveis por boa parte do tempo de execução em algoritmos de otimização. Segue daí que quanto mais eficiente o conjunto de rotinas de álgebra linear (BLAS) melhor será o desempenho de nossos métodos.

O Matlab utiliza o BLAS do ATLAS por padrão. Podemos configurá-lo para usar as bibliotecas MKL (Math Kernel Library) da Intel ou o ACML(AMD Core Math Library) da AMD. Estas bibliotecas podem aumentar a eficiência das operações de álgebra linear em até 30%. O *INTLAB* utiliza o BLAS do Matlab em suas operações.

Comparamos o desempenho das aritméticas de ponto flutuante e intervalar ao resolver problemas de álgebra linear. Esperamos que a eficiência no uso de aritmética intervalar seja menor que a aritmética de ponto flutuante.

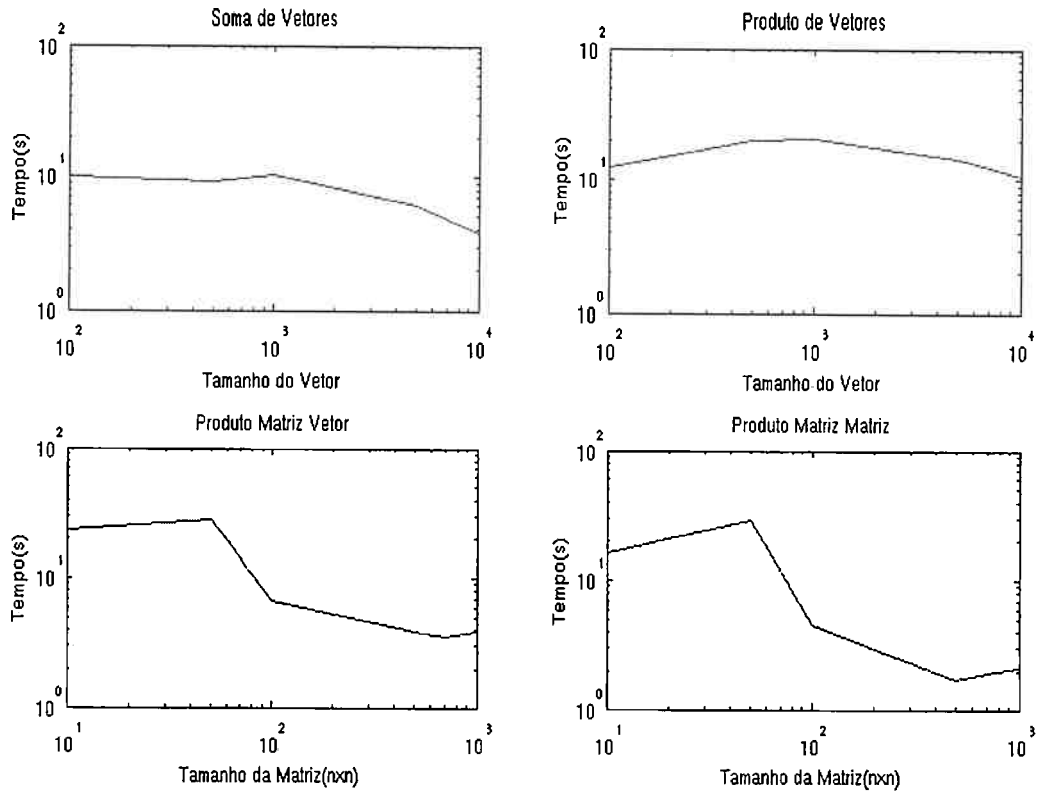

Figura 2.1: Comparação de performance entre as aritméticas de ponto flutuante e intervalar. Os gráficos dão a razão entre o tempo de execução com intervalos e o tempo de execução com ponto flutuante em escala logarítmica. As tarefas foram executadas 10000 vezes e os tempos médios de execução usados no cálculo da razão.

Os gráficos mostram que realizar operações de álgebra linear com aritmética intervalar é cerca de 10 vezes menos eficiente do que com ponto flutuante. Esta perda de eficiência está ligada à implementação do *INTLAB*. As funções do pacote que definem as operações usadas acima necessitam de uma série de verificações sobre a natureza dos dados de entrada e definem a cada chamada um objeto do tipo *intval*. Definir este objeto é uma operação custosa e depende do tamanho dos vetores de entrada.

Note que a diferença de performance diminui quando aumentamos a dimensão do problema, o que leva a crer que é possível criar algoritmos intervalares competitivos em problemas grandes. Este não é o objetivo de nosso trabalho pois pretendemos resolver problemas de dimensão baixa.

Capítulo 3

Sistemas de Equações Não Lineares

3.1 Introdução

Seja $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ uma função diferenciável e X_0 uma caixa. Neste capítulo estudamos o problema de encontrar **todas** as raízes do sistema de equações não lineares $f(x) = 0$ em X_0.

Floudas (2000) e Maranas e Floudas (1995) usam métodos de convexificação e aritmética de ponto flutuante para resolver este problema. Como veremos em 3.9, o algoritmo apresentado por Floudas pode não encontrar todas as raízes de f até mesmo em problemas simples.

A aritmética intervalar permite uma abordagem rigorosa e natural do assunto. Os primeiros trabalhos que utilizam esta ferramenta na análise de sistemas de equações não lineares são de Moore (1977) e Krawczyk (1969). Hansen e Greenberg (1983) apresentaram um método que avalia o comportamento da função em uma caixa e a elimina se pudermos garantir que não existem raízes em seu interior ou bordas. Consideramos as idéias destes autores ao longo deste capítulo. Nossos objetivos são:

1. Estudar o método de Hansen e Greenberg. Ele nos permite limitar todas as raízes de f em X_0.
2. Ilustrar o uso da função *intsolve* do *INTSOLVER*. Esta função implementa o algoritmo de Hansen e Greenberg.
3. Estudar o algoritmo de Krawczyk. Ele nos dá condições suficientes para existência e unicidade de soluções de f em X_0.
4. Ilustrar o uso da função *intksolve* do *INTSOLVER*. Esta função implementa o algoritmo de Krawczyk.
5. Avaliar a função *intsolve* em problemas da literatura de otimização global.

O texto se divide da seguinte forma. Na seção 3.2 apresentamos a notação usada ao longo do capítulo. Em 3.3 apresentamos o método de Newton intervalar. A seção 3.4 dedica-se ao método de

Hansen e Greenberg (1983). Na seção 3.5 ilustramos o uso da função *intsolve* do *INTSOLVER*. Na seção 3.6 apresentamos os teoremas de Moore (1977). Em 3.7 discutimos o algoritmo de Krawczyk (1969) (WOLFE, 1980). A seção 3.8 ilustra o uso da função *intksolve* do *INTSOLVER*. Esta função implementa o algoritmo de Krawczyk. A seção 3.9 analisa a precisão e eficiência da função *intsolve* em estudos computacionais.

O leitor interessado somente no algoritmo de Hansen e Greenberg pode ler as seções 3.2 - 3.5 e 3.9. Ao leitor interessado nos teoremas de Moore e algoritmo de Krawczyk recomendamos a leitura das seções 3.2 e 3.6 - 3.8.

3.2 Notação

Denotamos por J a matriz jacobiana de f. Pontos reais serão denotados por letras minúsculas, por exemplo x. Intervalos são representados por letras minúsculas em negrito, por exemplo $\mathbf{x}$. Escrevemos x^I nos casos em que temos um ponto real e desejamos representá-lo como intervalo. Por exemplo, entendemos que 0.1^I é o intervalo

```
>> intval(0.1);
intval ans =
[0.0999,0.1001]
```

dado pelo *INTLAB*. Denotamos caixas por letras latinas maísculas, por exemplo X e Y.

A terna $(M, \mathfrak{f}, \mathfrak{J})$ será chamada de extensão. Em uma extensão, M é o conjunto de números representáveis em nosso computador. As funções $\mathfrak{f} : \mathbb{I}(\mathbb{M})^n \to \mathbb{I}(\mathbb{M})^n$ e $\mathfrak{J} : \mathbb{I}(\mathbb{M})^n \to \mathbb{I}(\mathbb{M})^{nxn}$ são extensões intervalares de f e J.

Ao longo do capítulo assumimos que a caixa $X_0 \subseteq \mathbb{R}$ é representável. Isto quer dizer que X_0 é um vetor intervalar onde todas as entradas são representáveis no conjunto $\mathbb{M}^n$.

3.3 Newton Intervalar

Nesta seção apresentamos o método de Newton intervalar. A versão real deste método é famosa por gerar uma seqüência de pontos que converge quadraticamente para x^* tal que $f(x^*) = 0$, desde que algumas hipóteses sobre f e o ponto inicial x_0 sejam satisfeitas.

Analogamente, a versão intervalar do método de Newton gera uma seqüência de caixas cuja união contém **todas** as raízes de f em X_0. Assim como no caso real, devemos ter algumas hipóteses sobre f e X_0 para garantir este resultado. A principal diferença entre o caso real e intervalar é que ainda não conhecemos resultados sobre ordem de convergência no caso intervalar.

Para que nossa exposição fique completa, começamos por demonstrar o teorema do valor médio intervalar. Em seguida usamos este resultado para estender o método de Newton. Assim como no

caso real, o método de Newton exige a solução de um sistema de equações lineares a cada iteração e por esta razão concluímos a seção apresentando a versão intervalar do método de Gauss Seidel. Ela permite resolver sistemas lineares intervalares e será útil em nossa implementação do método de Newton.

3.3.1 Teorema do Valor Médio

Nesta subseção apresentamos o teorema do valor médio intervalar. Em sua versão real o resultado é válido somente para função univariadas. Mostramos que no caso intervalar ele continua válido para funções multivariadas.

Teorema 1 *Sejam $x, y \in X_0$. Se f, J e $(M, \mathfrak{f}, \mathfrak{J})$ estão definidos como em 3.2 então*

$$f(y) \in \mathfrak{f}(x^I) + \mathfrak{J}(X_0)(y - x) \tag{3.1}$$

Demonstração: Seja $i = 1, \ldots n$ e considere a função $h_i(t) = f_i(x + t(y - x))$. Segue do teorema do valor médio real que

$$h_i(1) = h_i(0) + h_i'(\varepsilon_i)(1 - 0). \text{ Para algum } \varepsilon_i \in [0, 1].$$

Se chamamos de J_i o vetor gradiente de f_i então a igualdade acima nos dá

$$h_i(1) = h_i(0) + J_i(x + \varepsilon_i(y - x))(y - x). \text{ Para algum } \varepsilon_i \in [0, 1].$$

Note que $h_i(1) = f_i(y)$ e $h_i(0) = f_i(x)$. Como x e y pertencem a X_0 e este é um conjunto convexo então $x + \varepsilon_i(y - x) \in X_0$. Além disso como $\mathfrak{f}$ e $\mathfrak{J}$ são extensões intervalares de f e J temos

$$f_i(y) \in \mathfrak{f}_i(x) + \mathfrak{J}_i(X_0)(y - x).$$

Agrupando as i expressões em forma matricial segue o resultado. □

3.3.2 O Método de Newton

Se $x, y \in X_0$ e y é um zero de f então

$$0 = f(y) \in \mathfrak{f}(x^I) + \mathfrak{J}(X_0)(y - x). \tag{3.2}$$

Segue daí que fixado $x \in X_0$, o conjunto

$$S_x := \{z \in X_0 \text{tal que existe } f^* \in \mathfrak{f}(x^I) \text{ e tal que} J^* \in \mathfrak{J}(X_0) \text{ tais que } f^* + J^*(z-x) = 0\} \quad (3.3)$$

contém todos os pontos $y \in X_0$ tais que $f(y) = 0$. Este resultado não depende da escolha de x. Isto é, para qualquer $x \in X_0$ tomado, o conjunto S_x contém todas as raízes de f em X_0.

O método de Newton intervalar é um procedimento iterativo que gera caixas que limitam S_x. Mais especificamente, o método de Newton intervalar gera uma seqüência de caixas tais que

$$X_0 \supseteq X_1 \supseteq \ldots \supseteq S_x,$$

onde cada caixa é da forma

$$X_{k+1} = X_k \cap N(X_k). \quad (3.4)$$

Pela construção temos que $X_0 \supseteq X_1 \supseteq \ldots$ e agora vamos mostrar como construir $N(X)$ que garantem que todas as caixas X_k contém S_x. Esta construção não é única e estamos seguindo as idéias de Hansen e Sengupta (1981).

Pela relação 3.2 garantimos que $X_{k+1} \supseteq S_x$ desde que $N(X_k)$ seja uma caixa limitante da solução do sistema linear intervalar

$$\mathfrak{J}(X_0)(y - x_k) = -\mathfrak{f}(x_k^I). \quad (3.5)$$

Este sistema fica bem definido se determinarmos x_k. Como temos liberdade na escolha desse ponto vamos tomá-lo como sendo uma aproximação do ponto médio da caixa X_k. Denotamos esta aproximação por $\bar{x}$.

Nosso objetivo é encontrar uma caixa limitante do conjunto solução de (3.5). Segundo Hansen e Smith (1967) este problema é equivalente a encontrar uma caixa limitante da solução do sistema pré-condicionado

$$M(y - \bar{x}) = b \quad (3.6)$$

onde $M = mid(J(X_0))^{-1}\mathfrak{J}(X_0)$ e $b = -mid(J(X_0))^{-1}\mathfrak{f}(\bar{x}^I)$.

Para resolver (3.6) utilizamos uma versão intervalar do método de Gauss Seidel. Este é um método iterativo e precisa de uma caixa inicial. Em nossas aplicações, para obter a caixa X_{k+1} na seqüência de Newton, utilizamos o método de Gauss Seidel a partir da caixa X_k. Esta é a razão pela qual o operador N tem X_k como parâmetro de entrada em (3.4).

3.3.3 Método de Gauss Seidel

Passamos agora a descrever o método de Gauss Seidel intervalar. Suponha que temos uma caixa inicial X e desejamos procurar as soluções de (3.6). Este é um método iterativo e seu passo geral é dado por

$$X_i^{'} = X_i \cap Y_i, \qquad i = 1 \ldots n, \tag{3.7}$$

onde

$$Y_i = \bar{x} + \frac{R_i}{M_{ii}} \tag{3.8}$$

e

$$R_i = b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} M_{ij}(X_j - \bar{x}_j). \tag{3.9}$$

Na prática o intervalo M_{ii} pode conter zero. Isto nos leva a dividir o algoritmo de Gauss Seidel intervalar em dois passos. Neste momento nossa definição de divisão estendida na seção 2.7 será importante.

O primeiro passo é o de Gauss Seidel não estendido. Ele consiste em aplicar o esquema (3.7) nas linhas i em que $0 \notin M_{ii}$. Neste estágio garantimos que Y_i é um intervalo pois não há divisão por intervalos que contenham o valor nulo. Segue daí que $X_i^{'}$ pode ser um intervalo ou o conjunto vazio. No primeiro caso obtemos uma nova estimativa para S_x na direção i que é no mínimo tão boa quanto a anterior. Caso $X_i^{'}$ seja vazio podemos excluir a caixa X pois não há soluções em seu interior ou bordas.

O primeiro estágio do método de Gauss Seidel pode ser aplicado enquanto a caixa resultante $X^{'}$ é diferente da caixa de entrada X. Note que pelas conclusões do parágrafo anterior $X^{'} \neq X$ implica que a região de busca foi reduzida parcial ou completamente.

O segundo estágio é o de Gauss Seidel estendido. Ele consiste em aplicar o esquema (3.7) nas linhas i em que $0 \in M_{ii}$. Neste estágio utilizamos divisão estendida e Y_i pode ser um intervalo ou dois. Conseqüentemente $X_i^{'}$ será um intervalo, dois intervalos disjuntos ou o conjunto vazio. No caso em que temos um intervalo ou o conjunto vazio as conclusões são as mesmas do estágio não estendido.

Caso existam índices i em que o resultado são dois intervalos disjuntos, procuramos aquele em que a distância entre os intervalos é a maior. Salvamos este índice e os intervalos resultantes x e y nesta direção. Esta informação será útil ao dividirmos a caixa. O estágio de Gauss Seidel estendido

é aplicado apenas uma vez a cada iteração do método de Newton.

3.4 Método de Hansen e Greenberg

Nesta seção apresentamos o método de Hansen e Greenberg. Este método limita todas as soluções de f em X_0 por caixas de tamanho pré-fixado. Apresentamos o algoritmo por partes. No final da seção demonstramos que se $y \in X_0$ é raiz de f então nosso algoritmo encontra ao menos uma caixa de tamanho pré-fixado que contém y. Nosso trabalho é baseado no artigo de Hansen e Greenberg (1983).

3.4.1 Visão Geral

Nesta subseção apresentamos o escopo geral do método de Hansen e Greenberg. É nosso objetivo que o leitor entenda os passos do algoritmo e como eles se encaixam em um esquema geral.

Nosso algoritmo mantém duas listas. A primeira, que chamamos de P, é a de caixas que ainda devem ser processadas. No inicio do programa temos $P = X_0$. A segunda, que chamamos de S, contém as caixas que salvamos como solução. Esta lista começa vazia. Segue o esquema geral do algoritmo.

1. **Passo de escolha**: Se P está vazia então encerramos o programa. Caso contrário escolhemos uma caixa em P e a chamamos de X. Excluímos X de P.

2. **Passo de eliminação**: Testamos a existência de raízes de f em X. Se provarmos que não há raízes na caixa então excluímos X e voltamos ao passo 1.

3. **Passo de redução**: Aplicamos o método de Newton intervalar para reduzir a região de busca. O resultado é a caixa X_1. Se X_1 é vazia então excluímos X e voltamos ao passo 1. Se X_1 pode ser salva como solução então guardamos X_1 em S e voltamos ao passo 1.

4. **Passo de divisão**: Dividimos X_1 em duas caixas X_1' e X_1''. Salvamos estas caixas no fim da lista P e voltamos ao passo 1.

Nas próximas subseções descrevemos cada passo em detalhes.

3.4.2 Passo de Escolha

Neste passo escolhemos o elemento de P que será processado na iteração atual. Existem diversas estratégias para o passo de escolha. Neste texto apresentamos apenas as mais comuns. O leitor interessado pode consultar Pedamallu et al. (2008).

Nos algoritmos para sistemas de equações não lineares, a escolha da caixa a ser processada baseia-se no comprimento $W(X)$ de cada elemento de P. As duas estratégias mais comuns são:

- Tomar o último elemento de P. Os elementos nesta posição já foram reduzidos e/ou vieram do passo de divisão. Esta estratégia prioriza caixas com valores pequenos de $W(X)$.

- Tomar o primeiro elemento de P. Os elementos nesta posição tendem a ser maiores pois caixas reduzidas e/ou divididas são armazenadas no fim de P. Esta estratégia valoriza caixas com valores altos de $W(X)$.

Métodos que privilegiam caixas de menores devem a percorrer porções menores de X_0 em um número fixo de iterações e tendem a encontrar mais soluções nestes casos. Por outro lado, métodos que privilegiam caixas maiores percorrem grandes porções de X_0 em um número fixo de iterações mas estão proprensas a encontrar um número menor de soluções nestes casos.

Na construção da função *intsolve* optamos pela estratégia de tomar o último elemento de P. Esta escolha é eficiente e baseia-se na suposição de que o usuário do programa está disposto a realizar apenas um número fixo de avaliações de função ou iterações do esquema descrito acima.

3.4.3 Passo de Eliminação

A propriedade de inclusão garante que se $0 \notin \mathfrak{f}(X)$ então $0 \notin \mathfrak{f}(X')$ para qualquer $X' \subseteq X$. Esta propriedade nos dá um teste simples para existência de zeros em X.

- Se $0 \notin \mathfrak{f}(X)$ então não existe $x \in X$ tal que $f(x) = 0$ e excluímos esta caixa.

Lembre que $\mathfrak{f}(X)$ é um vetor intervalar e $0 \notin \mathfrak{f}(X)$ significa que ao menos uma entrada de $\mathfrak{f}$ não contém o valor nulo.

3.4.4 Passo de Redução

Neste passo aplicamos o método de Newton intervalar descrito em 3.3. Nosso objetivo é reduzir a caixa onde procuramos raízes. Inicialmente aplicamos o primeiro estágio do algoritmo de Gauss Seidel intervalar enquanto houver redução no domínio de busca e no máximo por η vezes. Em nossos experimentos tomamos sempre $\eta = 5$.

O cuidado de definir um número máximo de aplicações do primeiro estágio de Gauss Seidel garante que nosso algoritmo deixa este passo em algum momento. Esta escolha também garante que não tenhamos gastos excessivos com o cálculo de caixas que reduzem pouco o domínio de busca.

Aplicamos então o segundo estágio do algoritmo de Gauss Seidel intervalar. Lembre que este estágio é aplicado apenas uma vez a cada iteração do método de Newton. Se este estágio do método devolver um índice i e dois intervalos disjuntos $\mathbf{x}$ e $\mathbf{y}$, conforme descrito em 3.3 então salvmos estas informações para usa-las no passo de divisão.

3.4.5 Passo de Divisão

Se uma caixa não pode ser processada ou não foi reduzida o suficiente então nós a dividimos. Vários autores discutiram heurísticas para divisão de caixas. Os trabalhos de Kearfott (1996b), Boyd, Gosh e Magnani (2003) e Pedamallu et al. (2008) são boas referências para o assunto.

Em nosso algoritmo adotamos um esquema de bisseção simples, considerando dois casos.

1. Não temos informações sobre saltos na caixa. Neste casos a dividimos em X' e X'' a partir da direção com maior comprimento.
2. Sabemos que X veio do passo de redução e nele salvamos o índice de maior salto durante a aplicação do segundo estágio de Gauss Seidel. Neste caso dividimos X em X' e X'' nessa direção. Se i é o índice dessa direção e $\mathbf{x}$ e $\mathbf{y}$ são os intervalos resultantes então tomamos $X'_i = \mathbf{x}$ e $X''_i = \mathbf{y}$.

3.4.6 Tomando Uma Caixa Como Solução

A caixa $X \subseteq X_0$ é solução do nosso problema se $W(X) < \varepsilon_X$ e $||\mathbf{f}(X)|| < \varepsilon_F$. Os valores positivos ε_X e ε_F são dados pelo usuário e não devem ser menor do que a unidade de arredondamento definida por M.

Estas condições controlam o comprimento máximo de X e a maior distância permitida de $\mathbf{f}(X)$ ao vetor nulo. Podemos omitir uma destas condições tomando ε_X ou ε_F como infinito.

Tomar $\varepsilon_X = \infty$ aumenta a eficiência do algoritmo nos casos em que sabemos que f tem muitas raízes em X_0. Não recomendamos tomar valores altos para ε_F pois isto introduz ruídos nos resultados. Outros critérios para aceitar X como solução são discutidos por Hansen e Walster (2004), Hargreaves (2002) e Ratschek e Rokne (1988).

3.4.7 O Algoritmo

Apresentamos agora detalhadamente o método de Hansen e Greenberg. O algoritmo deve ser seguido em seqüência exceto quando indicamos o contrário. Omitimos os passos em que testamos se o número máximo de iterações ou avaliações de função foi atingido. Não é difícil incorporar estes passos ao algoritmo.

Entradas

- A extensão $(M, \mathbf{f}, \mathfrak{J})$
- A caixa inicial representável X_0.
- Valores ε_X e ε_F compatíveis com M. Veja a subseção anterior.

Saídas

- Um conjunto de caixas $S := \{S_1, \ldots, S_k\}$ tais que se x^* é raiz de f então x^* pertence a pelo menos um elemento de S.

Não há relação de $1-1$ entre as raízes de f e os elementos de S. É possível que uma mesma raiz esteja em mais de uma caixa e também é possível que S tenha caixas que não contém raízes.

Passos do Algoritmo

1. Se P está vazia então encerre o programa. Caso contrário tome a última caixa de P e chame-a de X. Exclua X de P.
2. Avalie $\mathfrak{f}(X)$.
3. Se alguma entrada de $\mathfrak{f}(X)$ não contém zero então exclua X e volte ao passo 1.
4. Se $W(X) < \varepsilon_X$ e $||\mathfrak{f}(X)|| < \varepsilon_F$ então guarde X em S e volte ao passo 1.
5. Avalie $\mathfrak{J}(X)$ e tome $A = mid(\mathfrak{J}(X))$.
6. Estime os valores singulares A com aritmética de ponto flutuante. Chame de σ_1 o menor destes valores e σ_n o maior.
7. Se $\frac{\sigma_n}{\sigma_1} < \nu$ então divida X na direção de maior comprimento. Guarde as duas caixas no fim da lista P e volte ao passo 1. Em nossos experimentos usamos $\nu = 10^{-6}$.
8. Estime $\bar{x} = mid(X)$ e $\mathfrak{f}(\bar{x}^I)$. Tome $M = A^{-1}\mathfrak{J}(X)$ e $b = -A^{-1}\mathfrak{f}(\bar{x}^I)$.
9. Aplique o estágio não estendido de Gauss Seidel ao sistema $M(z - \bar{x}) = b$ tomando X como estimativa inicial. O resultado é uma nova caixa X_1.
10. Se alguma entrada de X_1 é vazia então exclua X e volte ao passo 1.
11. Avalie $\mathfrak{f}(X_1)$.
12. Se algum elemento de $\mathfrak{f}(X_1)$ não contém zero então exclua X e volte ao passo 1.
13. Se $W(X_1) < \varepsilon_X$ e $||\mathfrak{f}(X_1)|| < \varepsilon_F$ então guarde X_1 em S e volte ao passo 1.
14. Estime $\bar{x} = mid(X_1)$ e avalie $\mathfrak{f}(\bar{x}^I)$. Tome $b = -A^{-1}\mathfrak{f}(\bar{x}^I)$.
15. Se $X_1 \neq X$ e aplicamos o passo 8 menos do que $\eta + 1$ vezes então substitua $X = X_1$. Volte ao passo 8.
16. Aplique o estágio estendido de Gauss Seidel ao sistema $M(z - \bar{x}) = b$ tomando X_1 como estimativa inicial. O resultado é uma nova caixa X_2. Salve a direção em que ocorre o maior salto e os intervalos resultantes nessa direção.

17. Se algum elemento de X_2 é vazio então exclua X e volte ao passo 1.

18. Avalie $\mathfrak{f}(X_2)$.

19. Se algum elemento de $\mathfrak{f}(X_2)$ não contém zero então exclua X e volte ao passo 1.

20. Se $W(X_2) < \varepsilon_X$ e $||\mathfrak{f}(X_2)|| < \varepsilon_F$ então guarde X_2 em S e volte ao passo 1.

21. Se nenhum salto foi salvo então divida X_2 na direção de maior comprimento. Guarde as duas caixas em P e volte ao passo 1.

22. Se algum salto foi salvo então divida X_2 na direção deste salto. Utilize os intervalos resultantes nessa direção. Volte ao passo 1.

3.4.8 Convergência do Algoritmo

Concluímos esta seção mostrando que o algoritmo acima limita todas as raízes de f em X_0 desde que o número máximo de iterações ou avaliações de função não seja atingido.

Teorema 2 *Sejam $f : \mathbb{R}^n \to \mathbb{R}^n$ uma função diferenciável, $(M, \mathfrak{f}, \mathfrak{J})$ uma extensão e X_0 uma caixa representável em M. Suponha que ε_X e ε_F são números positivos consistentes com M. Então o algoritmo acima limita todas as raízes de f em X_0 por caixas $X_i \subseteq X_0$ tais que $W(X_i) < \varepsilon_X$ e $||f(X_i)|| < \varepsilon_F$ em um número finito de iterações.*

Demonstração: Como $\mathfrak{f}$ e $\mathfrak{J}$ são extensões intervalares de f e J garantimos que todos os cálculos envolvendo estas funções são verificados. Desta forma não perdemos soluções por erros de arredondamento. Pela mesma razão, o passo de eliminação não perde soluções. Já mostramos que o passo de redução limita o conjunto $S_{\bar{x}}$ que contém todas as soluções de f em X. Como os cálculos são verificados concluímos que este passo também não perde soluções. Os demais passos não excluem caixas ou subcaixas e portanto não podem excluir soluções do problema.

Os passos de divisão e redução garantem que uma caixa que não foi excluída será reduzida. Como ε_X e ε_F são consistentes com M garantimos que uma caixa que não foi excluída deverá ser aceita como solução em um número finito de iterações. Isto conclui a demonstração.□

3.5 A Função intsolve

A função *intsolve* do *INTSOLVER* implementa o algoritmo descrito na seção anterior. Nesta seção ilustramos seu uso e algumas de suas opções.

O *INTSOLVER* deve ser simples para usuários acostumados ao Matlab. A função *intsolve* exige poucos parâmetros de entrada. Por exemplo, desejamos encontrar todas as raízes do sistema

$$x_1^2 + x_2^2 - 1 = 0$$
$$x_1^2 - x_2^2 = 0$$

na caixa $-5 \leq x_1, x_2 \leq 5$. Para realizar esta tarefa com o *intsolve* precisamos apenas da função f e da caixa X_0.

```
>> f = @(x) [x(1)^2+ x(2)^2 - 1, x(1)^2 - x(2)];
>> x0 = [infsup(-5,5), infsup(-5,5)];
>> x  = intsolve(f, x0)
   ite    evf    evj    evinv    nsol    nlist    completed(%)

   0      0      0      0        0       1        0.000

   10     87     10     6        1       3        0.248

   20     112    20     11       1       3        0.436

   30     199    30     17       2       3        0.595

   40     224    40     22       2       3        0.901
intval x =
      [0.7861, 0.7862] [0.6180, 0.6181]
      [-0.7862, -0.7861] [0.6180, 0.6181]
```

A função *intsolve* assume os seguintes padrões,

- Número máximo de iterações: 1000.
- Número máximo de Avaliações de função: 10000.
- $\varepsilon_X = 10^{-8}$.
- $\varepsilon_F = 10^{-8}$.
- Nível do Display: 10.
- Iterar até limitar todas as raízes: Não.

Todas os parâmetros acima podem ser modificados. Em particular quando optamos por iterar até limitar todas as raízes a função ignora quaisquer valores em número máximo de iterações e número máximo de avaliações de função. Obviamente esta escolha deve aumentar o tempo de execução do algoritmo e cabe ao usuário usá-la coerentemente.

Para alterar os parâmetros de execução utilizamos a função *intoptimset*. No próximo exemplo aumentamos as quantidades ε_X e ε_X para 10^{-10}. Também atribuímos o valor 0 ao nível do display de forma que as informações sobre o processo não serão exibidas.

```
>> opt = intoptimset();
>> opt(1) = 10^-10;
>> opt(2) = 10^-10;
>> opt(8) = 0;
>> [x, iv, ef] = intsolve(f, x0, opt)

intval x =
[0.7861,0.7862] [0.6180,0.6181]
[-0.7862,-0.7861] [0.6180,0.6181]

intval iv =
  1.0e-010 *
[-0.4586,0.7405] [-0.6387,0.4511]
[-0.4586,0.7405] [-0.6387,0.4511]

ef =
     0
```

A saída x mostra as caixas soluções encontradas pela função. Em iv temos $f(X_i)$ onde X_1 e X_2 são as caixas em x. Finalmente o parâmetro de saída $ef = 0$ nos diz que a função foi finalizada com sucesso.

As funções do *INTSOLVER* estão documentadas e o usuário que desejar maiores informações deve digitar

```
>> doc nome_da_função
```

no Matlab.

3.6 Teoremas de Moore

Nesta seção apresentamos os teoremas de Moore (1977). Estes foram os primeiros resultados a explorar a aritmética intervalar na análise de sistemas de equações não lineares. O primeiro

teorema nos dá uma condição de suficiência para existência de soluções de f em X_0. O segundo nos dá condições suficientes para unicidade de solução.

O prêmio Nobel em economia L. Kantorovich é conhecido por analistas numéricos por seu teorema que, independente de Moore, dá condições suficientes para existência e unicidade de soluções de f em X_0. Sendo assim a comparação entre os teoremas de Moore e Kantorovich merece alguns comentários. Rall (1980) os comparou e concluiu que o teorema de Kantorovich é mais sensível e preciso do que o de Moore. Isto quer dizer que o teorema de Kantorovich é mais eficaz ao identificar a existência de soluções e dá limitantes melhores de erro. Por outro lado, o custo para verificar as condições de Moore é muito menor e faz com que esta seja uma boa opção nas aplicações onde o teorema de Kantorovich é usado.

Lema 1 *Sejam f, J e $(M, \mathfrak{f}, \mathfrak{J})$ definidos como na seção 3.2. Seja Y uma matriz real não singular e considere o operador $P(x) = x - Yf(x)$. Se*

$$\bar{P}(X) \subseteq X \tag{3.10}$$

então f tem ao menos uma raiz em X.

Demonstração: A continuidade de f implica a continuidade de P. Como X é um conjunto compacto e convexo e $P(X) \subseteq X$, o teorema de ponto fixo de Brouwer garante que P tem pelo menos um ponto fixo em X. Se $x^{'}$ é este ponto temos $P(x^{'}) - x^{'} = 0$. Como Y é não singular segue o resultado.□

Teorema 3 (Existência) *Sejam f, J e $(M, \mathfrak{f}, \mathfrak{J})$ definidos como em 3.2. Sejam $y \in X$ e*

$$K(X) := y - Y\mathfrak{f}(y^I) + \{Id - Y\mathfrak{J}(X)\}(X - y) \tag{3.11}$$

onde Y é uma matriz real não singular e Id a matriz identidade. Se $K(X) \subseteq X$ então existe ao menos uma solução de f em X.

Demonstração: Seja P o operador do teorema anterior. Somando e subtraindo $y - Yf(y)$ obtemos

$$P(x) = y - Yf(y) + x - y - Y(f(x) - f(y)). \tag{3.12}$$

O teorema do valor médio intervalar garante que para qualquer $x, y \in X$ temos

$$f(x) - f(y) \in J(X)(x - y). \tag{3.13}$$

Aplicando 3.13 em 3.12 temos

$$P(x) \in y - Yf(y) + \{I - YJ(X)\}(X - y) \tag{3.14}$$

para qualquer $x \in X$. Segue daí que $P(x) \subseteq K(X)$. Se $K(X) \subseteq X$ as hipóteses do teorema anterior são válidas e existe ao menos uma solução de f em X.□

O algoritmo de Krawczyk é dado por

$$X_{k+1} = X_k \cap K(X_k),\ k = 0, 1, \ldots, \tag{3.15}$$

onde

$$K(X_k) = \bar{x}_k - Y_k \mathfrak{f}(\bar{x}_k^I) + \{I - Y_k \mathfrak{J}(X_k)\}(X_k - \bar{x}_k) \tag{3.16}$$

e

$$Y_k = \begin{cases} mid(\mathfrak{J}(X_k))^{-1} & \text{se } ||I - mid(\mathfrak{J}(X_k))^{-1} * \mathfrak{J}(X_k)|| \leq r_{k-1} \\ Y_{k-1} & \text{caso contrário} \end{cases}$$

com $r_k = ||I - Y_k \mathfrak{J}(X_k)||$ e $\bar{x} = mid(X_k)$.

O segundo teorema de Moore nos dá condições suficientes para a existência de uma única raiz de f em X. Ele também mostra que o algoritmo de Krawczyk gera uma seqüência de caixas cada vez menor que limitam esta raiz. A demonstração deste teorema é apresentada em Moore (1977)

Teorema 4 (Unicidade) *Sejam f, J e $(M, \mathfrak{f}, \mathfrak{J})$ definidos como em 3.2. Se*

$$K(X_0) \subseteq X_0 \qquad e \qquad r_0 = ||I - Y_0 \mathfrak{J}(X_0)|| < 1$$

então existe uma única raiz x^* de f em X_0. Além disso

1. *$x^* \in X_k \subseteq X_{k-1}$ para $k = 1, 2, \ldots$.*
2. *$W(X_0) \leq r_0^k W(X_0)$.*

3.7 Método de Krawczyk

As condições de Moore foram implementadas pela primeira vez por R Krawczyk. Nesta seção apresentamos este algoritmo que será útil para verificar a existência de zeros de f em X_0. O

algoritmo deve ser seguido em seqüência exceto quando indicamos o contrário. Omitimos os passos em que testamos se o número máximo de iterações e avaliações de função foi atingido. Não é difícil incorporar estes passos ao algoritmo.

Entradas

- A extensão $(M, \mathfrak{f}, \mathfrak{J})$
- A caixa inicial representável X_0.

Saídas

- A variável $FLAG$ que pode assumir os valores:

  -1 Se o teorema de Moore garante a existência de soluções de f em X.

  0 Se o teorema de Moore garante a unicidade de solução de f em X.

  1 Se o algoritmo alcançou o número máximo iterações.

  2 Se o algoritmo alcançou o número máximo de avaliações de função.

  3 Se as condições de Moore não estão satisfeitas.

- Se $FLAG = 0$ devolve um limitante X^* da única solução de f em X.

Passos do Algoritmo

1. Tome $FLAG = 3$.
2. Avalie $\mathfrak{f}(X_0)$ e $\mathfrak{J}(X_0)$.
3. Tome $\bar{x} = mid(X_0)$ e $Y = mid(\mathfrak{J}(X_0))^{-1}$.
4. Avalie $K = \bar{x} - Y * \mathfrak{f}(\bar{x}^I) + (I - Y\mathfrak{J}(X_0))(X_0 - \bar{x})$.
5. Se $K \not\subseteq X_0$ então encerre o programa. Caso contrário flag $FLAG = -1$.
6. Avalie $r = ||I - Y\mathfrak{J}(X)||$. Se $r \geq 1$ encerre o programa. Caso contrário faça $FLAG = 0$.
7. Tome $X^* = K \cap X_0$.
8. Se $X^* == X_0$ então encerre o programa.
9. Tome $X_0 = X^*$
10. Avalie $\mathfrak{f}(X_0)$ e $\mathfrak{J}(X_0)$.
11. Tome $\bar{x} = mid(X_0)$.

12. Tome $Y' = mid(\mathfrak{J}(X_0)^{-1}$. Se $||I - Y'\mathfrak{J}(X_0)|| \leq r$ então $Y = Y'$ e $r = ||I - Y'\mathfrak{J}(X_0)||$.

13. Avalie $K = \bar{x} - Y * \mathfrak{f}(\bar{x}^I) + (I - Y\mathfrak{J}(X))(X - \bar{x})$.

14. Tome $X^* = K \cap X_0$. Volte ao passo 8.

3.8 A Função intksolve

O algoritmo *intksolve* do *INTSOLVER* implementa o algoritmo de Krawczyk descrito na seção acima. Nesta seção ilustramos seu uso. Considere o sistema de equações não lineares

$$\begin{aligned} x_1^2 + x_2^2 - 1 &= 0, \\ x_1^2 - x_2 &= 0. \end{aligned}$$

Desejamos saber se este sistema tem zeros na caixa

$$X = \begin{pmatrix} [0.7, 0.9] \\ [0.5, 0.7] \end{pmatrix}. \tag{3.17}$$

Para realizar esta tarefa com o *intksolve*, o usuário fornece a função $\mathfrak{f}$ e a caixa X.

```
>> f  = @(x) [x(1)^2+ x(2)^2 - 1, x(1)^2 - x(2)];
>> X = [infsup(.7, .9), infsup(.5, .7)];

intval x =
[0.7861,0.7862] [0.6180,0.6181]
intval iv =
[0.0000,0.0000] [0.0000,0.0000]
ef =
     0
```

O valor $ef = 0$ nos diz que as condições de Moore estão satisfeitas e existe apenas uma solução de f em X. A saída x é um limitante para esta raiz e iv nos dá $\mathfrak{f}(x)$. O *intksolve* assume os seguintes padrões,

- Número máximo de iterações: 1000.
- Número máximo de Avaliações de função: 10000.

- Nível do Display: 10.
- Iterar até obter resposta: Não.

Todos os parâmetros acima podem ser modificados. Em particular quando optamos por iterar até obter resposta, a função ignora quaisquer valores em número máximo de iterações e número máximo de avaliações de função. Obviamente esta escolha deve aumentar o tempo de execução do algoritmo e cabe ao usuário seu uso coerente.

Para alterar os parâmetros de execução utilizamos a função *intoptimset*. Veja o exemplo da seção 3.5 para maiores detalhes. As funções do *INTSOLVER* estão documentadas e o usuário que desejar maiores informações deve digitar

```
>> doc nome_da_função
```

no Matlab.

3.9 Estudos Computacionais

Nesta seção estudamos o desempenho da função *intsolve* do *INTSOLVER*. Esta função implementa o algoritmo descrito na seção 3.4. Para facilitar a análise de desempenho tomamos 7 funções da literatura de otimização global cujas raízes são conhecidas e apenas um com soluções não reportadas. Os problemas foram extraídos principalmente do livro de Floudas et al. (1999) e são reportados como de difícil solução. Discutimos um caso onde o autor perde soluções do problema ao considerar métodos com aritmética de ponto flutuante.

A menos de menção contrária, o número máximo de iterações em cada teste é 1000, o número máximo de avaliações de função é 10000 e ε_X, ε_F iguais a 10^{-8}. As funções foram implementadas em Matlab exatamente como aparecem descritas abaixo. A extensão é feita apenas trocando variáveis reais por intervalos. Os testes foram realizados em um processador AMD Sempron de $800MHZ$ e $1GB$ de memória RAM com sistema operacional Linux.

3.9.1 Himmelblau

Este problema foi extraído de Floudas et al. (1999). O objetivo é encontrar todas as raízes do sistema de Himmelblau.

$$4x_1^3 + 4x_1x_2 + 2x_2^2 - 42x_1 - 14 = 0,$$
$$4x_2^3 + 2x_1^2 + 4x_1x_2 - 26x_2 - 22 = 0.$$

na caixa $-5 \leq x_1, x_2 \leq 5$.

O autor reporta as seguintes soluções

x_1	3.0000	3.3852	0.0867	3.5844	-2.8051	-0.2708	-0.1280	-3.0730	-3.7793
x_2	2.0000	0.0739	2.8843	-1.8481	3.1313	-0.9230	-1.9537	-0.0814	-3.2832

Aplicando o *intsolve* obtemos

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
18.187	79	700	79	9	100

Reportamos o ponto médio de cada caixa solução, isto nos permite comparar nosso resultado ao de Floudas. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas soluções e não aos pontos reportados.

x_1	3.0000	3.3852	0.0867	3.5844	-2.8051	-0.2708	-0.1280	-3.0730	-3.7793
x_2	2.0000	0.0739	2.8843	-1.8481	3.1313	-0.9230	-1.9537	-0.0814	-3.2832
$\|\|f(X)\|\|$	5.4e-10	4.9e-10	1.6e-12	5.1e-13	9.4e-10	7.5e-10	1.7e-10	1.1e-9	1.2e-9
$\mathbb{W}(X)$	7.6e-11	1.2e-10	2.3e-12	1.0e-12	7.9e-11	2.7e-10	1.4e-10	5.8e-11	4.4e-11

3.9.2 Brown Almost Linear

Este problema foi extraído de Moré, Garbow e Hillstrom (1981). Desejamos encontrar todas as raízes do sistema de equações quase linear de Brown

$$\begin{aligned} 2x_1 + x_2 + x_3 + x_4 + x_5 - 6 &= 0, \\ x_1 + 2x_2 + x_3 + x_4 + x_5 - 6 &= 0, \\ x_1 + x_2 + 2x_3 + x_4 + x_5 - 6 &= 0, \\ x_1 + x_2 + x_3 + 2x_4 + x_5 - 6 &= 0, \\ x_1x_2x_3x_4x_5 - 1 &= 0, \end{aligned}$$

na caixa $-2 \leq x_1, \ldots, x_5 \leq 2$.

Floudas et al. (1999) reporta as seguintes soluções para este problema

x_1	x_2	x_3	x_4	x_5
0.916	0.916	0.916	0.916	1.418
1.0000	1.0000	1.0000	1.0000	1.0000

Aplicando o *intsolve* com a opção de iterar até a conclusão obtemos.

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
688.50	4661	23267	4661	33	100

Encontramos 33 caixas soluções para este problema. Isto não quer dizer que existam tantas raízes de f em X_0. Neste caso sabemos que o excesso de caixas limitando a raíz $x^* = (1,1,1,1,1)$ se deve ao passo de divisão do algoritmo. Quando aplicamos este passo a uma caixa de dimensão n, as caixas resultantes tem um hiperplano de dimensão $n-1$ em comum. Se x^* pertence a este hiperplano então nosso algoritmo encontra um limitante da raiz para cada caixa que a contém. Uma raiz pode ter até 2^n limitantes, como é o caso de x^* na função de Brown. Este custo é necessário para garantir que nenhuma raiz será perdida ao longo do processo.

Reportamos o ponto médio da caixa que limita a primeira raíz da função. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas de solução e não aos pontos reportados.

x_1	x_2	x_3	x_4	x_5	$\|\|f(X)\|\|$	$\mathbb{W}(X)$
0.9164	0.9164	0.9164	0.9164	1.4182	3.4e-14	5.9e-15

Para facilitar a exposição informamos apenas os valores $||f(X)||$ e $\mathbb{W}(X)$ das 32 caixas que limitam $(1,1,1,1,1)$.

$\|\|f(X)\|\|$	2.4e-14	4.4e-9	3.3e-9	2.5e-9	3.0e-9	2.5e-9	3.2e-9	4.4e-9
$\mathbb{W}(X)$	1.1e-15	8.3e-9	4.1e-9	4.5e-9	3.8e-9	4.5e-9	3.5e-9	6.5e-9

$\|\|f(X)\|\|$	2.5e-9	3.2e-9	4.4e-9	3.2e-9	4.4e-9	2.4e-9	3.0e-9	4.1e-9
$\mathbb{W}(X)$	4.5e-9	3.5e-9	6.5e-9	3.5e-9	6.5e-9	2.8e-9	1.6e-9	5.1e-9

$\|\|f(X)\|\|$	3.2e-9	4.4e-9	3.2e-9	4.4e-9	2.4e-9	3.0e-9	3.2e-9	4.4e-9
$\mathbb{W}(X)$	3.5e-9	6.5e-9	3.5e-9	6.5e-9	2.8e-9	1.6e-9	3.5e-9	6.5e-9

$\|\|f(X)\|\|$	3.0e-9	2.0e-9	3.0e-9	1.2e-9	4.1e-9	2.4e-9	2.5e-9	3.4e-9
$\mathbb{W}(X)$	1.6e-9	2.8e-9	1.6e-9	1.3e-9	1.7e-9	2.8e-9	4.5e-9	4.2e-9

3.9.3 Bullard e Biegler

Este problema aparece em Floudas et al. (1999). Desejamos encontrar todas as raízes de

$$10^4 x_1 x_2 - 1 = 0,$$
$$e^{-x_1} + e^{-x_2} - 1.001 = 0,$$

na caixa $0 \leq x_1, x_2 \leq 100$. Note que este problema é simétrico nas variáveis e portanto se o par (x_1, x_2) é solução então o par (x_2, x_1) também será.

Floudas reporta a seguinte solução

$$x^* = (1.45e^{-5}, 6.8933).$$

Note que o par $x^* = (6.8933, 1.45e^{-5})$ não é reportado como solução e o autor afirma em seu livro ter encontrado a única solução do problema. Acreditamos que este erro seja devido ao uso de aritmética de ponto flutuante na construção do algoritmo αBB porém não tivemos acesso ao código da função para tirar maiores conclusões.

Aplicando nosso método obtemos

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
7.2917	99	407	99	2	100

Reportamos o ponto médio de cada caixa solução, isto permite comparar nosso resultado ao de Floudas. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas de solução e não aos pontos reportados.

x_1	6.8934	1.4507e-05
x_2	1.4507e-05	6.8934
$\|\|f(X)\|\|$	9.1e-9	7.0e-9
$\mathbb{W}(X)$	4.1e-10	1.8e-10

Note que há uma discordância entre nosso resultado e o de Floudas com relação ao último digito significativo de x_1. Esta diferença acontece porque em nosso algoritmo consideramos como critério de parada ε_X e ε_F iguais a 10^{-8} ao passo que Floudas considera como critério de parada em seu método apenas três dígitos significativos.

3.9.4 Ferraris e Tronconi

Este problema foi extraído de Floudas et al. (1999). O objetivo é encontrar todas as raízes do sistema de Ferraris e Tronconi.

$$0.5\sin(x_1x_2) - \frac{0.25x_2}{\pi} - 0.5x_1 = 0$$
$$(1 - \frac{0.25}{\pi})(e^{2x_1} - e) + \frac{ex_2}{\pi} - 2ex_1 = 0$$

na caixa $0.25 \leq x_1 \leq 1$ e $1.5 \leq x_2 \leq 2\pi$.

O autor reporta as seguintes soluções

x_1	0.5	0.29945
x_2	3.1415	2.8369

Com a função *intsolve* obtemos

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
6.7991	53	249	53	2	100

Reportamos o ponto médio de cada caixa solução, isto nos permite comparar nosso resultado ao de Floudas. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas de solução e não aos pontos reportados.

x_1	0.5	0.29945
x_2	3.1415	2.8369
$\|\|f(X)\|\|$	5.2e-9	1.1e-9
$\mathbb{W}(X)$	1.0e-9	5.3e-9

3.9.5 Equilíbrio de Combustão

Este problema foi extraído de Floudas et al. (1999).

$$
\begin{aligned}
x_1x_2 + x_1 - 3x_5 = 0,, \\
2x_1x_2 + x_1 + 3R_{10}x_2^2 + x_2x_3^2 + R_7x_2x_3 + R_9x_2x_4 + R_8x_2 - Rx_5 = 0, \\
2x_2x_3^2 + R_7x_2x_3 + 2R_5x_3^2 + R_6x_3 - 8x_5 = 0, \\
R_9x_2x_4 + 2x_4^2 - 4Rx_5 = 0, \\
x_1x_2 + x_1 + R_{10}x_2^2 + x_2x_3^2 + R_7x_2x_3 + R_9x_2x_4 + R_8x_2 + R_5x_3^2 + R_6x_3 + x_4^2 - 1 = 0,
\end{aligned}
$$

na caixa $0.0001 \leq x_1, \ldots, x_5 \leq 100$.

Os parâmetros do sistema são

$R = 10$	$R_5 = 0.193$	$R_6 = 4.10622e-4$
$R_7 = 5.45177e-4$	$R_8 = 4.4975e-7$	$R_9 = 3.40735e-5$
$R_{10} = 9.615e-7$		

O autor reporta a seguinte solução em (FLOUDAS et al., 1999):

$$x^* = (0.003431, 31.325636, 0.068352, 0.859530, 0.036963)$$

Aplicando nosso método obtemos

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
37838.62	121734	65878	121734	1	100

Reportamos o ponto médio da caixa solução, isto nos permite comparar nosso resultado ao de Floudas. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem à caixa solução e não ao ponto reportado.

x_1	x_2	x_3	x_4	x_5	$\|\|f(X)\|\|$	$\mathbb{W}(X)$
0.00343	31.32649	0.06835	0.85952	0.03696	1.608e-11	2.716e-10

3.9.6 Kubicek

Este problema foi extraído de Floudas et al. (1999). O objetivo é encontrar todas as raízes do sistema

$$(1-R)(C_1 - x_1)e^{\frac{10x_1}{1+10x_1}} - x_1 = 0,$$
$$x_1 - 3x_2 + (1-R)(\frac{d}{10} - 2x_1 - 3x_2)e^{\frac{10x_2}{1+10x_2}} = 0,$$

na caixa $0 \leq x_1, \ldots, x_2 \leq 1$.

Os parâmetros do sistema são

$R = 0.955$	$C_1 = \frac{22}{30}$
$g = 1000$	$d = 22$

O autor reporta as seguintes soluções

x_1	0.72084	0.23633	0.051212	0.051212	0.051212
x_2	0.24453	0.52037	0.68234	0.23514	0.078028

Utilizando o *intsolve* obtemos

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
9.5355	87	395	87	5	100

Reportamos o ponto médio de cada caixa solução, isto nos permite comparar nosso resultado ao de Floudas. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas de solução e não aos pontos reportados.

x_1	0.72084	0.23632	0.051212	0.051212	0.051212
x_2	0.24453	0.52037	0.68234	0.23514	0.078027
$\|\|f(X)\|\|$	8.9e-9	2.5e-9	4.5e-9	4.1e-9	4.7e-9
$\mathbb{W}(X)$	3.6e-09	1.2e-10	4.9e-11	6.9e-10	1.7e-9

3.9.7 Smith

Este problema unidimensional foi extraído de Floudas et al. (1999). O objetivo é encontrar todas as raízes do sistema

$$\frac{b}{T}xe^{\frac{c}{x}} - \frac{b+baT}{aT}e^{\frac{c}{x}} + \frac{x}{T} - 1 = 0$$

na caixa $100 \leq x \leq 1000$.

Os parâmetros do sistema são

$H = -50000$	$a = \frac{-1000}{3H}$	$b = 1.344e9$
$c = -7548.1193$	$T = 298$	

O autor reporta as seguintes soluções

x	300.42
	347.32
	445.50

Utilizando o *intsolve* obtemos

Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
8.7099	111	486	111	3	100

Reportamos o ponto médio de cada caixa solução, isto nos permite comparar nosso resultado ao de Floudas. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas de solução e não aos pontos reportados.

x	445.50	347.32	300.43
$\|\|f(X)\|\|$	7.7e-09	1.5e-10	2.1e-11
$\mathbb{W}(X)$	2.2e-9	3.4e-9	6.0e-9

3.9.8 Equações Trigonométricas

Este problema foi extraído de Moré, Garbow e Hillstrom (1981). A dimensão do sistema é variável e desejamos analisar a performance do *intsolve* com o aumento de n. Nosso objetivo é limitar todas as raízes do sistema

$$f_i(x) = n - \sum_{j=1}^{n} cos(x_j) + i(1 - cos(x_i)) - sin(x_i) = 0$$

para $i = 1 \ldots n$ na caixa $0 \leq x_1, \ldots, x_n \leq 1$. Os autores não reportam soluções para este problema. A tabela abaixo permite avaliar a performance de nosso método

n	Tempo(s)	#Iterações	#Aval. Func.	#Aval. Jac.	#Soluções	Concluído(%)
2	4.1	27	125	27	2	100
3	22.0	107	447	107	2	100
4	129.6	561	2055	561	2	100
5	265.2	1001	3092	1000	0	98.7
6	258.1	1001	2771	1000	0	96.9
7	301.2	1001	2705	1000	0	75.0

Para facilitar a exposição reportamos o ponto médio de cada caixa solução. Os valores $||f(X)||$ e $\mathbb{W}(X)$ se referem às caixas soluções e não aos pontos reportados.

Para $n = 2$ temos

x_1	0.24306	0.00000
x_2	0.61268	0.00000
$\|\|f(X)\|\|$	1.47e-9	1.13e-11
$\mathbb{W}(X)$	7.83e-9	6.92e-12

No caso $n = 3$ obtemos

x_1	0.1386	0.00000
x_2	0.1523	0.00000
x_3	0.4677	0.00000
$\|\|f(X)\|\|$	1.87e-9	1.74e-11
$\mathbb{W}(X)$	7.41e-9	7.49e-12

Finalmente, se $n = 4$ temos

x_1	0.0891	0.00000
x_2	0.0940	0.00000
x_3	0.1003	0.00000
x_4	0.3808	0.00000
$\|\|f(X)\|\|$	2.35e-9	2.72e-9
$\mathbb{W}(X)$	7.33e-9	9.62e-10

Para $n \geq 5$ a função *intsolve* atinge o número máximo de iterações. Note que o número de iterações e o tempo de execução crescem rapidamente quando aumentamos a dimensão do problema o que torna inviável o uso do método para problemas de dimensões maiores.

Capítulo 4

Otimização Irrestrita

4.1 Introdução

Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função duas vezes diferenciável. Neste capítulo estudamos o problema de encontrar **todos** os mínimos globais de f em uma caixa X_0 da forma $l \leq x \leq u$.

Diversos trabalhos usam métodos de convexificação e aritmética de ponto flutuante para resolver este problema. Não discutimos estes métodos neste texto e o leitor interessado pode consultar Floudas (2000) e Tuy (1997).

A aritmética intervalar nos dá uma abordagem rigorosa e natural do assunto. Dentre os autores que usam esta ferramenta em seus algoritmos destacamos Moore (1991), Kearfott (1996b), Hansen (1980), Hansen e Walster (2004), Martinez et al. (2004), Pedamallu et al. (2008) e Ratz e Czendes (1995). Consideramos as idéias destes autores ao longo deste capítulo. Nossos objetivos são:

1. Estudar um algoritmo baseado no método de Newton intervalar que permite limitar todos os mínimos globais de f em X_0.
2. Ilustrar o uso da função *intminunc* do *INTSOLVER*. Esta função implementa o algoritmo desenvolvido ao longo do capítulo.
3. Validar nossa implementação através de estudos computacionais.

Dividimos o capítulo da seguinte forma. Na seção 4.2 apresentamos a notação usada ao longo do capítulo. Em 4.3 apresentamos a visão geral do algoritmo desenvolvido ao longo do capítulo. Na seção 4.4 descrevemos nosso algoritmo de otimização irrestrita em detalhes. A seção 4.5 ilustra o uso da função *intminunc* do *INTSOLVER* que implementa nosso método de otimização irrestrita. Em 4.6 avaliamos a precisão e eficiência da função *intminunc* em estudos computacionais.

As seções 4.2 - 4.4 serão úteis ao leitor que desejar implementar nosso algoritmo. Os interessados no uso de nosso pacote podem ir direto à seção 4.5.

4.2 Notação

Denotamos por g e H o vetor gradiente e a matriz hessiana de f. Pontos reais serão denotados por letras minúsculas, por exemplo x. Intervalos são representados por letras minúsculas em negrito, por exemplo $\mathbf{x}$. Escrevemos x^I nos casos em que temos um ponto real e desejamos representá-lo como intervalo. Por exemplo, entendemos que 0.1^I é o intervalo

```
>> intval(0.1);
intval ans =
[0.0999,0.1001]
```

dado pelo *INTLAB*. Denotamos caixas por letras latinas maíusculas, por exemplo X e Y.

A quádrupla $(M, \mathfrak{f}, \mathfrak{g}, \mathfrak{H})$ será chamada de extensão. Em uma extensão, M é o conjunto de números representáveis em nosso computador e as funções $\mathfrak{f}$, $\mathfrak{g}$ e $\mathfrak{H}$ são extensões intervalares de f, g e H. Ao longo do capítulo assumimos que $X_0 \subseteq \mathbb{R}$ é representável. Isto quer dizer que todos os vértices de X_0 pertencem ao conjunto $\mathbb{M}^n$.

4.3 Visão Geral do Método

Nesta seção apresentamos o escopo geral de nosso método. Esta visão será útil para que o leitor entenda os passos do algoritmo e como eles se encaixam no esquema geral. Todos os passos apresentados aqui são discutidos na próxima seção.

A idéia é procurar caixas $X \subseteq X_0$ tais que $0 \in \mathfrak{g}(X)$ e descartar aquelas que não satisfazem esta condição. Queremos também descartar o mais rápido possível caixas que satisfazem esta condição mas que não contenham mínimos globais.

Nosso algoritmo mantém duas listas. A primeira, que chamamos de P, é a de caixas que ainda devem ser processadas. No inicio do programa temos $P = X_0$. A segunda, que chamamos de C, é a de caixas candidatas à solução. Esta lista começa o programa vazia. Nosso algoritmo também exige um limitante superior para o mínimo global μ. Quando não há nenhuma informação sobre este valor tomamos $\mu = \infty$. Segue o esquema geral do algoritmo.

1. **Passo da escolha**: Se P está vazia então encerramos o programa. Caso contrário escolhemos uma caixa em P e a chamamos de X. Excluímos X de P.

2. **Análise do gradiente**: Testamos a existência de pontos estacionários de f em X. Se provarmos que estes pontos não existem então excluímos X e voltamos ao passo 1.

3. **Passo da hessiana**:Testamos a existência de pontos em $x \in X$ tais que $H(x)$ é semidefinida positiva. Se provarmos que estes pontos não existem então excluímos X e voltamos ao passo 1.

4. **Passo de atualização**: Atualizamos μ. Excluímos caixas em P e C que não podem conter minimizadores globais.

5. **Passo de redução**: Aplicamos o método de Newton intervalar ao sistema $\mathbf{g} = 0$ para reduzir a região de busca. O resultado é a caixa X_1. Se qualquer entrada de X_1 é vazia então excluímos X e voltamos ao passo 1. Se X_1 pode ser salvo como solução então guardamos X_1 em S e voltamos ao passo 1.

6. **Passo da divisão**: Dividimos X_1 em duas caixas X_1' e X_1''. Salvamos estas caixas em P e voltamos ao passo 1.

4.4 Descrição do Algoritmo

Nesta seção detalhamos os passos do algoritmo de otimização irrestrita. Ele é capaz de limitar todos os mínimos globais de f em X_0 por caixas de tamanho pré-fixado. Apresentamos o algoritmo por partes para facilitar seu entendimento. No final da seção mostramos que se x^* é um mínimo global de f em X_0 então nosso algoritmo encontra ao menos uma caixa de tamanho pré-fixado que contém x^*.

4.4.1 Passo de Escolha

Neste passo escolhemos o elemento de P que será processado. Este passo é importante pois afeta a eficiência do algoritmo. Ao contrário do algoritmo para sistemas não lineares, a estrutura do problema nos dá informações que permitem escolher a caixa que será processada.

Se chamarmos de $X_1, X_2, \ldots, X_m$ as caixas de P, avaliamos $\mathbf{f}(X_i)$ para $i = 1, \ldots, m$. Escolhemos então a caixa que satisfaz

$$\min_{i=1\ldots m} \underline{\mathbf{f}(X_i)}.$$

onde $\underline{\mathbf{f}(X_i)}$ é o limitante inferior do intervalo $\mathbf{f}(X_i)$.

Como procuramos por minimizadores globais, é natural esperar que eles pertençam a caixas que satisfazem a condição acima. Esta estratégia não garante que tomaremos elementos de P que contém mínimos globais. Ela apenas prioriza as caixas com potencial para conter estes pontos.

Ao implementar nosso algoritmo de otimização irrestrita, tomamos o cuidado de manter a lista P ordenada segundo os valores de $\underline{\mathbf{f}}$. Assim, a cada iteração tomamos o primeiro elemento de P como caixa a ser processada. Manter a lista P ordenada também nos ajudará a excluir caixas que não podem conter o mínimo global.

4.4.2 Análise do Gradiente

A propriedade de inclusão garante que se $0 \notin \mathfrak{g}(X)$ então $0 \notin \mathfrak{g}(X')$ para qualquer $X' \subseteq X$. Esta propriedade nos dá um teste simples para existência de pontos estacionários de f zeros em X.

- Se $0 \notin \mathfrak{g}(X)$ então não existe $x \in X$ tal que $g(x) = 0$ e excluímos esta caixa.

Lembre que $\mathfrak{g}(X)$ é um vetor intervalar e $0 \notin \mathfrak{g}(X)$ significa que pelo menos uma entrada de $\mathfrak{g}$ não contém zero.

4.4.3 Passo da Hessiana

Uma matriz real A é semidefinida positiva se e somente se $x^T Ax \geq 0$ para qualquer vetor $x \in \mathbb{R}^n$. Segue da definição que uma matriz semidefinida positiva não pode ter entradas negativas em sua diagonal principal. Para entender este fato suponha que a entrada a_{11} de A tem valor negativo. Se tomarmos o vetor $e_1 = (1, 0, \ldots, 0)$ teremos $e_1^T Ae_1 < 0$ independente das demais entradas de A.

Um resultado importante da teoria de otimização diz que se x^* é um ponto de mínimo de f então $H(x^*)$ é uma matriz semidefinida positiva. Este resultado sugere o seguinte teste computacional

- Se $\overline{\mathfrak{H}_{ii}(X)} < 0$ para algum $i = 1, \ldots, n$ então podemos excluir a caixa X.

Onde $\overline{\mathfrak{H}_{ii}(X)}$ é o limitante superior do intervalo $\mathfrak{H}_{ii}(X)$.

Observe que neste teste utilizamos apenas as entradas diagonais de $\mathfrak{H}(X)$. No entanto, em nosso algoritmo calculamos a matriz $\mathfrak{H}(X)$ neste passo. Isto porque usamos esta informação no passo de redução. Como não queremos calcular duas vezes os intervalos da diagonal principal da matriz, avaliamos $\mathfrak{H}(X)$ e guardamos esta informação para usar posteriormente. O próximo passo não exclui X e portanto não corremos o risco de realizar cálculos desnecessários.

4.4.4 Passo de Atualização

Neste passo atualizamos o valor μ. A medida que exploramos X_0 obtemos informações sobre o limitante superior do mínimo global. Usamos estas informações para excluir caixas em P e C que não podem conter mínimos globais.

A atualização de μ é feita através do ponto médio de X.

- Se $\overline{\mathfrak{f}(\bar{x}^I)} < \mu$ então $\mu = \overline{\mathfrak{f}(\bar{x}^I)}$. Onde $\overline{\mathfrak{f}(\bar{x}^I)}$ é o limitante superior de $\mathfrak{f}(\bar{x}^I)$.

A escolha do ponto médio de X é arbitrária. Na prática quando $\bar{x}$ não é representável em M^n tomamos uma aproximação. A idéia do teste é mostrar que existe ao menos um ponto em X que nos permite reduzir o limitante superior μ.

Uma vez que atualizamos μ eliminamos todas as caixas Y em P que satisfazem

$$\underline{f(Y)} > \mu. \tag{4.1}$$

Como mantemos P ordenada, basta encontrar o primeiro elemento que satisfaz esta relação e excluímos todas as caixas deste elemento até o fim da lista.

Ao contrário do algoritmo para sistemas não lineares, não mantemos uma lista de caixas salvas S e sim uma lista de candidatas C. Essa diferença acontece porque neste algoritmo, um elemento em C pode limitar apenas mínimos locais de f. Sendo assim, quando atualizamos μ faz sentido percorrer C e eliminar todos os elementos Y desta lista que satisfazem a relação (4.1).

É possível usar nosso algoritmo para encontrar **todos** os mínimos locais de f. Basta não aplicar o teste proposto acima na lista C. Desta forma, todas as caixas que entram nesta lista permanecem nela até o fim do programa e garantimos limitar todos os mínimos de f em X_0.

4.4.5 Passo de Redução

Neste passo aplicamos o método de Newton intervalar descrito em 3.3 ao sistema $\mathbf{g} = 0$. Nosso objetivo é reduzir a caixa onde buscamos minimizadores.

Inicialmente aplicamos o primeiro estágio do algoritmo de Gauss Seidel intervalar enquanto houver redução no domínio de busca e no máximo por 5 vezes. Isto é, enquanto $X' \neq X$ e não excedermos 5 aplicações.

O cuidado de definir um número máximo de aplicações do primeiro estágio de Gauss Seidel garante que nosso algoritmo deixa este passo em algum momento. Esta escolha também garante que não tenhamos gastos excessivos com o cálculo de caixas que reduzem pouco o domínio de busca. O número 5 é arbitrário e pode ser substituído de acordo com a implementação ou problema a ser resolvido.

Aplicamos então o segundo estágio do algoritmo de Gauss Seidel intervalar. Lembre que este estágio é aplicado apenas uma vez a cada iteração do método de Newton. Se este estágio do método devolver um índice i e dois intervalos disjuntos $\mathbf{x}$ e $\mathbf{y}$ conforme descrito em 3.3 então salve estas informações para usa-las no passo de divisão.

4.4.6 Passo de Divisão

Se uma caixa não pode ser processada ou não foi reduzida o suficiente devemos dividi-la. Vários autores discutiram heurísticas para divisão de caixas. Os trabalhos de Kearfott (1996b), Boyd, Gosh e Magnani (2003) e Pedamallu et al. (2008) são boas referências para o assunto.

Em nosso algoritmo adotamos um esquema de bisseção simples, considerando dois casos.

1. Não temos informações sobre saltos na caixa. Neste casos a dividimos X em X' e X'' a partir da direção com maior comprimento.

2. Sabemos que X veio do passo de redução e nele salvamos o índice de maior salto durante a aplicação do segundo estágio de Gauss Seidel. Neste caso dividimos X em X' e X'' nessa direção. Se i é o índice dessa direção e $\mathbf{x}$ e $\mathbf{y}$ são os intervalos resultantes então tomamos $X_i' = \mathbf{x}$ e $X_i'' = \mathbf{y}$.

Uma vez que X' e X'' foram obtidas, devemos guarda-las em L. Neste ponto avaliamos $\underline{f}(X')$ e $\underline{f}(X'')$. Guardamos estas caixas em L e ordenamos a lista de acordo com os valores de $\underline{f}$.

4.4.7 Tomando Uma Caixa Como Candidata

A caixa $X \subseteq X_0$ é candidata à solução e a guardamos em C se $W(X) < \varepsilon_X$ e $||\mathfrak{g}(X)|| < \varepsilon_F$. Os valores positivos ε_X e ε_F são dados pelo usuário e não devem ser menor do que a unidade de arredondamento definida por M.

Estas condições controlam o comprimento máximo de X e a maior distância permitida de $\mathfrak{g}(X)$ ao vetor nulo. Podemos omitir uma destas condições tomando ε_X ou ε_F como infinito.

Tomar $\varepsilon_X = \infty$ aumenta a eficiência do algoritmo nos casos em que sabemos que f tem muitos mínimos locais em X_0. Não recomendamos tomar valores altos de ε_F pois isto introduz ruídos nos resultados. Outros critérios para aceitar X como solução são discutidos por Hansen e Walster (2004), Hargreaves (2002) e Ratschek e Rokne (1988).

Ao contrário do que fizemos no capítulo anterior, apenas nomeamos as caixas como candidatas à solução. Isto porque podemos excluir caixas em C durante o processo se descobrirmos que ela limita apenas pontos de mínimo local de f. O conjunto solução S é dado pelas caixas que permaneceram em C até fim do programa. Este conjunto pode ter caixas que não contém soluções além de caixas que limitam mais de uma vez a mesma solução.

4.4.8 O Algoritmo

Apresentamos agora o algoritmo para resolver problemas de otimização irrestrita. Ele deve ser seguido em seqüência exceto quando indicamos o contrário. Omitimos os passos em que testamos se o número máximo de iterações ou avaliações de função foi atingido. Não é difícil incorporar estes passos ao algoritmo.

Entradas

- A extensão $(M, \mathfrak{f}, \mathfrak{J})$
- A caixa inicial representável X_0.
- Um limitante superior do mínimo global μ. Pode ser ∞.

- Valores ε_X e ε_F compatíveis com M. Veja a subseção anterior.

Saídas

- Um conjunto de caixas $S := \{S_1, \ldots, S_k\}$ tais que se x^* é mínimo global de f então x^* pertence a pelo menos um elemento de S.
- Um limitante superior do mínimo global μ.

Não há relação de $1-1$ entre os mínimos de f e os elementos de S. É possível que uma mesma solução esteja em mais de uma caixa e também é possível que S tenha caixas que não contém soluções.

Passos do Algoritmo

1. Se P está vazia então encerre o programa. Caso contrário tome a primeira caixa de P e chame-a de X. Exclua X de P.
2. Avalie $\mathfrak{f}(X)$ e $\mathfrak{g}(X)$.
3. Se alguma entrada de $\mathfrak{g}(X)$ não contém zero então exclua X e volte ao passo 1.
4. Avalie $\mathfrak{H}(X)$.
5. Se $\overline{\mathfrak{H}_{ii}(X)} < 0$ para algum $i = 1, \ldots, n$ então exclua X e volte ao passo 1.
6. Estime $\bar{x} = mid(X)$ e $\mathfrak{f}(\bar{x}^I)$.
7. Se $\overline{\mathfrak{f}(\bar{x}^I)} < \mu$ então $\mu = \overline{\mathfrak{f}(\bar{x}^I)}$.
8. Se atualizamos μ então exclua todas as caixas Y em P e C tais que $\underline{\mathfrak{f}(Y)} > \mu$.
9. Se $W(X) < \varepsilon_X$ e $||\mathfrak{g}(X)|| < \varepsilon_F$ então guarde X em C e volte ao passo 1.
10. Estime $A = mid(\mathfrak{H}(X))$.
11. Estime os valores singulares A usando aritmética de ponto flutuante. Chame de σ_1 o menor destes valores e σ_n o maior.
12. Se $\frac{\sigma_n}{\sigma_1} < \nu$ então divida X na direção de maior comprimento. Guarde as duas caixas em P. Ordene P de acordo com $\underline{\mathfrak{f}}$. Em nossos experimentos usamos $\nu = 10^{-6}$.
13. Estime $M = A^{-1}\mathfrak{H}(X)$ e $b = -A^{-1}\mathfrak{f}(\bar{x}^I)$.
14. Aplique o estágio não estendido de Gauss Seidel ao sistema $M(z - \bar{x}) = b$ tomando X como estimativa inicial. O resultado é uma nova caixa X_1.

15. Se alguma entrada de X_1 é vazia então exclua X e volte ao passo 1.

16. Avalie $\mathfrak{f}(X_1)$ e $\mathfrak{g}(X_1)$.

17. Se alguma entrada de $\mathfrak{g}(X)$ não contém zero então exclua X e volte ao passo 1.

18. Se $\underline{\mathfrak{f}(X_1)} > \mu$ então exclua X e volte ao passo 1.

19. estime $\bar{x} = mid(X_1)$ e avalie $\mathfrak{f}(\bar{x}^I)$.

20. Se $\overline{\mathfrak{f}(\bar{x}^I)} < \mu$ então $\mu = \overline{\mathfrak{f}(\bar{x}^I)}$.

21. Se atualizou μ então exclua todas as caixas Y em P e C tais que $\underline{\mathfrak{f}(Y)} > \mu$.

22. Se $W(X_1) < \varepsilon_X$ e $||\mathfrak{g}(X_1)|| < \varepsilon_F$ então guarde X_1 em C e volte ao passo 1.

23. Tome $b = -A^{-1}\mathfrak{f}(\bar{x}^I)$.

24. Se $X_1 \neq X$ e aplicamos o passo 13 menos do que 6 vezes então substitua $X = X_1$. Volte ao passo 13.

25. Aplique o estágio estendido de Gauss Seidel ao sistema $M(z - \bar{x}) = b$ tomando X_1 como estimativa inicial. O resultado é uma nova caixa X_2. Salve a direção em que ocorre o maior salto e os intervalos resultantes nessa direção.

26. Se algum elemento de X_2 é vazio então exclua X e volte ao passo 1.

27. Avalie $\mathfrak{f}(X_2)$ e $\mathfrak{g}(X_2)$.

28. Se algum elemento de $\mathfrak{g}(X_2)$ não contém zero então exclua X e volte ao passo 1.

29. Se $\underline{\mathfrak{f}(X_2)} > \mu$ então exclua X e volte ao passo 1.

30. Avalie $\bar{x} = mid(X_2)$ e $\mathfrak{f}(\bar{x}^I)$.

31. Se $\overline{\mathfrak{f}(\bar{x}^I)} < \mu$ então $\mu = \overline{\mathfrak{f}(\bar{x}^I)}$.

32. Se atualizou μ então exclua todas as caixas Y em P e C tais que $\underline{\mathfrak{f}(Y)} > \mu$.

33. Se $W(X_2) < \varepsilon_X$ e $||\mathfrak{g}(X_2)|| < \varepsilon_F$ então guarde X_2 em C e volte ao passo 1.

34. Se nenhum salto foi salvo então divida X_2 na direção de maior comprimento. Guarde as duas caixas em P. Ordene P de acordo com $\underline{\mathfrak{f}}$. Volte ao passo 1.

35. Se algum salto foi salvo então divida X_2 na direção deste salto. Utilize os intervalos resultantes nessa direção. Ordene P de acordo com $\underline{\mathfrak{f}}$. Volte ao passo 1.

4.4.9 Convergência do Algoritmo

Concluímos esta seção mostrando que o algoritmo acima limita todos os mínimos globais de f em X_0 desde que o número máximo de iterações ou avaliação de funções não seja atingido.

Teorema 1 *Sejam $f : \mathbb{R}^n \to \mathbb{R}$ uma função diferenciável, $(M, \mathtt{f}, \mathtt{g}, \mathtt{H})$ uma extensão e X_0 uma caixa representável em M. Suponha que ε_X e ε_F são números positivos consistentes com M e μ um limitante superior do mínimo global de f. Então o algoritmo acima limita todas os mínimos globais de f em X_0, que satisfazem $g(x) = 0$, por caixas $X_i \subseteq X_0$ tais que $W(X_i) < \varepsilon_X$ e $||\mathtt{g}(X_i)|| < \varepsilon_F$ em um número finito de iterações.*

Demonstração: Como $\mathtt{f}$, $\mathtt{g}$ e $\mathtt{H}$ são extensões intervalares de f e g e H garantimos que todos os cálculos envolvendo estas funções são verificados. Desta forma não perdemos soluções por erros de arredondamento.

Os argumentos usados para justificar que os passos de redução, eliminação e divisão não perdem soluções no capítulo anterior continuam válidos e através deles temos que os passos de redução, divisão, gradiente e Hessiana não devem perder soluções. O passo de atualização não exclui mínimos globais de f pois ele só se aplica quando encontramos um ponto que reduz verificadamente o limitante superior μ.

Os passos de divisão e redução garantem que uma caixa que não foi excluída será reduzida. Como ε_X e ε_F são consistentes com M garantimos que uma caixa que não foi excluída deverá ser aceita como solução em um número finito de iterações. Isto conclui a demonstração.□

4.5 A Função intminunc

A função *intminunc* do nosso pacote *INTSOLVER* implementa o algoritmo descrito na seção anterior. Nesta seção ilustramos seu uso e algumas de suas opções.

O *INTSOLVER* deve ser simples para usuários acostumados ao Matlab. A função *intminunc* exige poucos parâmetros de entrada. Por exemplo, desejamos encontrar os mínimos globais de

$$(x_1^2 + x_2^2 - 1)^2 + (x_1^2 - x_2)^2$$

na caixa $-10 \leq x_1, x_2 \leq 10$. Para realizar esta tarefa com a função *intminunc* precisamos apenas da função $\mathtt{f}$ e da caixa X_0.

```
>> f  = @(x) (x(1)^2+ x(2)^2 - 1)^2 +  (x(1)^2 - x(2))^2
>> x0 = [infsup(-10,10), infsup(-10,10)];
>> [x, iv]  = intminunc(f, x0)
```

```
  ite   evf     evg    evi   evh   min          nsol   nlist   completed(%)

   0     1      0      0     0     Inf          0      1       0.000

  10     71     50     10    10    1.000e+00    0      5       0.625

  20     141    100    20    20    1.000e+00    0      5       0.938

  30     211    150    30    30    1.028e-01    0      6       0.986

  40     287    206    40    40    1.028e-01    0      13      0.991
intval x =
 [-0.7862,-0.7861] [0.6180, 0.6181]
 [0.7861, 0.7862] [0.6180, 0.6181]

intval iv =
 1.0e-017 *
 [0.0000,0.2731]
 [0.0000,0.2731]
```

A função *intminunc* assume os seguintes padrões,

- Número máximo de iterações: 1000.
- Número máximo de Avaliações de função: 10000.
- $\mu = \infty$.
- $\varepsilon_X = 10^{-8}$.
- $\varepsilon_F = 10^{-8}$.
- Nível do Display: 10.
- Iterar até a limitar todas as soluções: Não.

Todas os parâmetros acima podem ser modificados. Em particular quando optamos por iterar até limitar todas as soluções, a função ignora quaisquer valores em número máximo de iterações e número máximo de avaliações de função. Obviamente esta escolha deve aumentar o tempo de execução do algoritmo e cabe ao usuário o uso coerente desta opção.

No próximo exemplo mudamos as quantidades ε_X e ε_X para 10^{-10}. Também atribuímos o valor 0 ao nível do display de forma que as informações sobre o processo não serão exibidas.

```
>> opt = intoptimset();
>> opt(1) = 10^-10;
>> opt(2) = 10^-10;
>> opt(8) = 0;

>> [x, iv, ef] = intminunc(f, x0, opt)

intval x =
[   -0.7862,   -0.7861] [    0.6180,    0.6181]
[    0.7861,    0.7862] [    0.6180,    0.6181]

intval iv =
  1.0e-023 *
[    0.0000,    0.1567]
[    0.0000,    0.1567]

ef =
     0
```

A saída x mostra as caixas soluções encontradas pela função. Em iv temos $f(X_i)$ onde X_1 e X_2 são as caixas em x. Finalmente o parâmetro de saída $ef = 0$ nos diz que a função foi finalizada com sucesso.

As funções do *INTLAB* estão documentadas e o usuário que desejar maiores informações deve digitar

```
>> doc nome_da_função
```

no Matlab.

4.6 Estudos Computacionais

Para avaliar o desempenho da função *intminunc* escolhemos 10 problemas irrestritos da literatura de otimização global. Levamos em conta a precisão e a eficiência da função. Com relação à precisão veremos que a função é capaz de encontrar todas as soluções dos problemas desde que o número máximo de iterações ou avaliações de função não seja atingido. Com relação à eficiência, apresentamos em cada caso o número de iterações, avaliações de função, gradiente, Hessiana e o tempo de execução em segundos da função.

A menos de menção contrária, o número máximo de iterações em cada teste é 1000, o número máximo de avaliações de função é 10000, $\mu = +\infty$ e os valores ε_X e ε_F são iguais a 10^{-8}. As funções

foram implementadas em Matlab exatamente como aparecem descritas abaixo. A extensão é feita apenas trocando variáveis reais por intervalos. Os testes foram realizados em um processador AMD Sempron de $800MHZ$ e $1GB$ de memória RAM com sistema operacional Linux.

4.6.1 Rosenbrock

Esta é uma generalização da função de Rosenbrock. O objetivo é encontrar todos os mínimos globais da função

$$\sum_{i=1}^{n-1}[100(x_i - x_{i+1})^2 + (x_i - 1)^2]$$

na caixa $-5 \leq x_1, \ldots, x_n \leq 10$.

Este problema tem uma única solução dada por

$$x^* = (1, \ldots, 1)$$

e sabemos que $f(x^*) = 0$. A figura abaixo mostra o gráfico da função no caso em que $n = 2$.

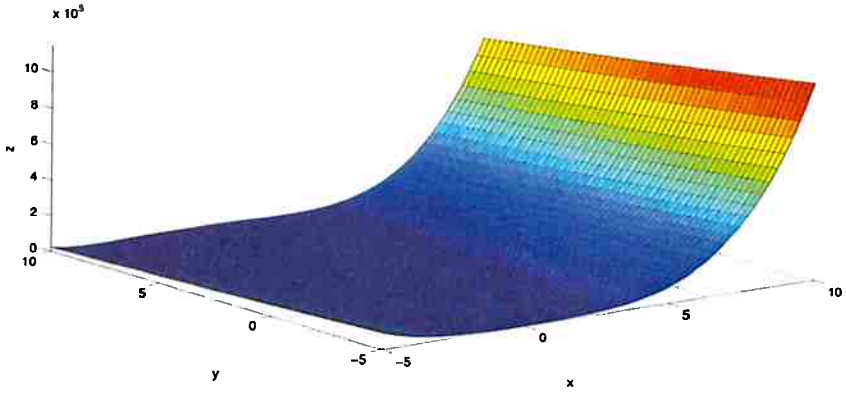

Figura 4.1: Função de Rosenbrock no domínio $-5 \leq x_1, x_2 \leq 10$

Aplicando a função *intminunc* obtemos

#Variáveis	#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
2	69	507	377	69
5	171	1424	1090	171
10	275	2830	2288	275
15	375	3982	3240	375
20	475	5042	4100	475

#Variáveis	Tempo(s)	μ	#Soluções	Concluído(%)
2	10.185	4.457e-29	1	100
5	90.488	1.932e-22	1	100
10	384.86	4.774e-22	1	100
15	892.81	4.579e-22	1	100
20	1444.5	4.579e-22	1	100

Em todos os casos encontramos caixas que contém o ponto.

$$x^* = (1, \ldots, 1).$$

A tabela abaixo traz estatísticas sobre estas caixas

#Variáveis	$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
2	5.6047e-11	9.7256e-14	$[0, 1.959e-24]$
5	6.2655e-9	1.3630e-11	$[0, 2.165e-20]$
10	6.7234e-9	1.3272e-11	$[0, 2.477e-20]$
15	6.7110e-9	1.3126e-11	$[0, 2.464e-20]$
20	6.7110e-9	1.3125e-11	$[0, 2.464e-20]$

4.6.2 Rastringin

Este problema foi extraído de Torn e Zilinskas (1989). A função de Rastringin é dada por

$$20 + x_1^2 - 10\cos(2\pi x_1) + x_2^2 - 10\cos(2\pi x_2)$$

na caixa $-5.12 \leq x_1, x_2 \leq 5.12$.

Este problema tem muitos mínimos locais e somente um global dado por

$$x^* = (0, 0)$$

e sabemos que $f(x^*) = 0$. A figura abaixo mostra o gráfico da função.

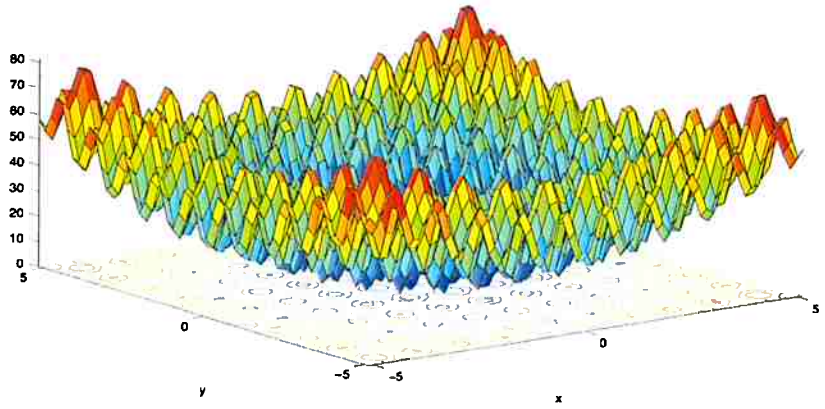

Figura 4.2: Função de Rastringin no domínio $-5.12 \leq x_1, x_2 \leq 5.12$

Aplicando nosso método obtemos

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
31	316	265	31

Tempo(s)	μ	#Soluções	Concluído(%)
10.926	3.553e-15	4	100

Encontramos mais de um limitante para o único mínimo global x^*. Isto se deve ao passo de divisão do algoritmo, veja a subseção 3.9.2 para maiores detalhes. Todas as caixas soluções deste exemplo limitam x^*. A tabela abaixo traz estatísticas elas

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
7.180e-23	1.809e-24	$10^{-14}[-0.7106, 0.7106]$
7.180e-23	1.809e-24	$10^{-14}[-0.7106, 0.7106]$
7.180e-23	1.809e-24	$10^{-14}[-0.3553, 0.7106]$
7.180e-23	1.809e-24	$10^{-14}[-0.3553, 0.7106]$

4.6.3 Levy

Este problema foi extraído de Levy et al. (1981). Desejamos encontrar todos mínimos globais de

$$\sin(\pi z_1)^2 + \sum_{i=1}^{n-1}(z_i - 1)^2(1 + 10(\sin(\pi z_i + 1))^2) + (z_n - 1)^2(1 + (\sin(2\pi z_n))^2)$$

com $z_i = 1 + \frac{x_i - 1}{4}$ na caixa $-10 \leq x_1, \ldots, x_n \leq 10$.

Este problema tem uma única solução dada por

$$x^* = (1, \ldots, 1)$$

e sabemos que $f(x^*) = 0$. A figura abaixo mostra o gráfico da função no caso em que $n = 2$.

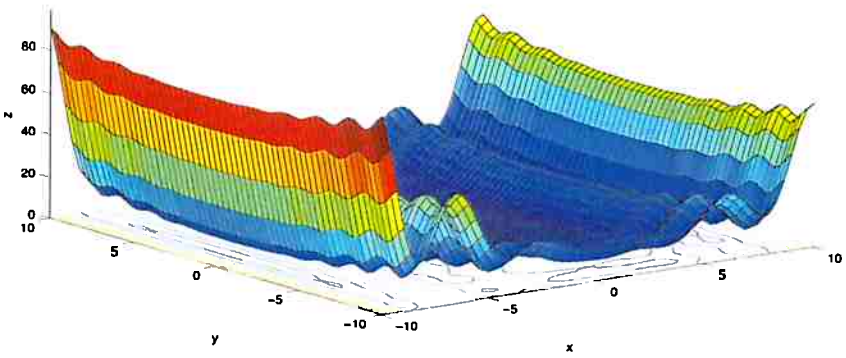

Figura 4.3: Função de Levy no domínio $-10 \leq x_1, x_2 \leq 10$

Utilizando a função *intminunc* do *Intsolver* obtemos

#Variáveis	#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
2	11	146	126	11
5	29	374	318	29
10	59	754	638	59
15	89	1134	958	89
20	119	1514	1278	119

#Variáveis	Tempo(s)	μ	#Soluções	Concluído(%)
2	9.5531	6.829e-19	1	100
5	48.231	6.829e-19	1	100
10	187.00	6.829e-19	1	100
15	415.58	6.829e-19	1	100
20	733.59	6.829e-19	1	100

Em todos os casos encontramos caixas que contém o ponto

$$x^* = (1, \ldots, 1).$$

A tabela abaixo traz estatísticas sobre estas caixas

#Variáveis	$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
2	4.9032e-9	2.8101e-9	$[0, 5.358e-18]$
5	4.9032e-9	2.8101e-9	$[0, 5.358e-18]$
10	4.9032e-9	2.8101e-9	$[0, 5.358e-18]$
15	4.9032e-9	2.8101e-9	$[0, 5.358e-18]$
20	4.9032e-9	2.8101e-9	$[0, 5.358e-18]$

4.6.4 Griewank

Este problema foi extraído de Torn e Zilinskas (1989). Desejamos encontrar todos mínimos globais de

$$\sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} \frac{\cos(x_i}{\sqrt{(i)}} + 1$$

na caixa $-600 \leq x_1, x_2 \leq 600$.

Este problema tem uma única solução dada por

$$x^* = (0, 0)$$

e sabemos que $f(x^*) = 0$. A figura abaixo mostra o gráfico da função.

Com a função *intminunc* obtemos

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
79	554	407	79

Tempo(s)	μ	#Soluções	Concluído(%)
18.454	2.220e-016	4	100

Encontramos mais de um limitante para o único mínimo global x^*. Isto se deve ao passo de divisão do algoritmo, veja a subseção 3.9.2 para maiores detalhes. Todas as caixas soluções deste exemplo limitam x^*. A tabela abaixo traz estatísticas sobre estas caixas

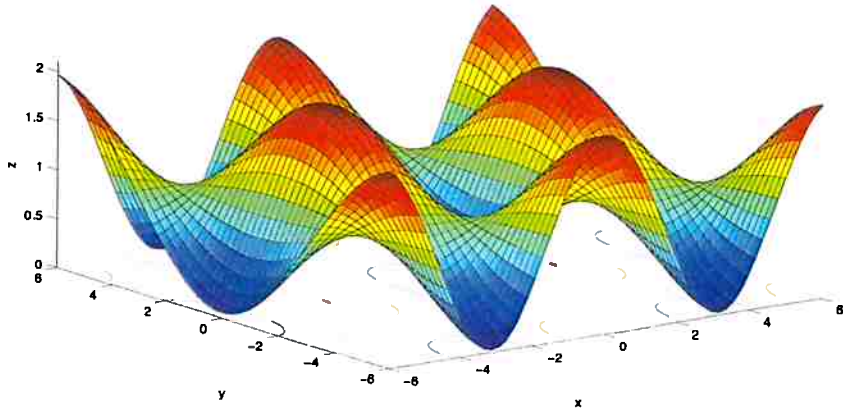

Figura 4.4: Função de Griewank no domínio $-6 \leq x_1, x_2 \leq 6$

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
4.2361e-16	5.986e-16	$[-6.662e-16, 3.331e-16]$
4.2361e-16	5.986e-16	$[-2.221e-16, 3.331e-16]$
4.2361e-16	5.986e-16	$[-2.221e-16, 3.331e-16]$
4.2361e-16	5.986e-16	$[0, 3.331e-16]$

4.6.5 Goldstein e Price

Este problema foi extraído de Floudas et al. (1999). Desejamos encontrar todos mínimos globais de

$$[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] * \\ [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$

na caixa $-2 \leq x_1, x_2 \leq 2$.

Este problema tem uma única solução dada por

$$x^* = (0, -1)$$

e sabemos que $f(x^*) = 3$. A figura abaixo mostra o gráfico da função.

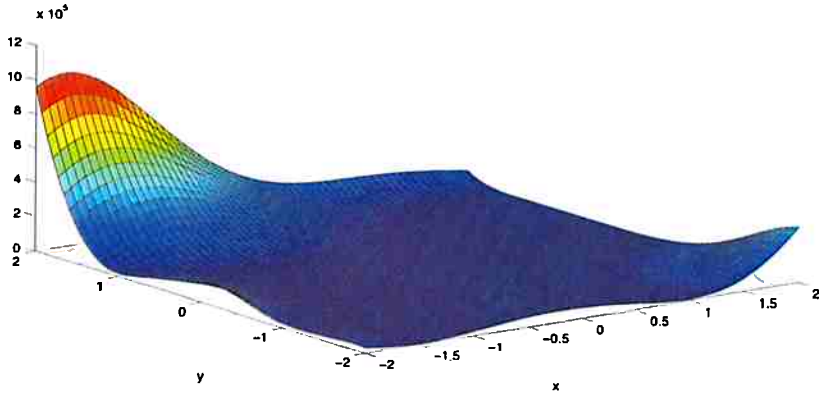

Figura 4.5: Função de Goldstein e Price no domínio $-2 \leq x_1, x_2 \leq 2$

Resolvemos este problema com a opção de iterar até a convergência. Desta forma obtemos

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
14475	63272	54520	63272

Tempo(s)	μ	#Soluções	Concluído(%)
18985	3	4	100

Encontramos mais de um limitante para o único mínimo global x^*. Isto se deve ao passo de divisão do algoritmo, veja a subseção 3.9.2 para maiores detalhes. Todas as caixas soluções deste exemplo limitam x^*. A tabela abaixo traz estatísticas sobre elas

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
9.5891e-9	4.379e-12	[2.9999, 3.0001]
9.7680e-9	4.040e-12	[2.9999, 3.0001]
9.2497e-9	3.858e-12	[2.9999, 3.0001]
5.3886e-9	2.455e-12	[2.9999, 3.0001]

4.6.6 Three Hump Camel

Este problema foi extraído de Floudas et al. (1999). Desejamos encontrar todos mínimos globais de

$$4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

na caixa $-5 \leq x_1, x_2 \leq 5$.

Este problema tem como soluções os vetores

$$x_1^* = (0.0898, -0.7126)$$
$$x_2^* = (-0.0898, 0.7126)$$

e sabemos que $f(x^*) = -1.03163$. A figura abaixo mostra o gráfico da função.

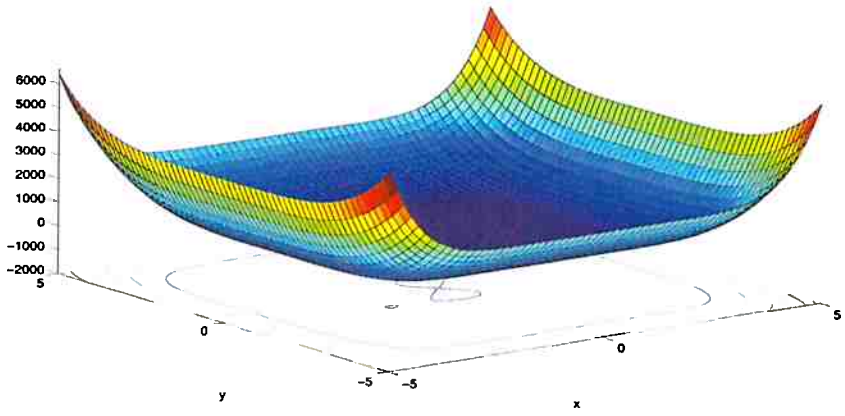

Figura 4.6: Função Three Hump Camel no domínio $-5 \leq x_1, x_2 \leq 5$

Nosso método nos dá

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
229	1162	1009	229

Tempo(s)	μ	#Soluções	Concluído(%)
51.3149	-1.0316	2	100

As caixas soluções limitam corretamente os pontos x_1^* e x_2^*. A tabela abaixo traz estatísticas sobre elas

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
1.7553e-9	2.406e-10	$[-1.0317, -1.0316]$
1.7553e-9	2.406e-10	$[-1.0317, -1.0316]$

4.6.7 Branin

Este problema foi extraído de Ratz e Czendes (1995). Desejamos encontrar todos mínimos globais de

$$(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5x_1}{\pi} - 6)^2 + 10(1 - \frac{1}{8\pi})cos(x_1) + 10$$

na caixa $-5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$.

Este problema tem as seguintes soluções

$$x_1^* = (-\pi, 12.275)$$
$$x_2^* = (\pi, 2.275)$$
$$x_3^* = (9.42478, 2.475)$$

e sabemos que $f(x^*) = 0.397887$. A figura abaixo mostra o gráfico da função.

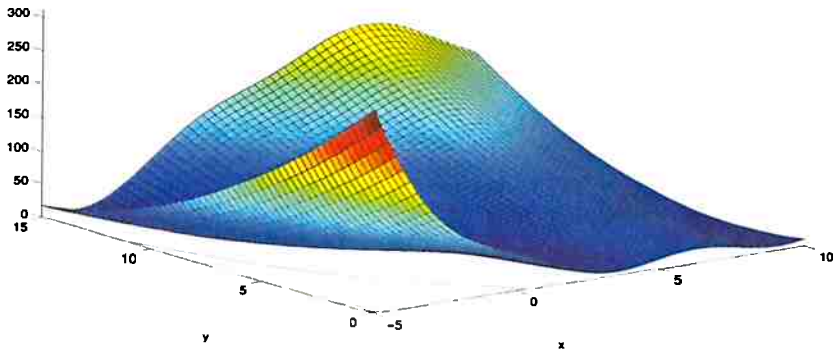

Figura 4.7: Função de Branin no domínio $-5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$

Através da função *intminunc* obtemos

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
24	251	217	24

Tempo(s)	μ	#Soluções	Concluído(%)
7.0609	0.3979	3	100

As caixas solução limitam corretamente os pontos reportados acima. Segue uma tabela com estatísticas sobre estas caixas

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
2.1889e-9	6.12e-10	[0.3978, 0.3979]
4.6852e-9	1.232e-9	[0.3978, 0.3979]
5.8498e-9	7.54e-10	[0.3978, 0.3979]

4.6.8 Easom

Este problema foi extraído de Easom (1990). Desejamos encontrar todos mínimos globais de

$$-cos(x_1)cos(x_2)e^{-((x_1-\pi)^2+(x_2-\pi)^2)}$$

na caixa $-30 \leq x_1, x_2 \leq 30$.

Este problema tem uma única solução

$$x_1^* = (\pi, \pi)$$

e sabemos que $f(x^*) = -1$. A figura abaixo mostra o gráfico da função.

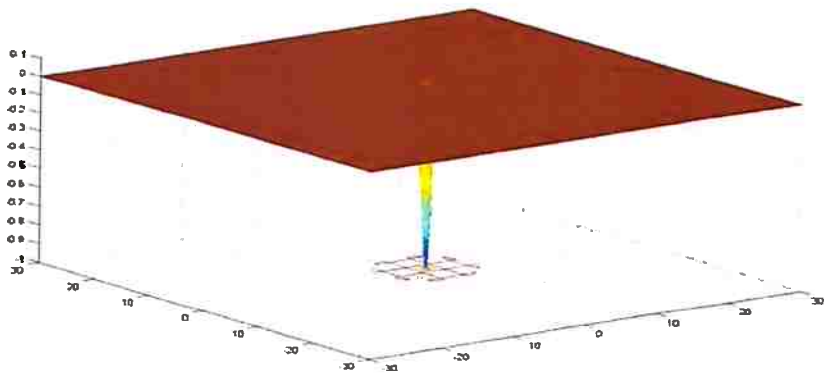

Figura 4.8: Função de Easom no domínio $-30 \leq x_1, \leq x_2 \leq 30$

Aplicando nossa função obtemos

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
19	102	78	19

Tempo(s)	μ	#Soluções	Concluído(%)
4.4524	-1.000	1	100

A caixa obtida limita corretamente a única solução do Problema. Seguem estatísticas sobre ela

$\|\|\mathfrak{g}(X)\|\|$	$\mathbb{W}(X)$	$\mathfrak{f}(X)$
3.1227e-9	1.2280e-9	$[-1.0000, -0.9999]$

4.6.9 Shubert

Este problema foi extraído de Michalewicz (1996). Desejamos encontrar todos mínimos globais de

$$\sum_{i=1}^{5} i\cos((i+1)x_1 + i) * \sum_{i=1}^{5} i\cos((i+1)x_2 + i)$$

na caixa $-10 \leq x_1, x_2 \leq 10$.

Este problema tem muitas soluções locais e 18 soluções globais. Nas soluções globais sabemos que $f(x^*) = -186.73067$. A figura abaixo mostra o gráfico da função.

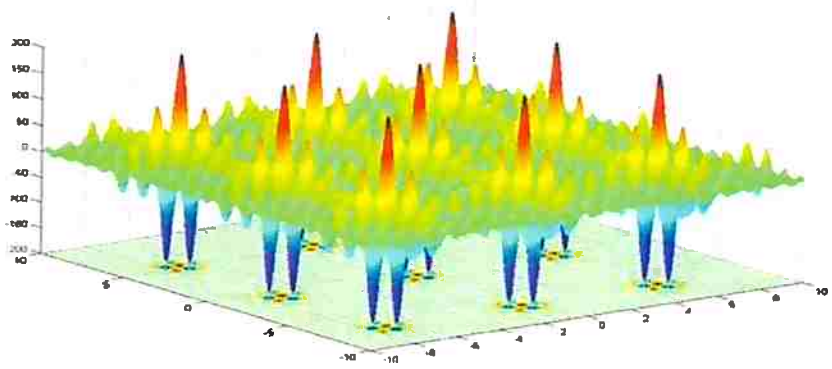

Figura 4.9: Função de Shubert no domínio $-10 \leq x_1, x_2 \leq 10$

Aplicando nossa função obtemos

#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
834	6015	4630	833

Tempo(s)	μ	#Soluções	Concluído(%)
968.606	-186.7309	18	100

As caixas obtidas limitam corretamente as soluções do Problema. Seguem estatísticas delas

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
9.254e-9	3.868e-11	$[-186.7310, -186.7309]$
9.307e-9	3.801e-11	$[-186.7310, -186.7309]$
9.322e-9	3.341e-11	$[-186.7310, -186.7309]$
8.260e-9	3.439e-11	$[-186.7310, -186.7309]$
9.527e-9	3.304e-11	$[-186.7310, -186.7309]$
8.575e-9	3.812e-11	$[-186.7310, -186.7309]$
2.856e-9	1.100e-11	$[-186.7310, -186.7309]$
3.976e-9	1.522e-11	$[-186.7310, -186.7309]$
2.658e-9	1.009e-11	$[-186.7310, -186.7309]$
1.693e-9	6.430e-12	$[-186.7310, -186.7309]$
1.010e-9	4.040e-12	$[-186.7310, -186.7309]$
6.63e-10	2.32e-12	$[-186.7310, -186.7309]$
6.22e-10	2.50e-12	$[-186.7310, -186.7309]$
4.56e-10	1.75e-12	$[-186.7310, -186.7309]$
2.36e-10	9.5e-13	$[-186.7310, -186.7309]$
1.02e-10	4.4e-13	$[-186.7310, -186.7309]$
1.44e-10	5.4e-13	$[-186.7310, -186.7309]$
5.7e-11	2.3e-13	$[-186.7310, -186.7309]$

4.6.10 Michalewics

Este problema foi extraído de Michalewicz (1996). Desejamos encontrar todos mínimos globais de

$$-\sum_{i=1}^{n} \sin(x_i)(\sin(\frac{2ix_i^2}{\pi}))^{20}$$

na caixa $0 \leq x_1, \ldots x_n \leq \pi$.

Este problema tem muitas soluções locais e apenas uma solução global.

$$x^* = \begin{cases} -1.8013 \text{ se } n = 2 \\ -4.687658 \text{ se } n = 5 \\ -9.66015 \text{ se } n = 10 \end{cases}$$

A figura abaixo mostra o gráfico da função.

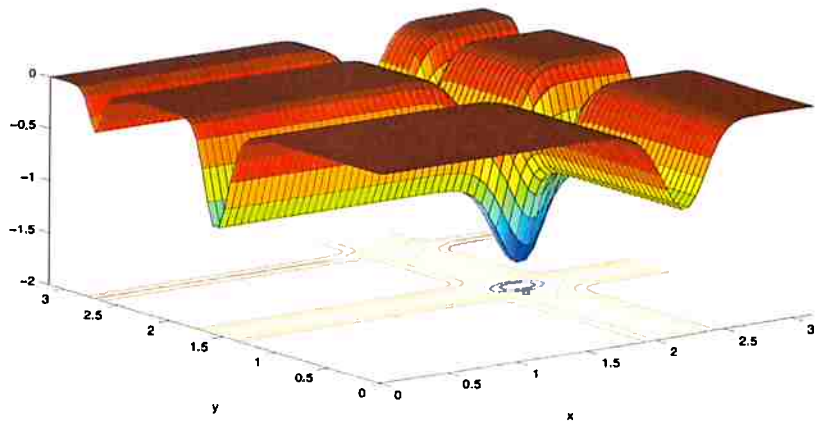

Figura 4.10: Função de Michalewicz no domínio $0 \leq x_1, x_2 \leq \pi$

Aplicando nossa função obtemos

#Variáveis	#Iterações	#Aval. Func.	#Aval. Grad.	#Aval. Hess.
2	18	166	138	18
5	108	1051	870	108
10	1001	9840	7890	1000

#Variáveis	Tempo(s)	μ	#Soluções	Concluído(%)
2	11.059	-1.8015	1	100
5	168.28	-4.688	1	100
10	418.55	-8.036	0	68

Nosso método não é capaz de resolver o problema de Michalewicz com $n = 10$ em até 1000 iterações. Nos demais casos o programa limita corretamente os mínimos globais de f. Seguem estatísticas das caixas soluções

Para $n = 2$

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
8.9393e-09	4.084e-10	[-1.8014,-1.8013]

Para $n = 5$

$\|\|g(X)\|\|$	$\mathbb{W}(X)$	$f(X)$
2.9457e-09	5.276e-11	[-4.6877,-4.6876]

Capítulo 5

Otimização Com Restrições de Igualdade

5.1 Introdução

Sejam $f : \mathbb{R}^n \to \mathbb{R}$ e $c : \mathbb{R}^n \to \mathbb{R}^m$ funções duas vezes diferenciáveis com $m < n$. Neste capítulo estudamos o problema de limitar **todas** as soluções de

$$\min f(x) \tag{5.1}$$
$$\text{Sujeito a } c(x) = 0 \tag{5.2}$$
$$x \in X_0 \tag{5.3}$$

onde X_0 é uma caixa da forma $l \leq x \leq u$.

Assim como nos casos anteriores o problema pode ser tratado por métodos de convexificação e aritmética de ponto flutuante, no entanto, acreditamos que a abordagem intervalar é mais natural e rigorosa. Ainda seguindo a filosofia dos capítulos anteriores, adaptamos o método de Newton intervalar para resolver o problema. Nossos objetivos são:

1. Estudar um algoritmo capaz de limitar, sob determinadas hipóteses, todos os mínimos globais de f em X_0 restritos a c.
2. Ilustrar o uso da função *intmincon* do *INTSOLVER* que implementa o algoritmo desenvolvido ao longo do capítulo.
3. Validar nossa função através de estudos computacionais.

Destacamos que algumas hipóteses devem valer para que nosso algoritmo seja capaz de limitar todas soluções do problema. Veremos na seção 5.3 quais são essas condições mas adiantamos ao leitor acostumado com a teoria de otimização que x^* deve satisfazer alguma qualificação de restrição. Ao leitor com pouco conhecimento de otimização é necessário dizer que a teoria para

resolver problemas restritos é diferente daquela para problemas irrestritos. Em nosso trabalho usamos resultados dados por Lagrange no século XVIII para resolver o problema. Estas idéias serão resumidas neste capítulo.

Dividimos o texto da seguinte forma. Na seção 5.2 apresentamos a notação no capítulo. A seção 5.3 resume as condições de Lagrange e não precisa ser lida por leitores familiarizados com otimização. Em 5.4 apresentamos a visão geral de nosso algoritmo e na seção 5.5 o descrevemos em detalhes. A seção 5.6 ilustra a função *intmincon* do *INTSOLVER* que implementa nosso método. Em 5.7 avaliamos a precisão e eficiência da função *intmincon* através de testes computacionais.

As seções 5.2 - 5.5 serão úteis ao leitor que desejar implementar nosso algoritmo. Os interessados no uso do mesmo podem ir direto à seção 5.6.

5.2 Notação

Denotamos por g e H o vetor gradiente e a matriz hessiana de f. O vetor gradiente e matriz hessiana das restrições c_i são dados por J_i e D_i. Pontos reais serão denotados por letras minúsculas, por exemplo x ao passo que intervalos são representados por letras minúsculas em negrito, por exemplo $\mathbf{x}$. Escrevemos x^I nos casos em que temos um ponto real e desejamos representá-lo como intervalo. Por exemplo, entendemos que 0.1^I é o intervalo

```
>> intval(0.1);
intval ans =
[0.0999,0.1001]
```

dado pelo *INTLAB*. Denotamos caixas por letras latinas maíusculas, por exemplo X e Y.

O conjunto $(M, \mathfrak{f}, \mathfrak{g}, \mathfrak{H}, \mathfrak{c}, \mathfrak{J}, \mathfrak{D})$ será chamado de extensão. Em uma extensão, M é o conjunto de números representáveis em nosso computador, as funções $\mathfrak{f}$, $\mathfrak{g}$ e $\mathfrak{H}$ são extensões intervalares de f, g e H e as funções $\mathfrak{c}$, $\mathfrak{J}$ e $\mathfrak{D}$ são extensões intervalares de c, J e D. Ao longo do capítulo assumimos que a caixa X_0 é representável o que quer dizer que todos os vértices de X_0 pertencem ao conjunto $\mathbb{M}^n$.

5.3 Condições de Lagrange

Esta seção discute as condições de Lagrange. Elas são necessárias para que o ponto x^* seja candidato a mínimo global. O algoritmo desenvolvido nas próximas seções depende fortemente dos resultados desta seção. Nossa exposição é elementar e o para uma exposição completa do assunto o leitor deve consultar Izmailov e Solodov (2005) e Luenberger (1989).

Começamos por definir um tópico importante da teoria de otimização, a qualificação das restrições. O teorema de lagrange, que apresentamos nesta seção, supõe que um candidato a mínimo

x^* deve satisfazer alguma dessas qualificações. A mais famosa das qualificações de restrição é a de regularidade

Definição 1 *Seja x^* um ponto que satisfaz $c(x^*) = 0$. Diremos que x^* é ponto regular das restrições se e somente se o conjunto de vetores $\{J_1(x^*), J_2(x^*), \ldots, J_m(x^*)\}$ é linearmente independente.*

Para facilitar a notação, chamamos os pontos regulares das restrições de pontos regulares. Em geral, a não validade desta qualificação está ligada a problemas mal formulados ou instáveis e nosso algoritmo pode falhar ao tratar estes casos. Antes discutirmos as condições Lagrange precisamos ainda da idéia de função Lagrangeana

Definição 2 *Sejam f e c definidos como em 5.2. A função Lagrangeana $L : \mathbb{R}^{n+m} \to \mathbb{R}^n$ é dada por*

$$L(x, \lambda) = g(x) - \sum_{i=1}^{m} \lambda_i J_i(x). \tag{5.4}$$

Os valores $\lambda_1, \ldots \lambda_m$ são conhecidos como multiplicadores de Lagrange. É claro que se f e c são extensões intervalares de f e c então temos uma extensão intervalar da função Lagrangeana. Denotamos esta extensão por $\mathfrak{L}$.

Apresentamos agora o teorema de Lagrange. Ele nos dá condições necessárias para o ponto x^* ser candidato a mínimo. Este resultado é demonstrado em vários livros de otimização e o leitor interessado pode consultar o livro de Izmailov e Solodov (2005).

Teorema 1 (Lagrange) *seja x^* um minimizador local do problema 5.1 - 5.3. Suponhamos que x^* é um ponto regular das restrições. Então existe um único $\lambda^* \in \mathbb{R}^m$ tal que*

$$L(x^*, \lambda^*) = 0. \tag{5.5}$$

5.4 Visão Geral do Método

Nesta seção apresentamos o escopo geral de nosso método. Esta visão será útil para que o leitor entenda os passos do algoritmo e como eles se encaixam no esquema geral. Todos os passos apresentados aqui são discutidos na próxima seção.

Seja $X \subseteq X_0$ uma caixa. Vimos na seção anterior que um ponto $x^* \in X$ regular das restrições é candidato a mínimo se existe λ^* tal que $L(x^*, \lambda^*) = 0$. Durante todo o capítulo supomos que x^*

é regular das restrições. Desejamos usar o método de Newton intervalar para limitar as raízes do sistema

$$L(x, \lambda) = 0 \tag{5.6}$$

$$c(x) = 0 \tag{5.7}$$

na caixa X. Note que este sistema tem $n + m$ equações e $n + m$ incógnitas e a diferença entre este problema e aqueles estudados no capítulo 3 é que neste caso não conhecemos limitantes para λ. Sendo assim nosso algoritmo trabalha em dois estágios.

No primeiro limitamos os multiplicadores de Lagrange λ através de mínimos quadrados intervalares. No segundo aplicamos o método de Newton intervalar para limitar caixas candidatas. Assim como nos casos anteriores, queremos descartar o mais rápido possível caixas que não contém mínimos globais.

O algoritmo mantém duas listas. A primeira, que chamamos de P, é a de caixas que ainda devem ser processadas. No inicio do programa temos $P = X_0$. A segunda, que chamamos de C, é a de caixas candidatas a solução. Esta lista começa o programa vazia. Nosso algoritmo exige um limitante superior para o mínimo global μ como entrada. Quando não há nenhuma informação sobre este valor tomamos $\mu = \infty$. Segue sua visão geral.

1. **Passo da escolha**: Se P está vazia então encerramos o programa. Caso contrário escolhemos uma caixa em P e a chamamos de X. Descartamos X de P.

2. **Passo das restrições**: Testamos a existência de zeros de c em X. Se provarmos que estes pontos não existem então descartamos X e voltamos ao passo 1.

3. **Limitando os lagrangeanos**: Procuramos limitantes para λ usando mínimos quadrados intervalar. Se não encontramos λ então dividimos X em duas caixas X' e X'', salvamos estas caixas em P e voltamos ao passo 1.

4. **Passo de atualização**: Atualizamos μ. Descartamos caixas em P e C que não podem conter minimizadores globais.

5. Testamos a existência de raízes de L em (X, λ). Se provarmos que estes pontos não existem então descartamos X e voltamos ao passo 1.

6. **Passo de redução**: Aplicamos o método de Newton intervalar ao sistema $\mathfrak{L} = 0$ para reduzir a região de busca. O resultado é a caixa X_1. Se X_1 é vazia então descartamos X e voltamos ao passo 1. Se X_1 pode ser salvo como solução então guardamos X_1 em S e voltamos ao passo 1.

7. **Passo de divisão**: Dividimos X_1 em duas caixas X_1' e X_1''. Salvamos estas caixas em P e voltamos ao passo 1.

5.5 Descrição do Algoritmo

Nesta seção detalhamos os passos do algoritmo. Ele é capaz de limitar por caixas de tamanho pré-fixado todos os mínimos globais regulares de f, restritos a c que pertencem a X_0. Apresentamos o algoritmo em partes.

5.5.1 Passo de Escolha

Neste passo escolhemos o elemento de P que será processado. Este passo é importante pois afeta a eficiência do algoritmo. Ao contrário do algoritmo para sistemas não lineares, a estrutura do problema nos dá informações que permitem escolher a caixa que será processada.

Se chamarmos de $X_1, X_2, \ldots, X_m$ as caixas de P, avaliamos $\mathfrak{f}(X_i)$ para $i = 1, \ldots, m$ e escolhemos a caixa que satisfaz

$$\min_{i=1\ldots m} \underline{\mathfrak{f}(X_i)}$$

onde $\underline{\mathfrak{f}(X_i)}$ é o limitante inferior do intervalo $\mathfrak{f}(X_i)$. Esta estratégia não garante que tomaremos elementos de P que contém mínimos globais. Ela apenas prioriza caixas com potencial para conter estes pontos.

Ao implementar nosso algoritmo de otimização restrita, tomamos o cuidado de manter a lista P ordenada segundo os valores de $\underline{\mathfrak{f}}$. Assim, a cada iteração tomamos o primeiro elemento de P como caixa a ser processada. Manter a lista P ordenada também nos ajudará a excluir caixas que não podem conter o mínimo global.

5.5.2 Passo das Restrições

A propriedade de inclusão garante que se $0 \notin \mathfrak{c}(X)$ então $0 \notin \mathfrak{c}(X')$ para qualquer $X' \subseteq X$. Esta propriedade nos dá um teste simples para existência de pontos viáveis em X.

- Se $0 \notin \mathfrak{c}(X)$ então não existe $x \in X$ tal que $c(x) = 0$ e descartamos a caixa.

Lembre que $\mathfrak{c}(X)$ é um vetor intervalar e $0 \notin \mathfrak{c}(X)$ significa que pelo menos uma entrada de $\mathfrak{c}$ não contém zero.

5.5.3 Limitando os Lagrangeanos

Neste passo procuramos limitantes para os multiplicadores de Lagrange. Esta informação é necessária para resolver o sistema 5.6 - 5.7 pelo método de Gauss Seidel intervalar. Note que fixada a caixa X, a extensão intervalar da Lagrangeana nos dá

$$\mathrm{f}(X) = \mathfrak{J}_1(X)\lambda_1 + \ldots + \mathfrak{J}_m(X)\lambda_m. \tag{5.8}$$

Agrupando os termos no lado direito temos

$$\mathrm{f}(X) = \mathfrak{J}^T(X)\lambda. \tag{5.9}$$

onde $\mathrm{f}(X)$ é um vetor de dimensão n, λ é um vetor de dimensão m e $\mathfrak{J}^T(X)$ é uma matriz de dimensão nxm, todos intervalares. Determinar limitantes λ dos multiplicadores de Lagrange é um problema de mínimos quadrados intervalar. Especificamente, desejamos encontrar limitantes do conjunto

$$Z := \{\lambda^* \in \mathbb{R}^m | \exists f^* \in \mathrm{f}(X) \text{ e } J^* \in \mathfrak{J}^T(X) \text{ tais que } ||J^*\lambda^* - f^*|| = \min_{\lambda \in \mathbb{R}^m} ||J^*\lambda - f^*||\}. \tag{5.10}$$

Este problema nem sempre tem solução. Em geral ele pode ser resolvido quando $\mathfrak{J}^T(X)$ tem posto completo, isto é, quando qualquer matriz $A \in \mathfrak{J}^T(X)$ tem posto completo. O leitor interessado em mínimos quadrados intervalares pode consultar o trabalho de Bentbib (2002).

A função *verifylss* do *INTLAB* resolve problemas de mínimos quadrados lineares para matrizes com posto completo. Este passo usa a *verifylss* para encontrar os limitantes λ.

- Dada a caixa X, se podemos resolver $\mathrm{f}(X) = \mathfrak{J}^T(X)\lambda$ usando *verifylss* então guardamos λ. Caso contrário seguimos para o passo de divisão.

É claro que se conhecermos limitantes para λ associados a X podemos pular este passo. A propriedade de inclusão evita cálculos desnecessários. Se conhecermos limitantes de λ associados a X então podemos usar estes limitantes associados a qualquer caixa $X^* \subseteq X$.

5.5.4 Passo de Atualização

Neste passo atualizamos o valor μ. A medida que exploramos X_0 obtemos informações sobre o limitante superior do mínimo global. Usamos estas informações para excluir caixas em P e C que não podem conter mínimos globais.

No capítulo anterior argumentamos que qualquer ponto em X_0 pode ser usado para atualizar μ. Isto porque todos os pontos em X_0 são considerados viáveis no caso irrestrito. Naquele capítulo tomamos uma aproximação do ponto médio de X_0 para atualizar o limitante superior do mínimo global.

Em problemas restritos o passo de atualização é mais complicado. Dada uma caixa X_0 preci-

samos decidir se ela contém ou não pontos viáveis. Somente atualizamos μ se pudermos garantir a existência pontos viáveis na caixa. Lembrando que nosso conjunto de restrições é dado por c, somente atualizamos μ se existe pelo menos um ponto $x \in X_0$ tal que

$$c(x) = 0.$$

O método de Krawczyk nos dá condições suficientes para a existência de raízes de um sistema quadrado de equações não lineares em uma caixa X. Não podemos usar estas condições imediatamente pois o sistema c não é quadrado e temos mais variáveis do que equações.

Alguns autores recomendam que o teste de Krawczyk seja aplicado em c fixando quantas variáveis forem necessárias para tornar o sistema quadrado. Nesta seção apresentamos um método baseado na decomposição QR que procura as variáveis que são responsáveis pelas maiores variações de c. Em nosso algoritmo usamos estas variáveis como livres e fixamos aquelas que são responsáveis por pequenas variações em c.

Bentbib (2002) introduziu a decomposição QR intervalar através da generalização das matrizes de Householder. Em nosso trabalho apresentamos a decomposição QR intervalar a partir da generalização das matrizes de Givens com permutação de colunas e linhas. Esta decomposição é dada pelo seguinte algoritmo,

Entradas

- Uma matriz A com m linhas e n colunas, $n \geq m$.
- Um vetor P de dimensão n com $P(1) = 1, P(2) = 2, \ldots, P(n) = n$.

Saídas

- A matriz A triangular superior com m linhas e n colunas, $n \geq m$.
- Um vetor P de permutações com dimensão n.

Passos do Algoritmo

1. Tome $k = 1$.
2. Se $k > m$ encerre o programa.
3. Tome i como o índice, a partir de k, onde o vetor coluna A_i tem a maior norma.
4. Se a maior norma obtida no passo anterior é igual a zero então encerre o programa.
5. Permute as colunas A_i e A_k.

6. Permute os elementos P_i e P_k.

7. Tome i como o índice, a partir de k, onde o vetor linha A_i tem a maior norma.

8. Permute as linhas A_i e A_k.

9. Para j de $k+1$ até m faça

10. Tome $R = \frac{A_{jk}}{A_{kk}}$.

11. Tome $A_{kk} = \sqrt{A_{kk}^2 + A_{jk}^2}$.

12. Tome $A_{jk} = 0$

13. Para p de $k+1$ até n faça

14. Tome $u = A_{kp}$.

15. Tome $A_{kp} = \frac{A_{kp}+A_{jp}R}{\sqrt{1+R^2}}$.

16. Tome $A_{kp} = \frac{-uR+A_{jp}}{\sqrt{1+R^2}}$.

17. Fim de Para.

18. Fim de Para.

19. Tome $k = k+1$.

20. Volte ao passo 2.

Note que este algoritmo pode ser interrompido no passo 4. Neste caso não sabemos dizer se há ou não pontos viáveis na caixa de interesse e o passo de atualização se encerra. Caso o algoritmo termine no passo 2 então podemos tomar as variáveis correspondentes aos m primeiros índices de P como livres e fixar as restantes. Agora podemos aplicar o método de Krawczyk e procurar por limitantes do mínimo global.

Caso o algoritmo de Krawczyk seja incapaz de dizer se há ou não pontos viáveis em X então encerramos o passo de atualização. Por outro lado, se o algoritmo de Krawczyk garantir a existência de pontos viáveis em X então tomamos

$$\mu = \min(\mu, \overline{f(X)}).$$

Uma vez que atualizamos μ eliminamos todas as caixas Y em P que satisfazem

$$\underline{f(Y)} > \mu. \tag{5.11}$$

Como mantemos P ordenada, basta encontrar o primeiro elemento que satisfaz esta relação e descartamos todas as caixas deste elemento até o fim da lista.

Ao contrário do algoritmo para sistemas não lineares, não mantemos uma lista de caixas salvas S e sim uma lista de candidatas C. Essa diferença acontece porque, neste algoritmo, um elemento em C pode limitar apenas mínimos locais de f. Sendo assim, quando atualizamos μ faz sentido percorrer C e eliminar todos os elementos Y desta lista que satisfazem a relação acima.

É possível usar nosso algoritmo para encontrar **todos** os mínimos de f sujeitos a c. Basta não aplicar o teste proposto acima na lista C. Desta forma, todas as caixas que entram nesta lista permanecem nela até o fim do programa e garantimos limitar todos os mínimos de f em X_0 sujeitos a c.

5.5.5 Passo de Redução

Neste passo aplicamos o método de Newton intervalar descrito em 3.3 ao sistema 5.6 - 5.7. Este passo só pode ser aplicado depois de encontrarmos limitantes λ dos multiplicadores de Lagrange. Para facilitar a notação consideramos o vetor intervalar $Y = (X, \lambda)$. Isto é, o vetor que tem X nas n primeiras coordenadas e λ nas restantes. Desejamos reduzir as dimensões de Y e com isso encontrar uma região de busca menor em X ou limitantes melhores para λ.

Inicialmente aplicamos o primeiro estágio do algoritmo de Gauss Seidel intervalar ao sistema 5.6 - 5.7 partindo de Y. Aplicamos este estágio enquanto houver redução no domínio de busca e no máximo por 5 vezes. Isto é, enquanto $Y' \neq Y$ e não excedermos 5 aplicações.

O cuidado de definir um número máximo de aplicações do primeiro estágio de Gauss Seidel garante que nosso algoritmo deixa este passo em algum momento. Esta escolha também garante que não tenhamos gastos excessivos com o cálculo de caixas que reduzem pouco o domínio de busca. O número 5 é arbitrário e pode ser substituído de acordo com a implementação ou problema a ser resolvido.

Aplicamos então o segundo estágio do algoritmo de Gauss Seidel intervalar. Lembre que este estágio é aplicado apenas uma vez a cada iteração do método de Newton. Se este estágio do método devolver um salto em X então devemos guarda-lo. Isto é, se a aplicação deste estágio devolver um índice i entre 1 e n e dois intervalos disjuntos $\mathbf{x}$ e $\mathbf{y}$ conforme descrito em 3.3 então salve estas informações para usá-las no passo de divisão.

5.5.6 Passo de Divisão

Se uma caixa não pode ser processada ou não foi reduzida o suficiente devemos dividi-la. Vários autores discutiram heurísticas para divisão de caixas. Os trabalhos de Kearfott (1996b), Boyd, Gosh e Magnani (2003) e Pedamallu et al. (2008) são boas referências para o assunto.

Em nosso algoritmo adotamos um esquema de bisseção simples, considerando dois casos.

1. Não temos informações sobre saltos na caixa. Neste caso a dividimos X em X' e X'' a partir da direção com maior comprimento.

2. Sabemos que X veio do passo de redução e nele salvamos o índice de maior salto durante a aplicação do segundo estágio de Gauss Seidel. Neste caso dividimos X em X' e X'' nessa direção. Se i é o índice dessa direção e $\mathbf{x}$ e $\mathbf{y}$ são os intervalos resultantes então tomamos $X_i' = \mathbf{x}$ e $X_i'' = \mathbf{y}$.

Uma vez que X' e X'' foram obtidas, devemos guarda-las em L. Neste ponto avaliamos $\underline{\mathfrak{f}}(X')$ e $\underline{\mathfrak{f}}(X'')$. Guardamos estas caixas em L e ordenamos a lista de acordo com os valores de $\underline{\mathfrak{f}}$.

5.5.7 Tomando Uma Caixa Como Candidata

A caixa $X \subseteq X_0$ é candidata a solução e a guardamos em C se $W(X) < \varepsilon_X$, $||\mathfrak{L}(X, \lambda)|| < \varepsilon_F$ e $||\mathfrak{c}(X)|| < \varepsilon_C$. Os valores positivos ε_X, ε_F e ε_C são dados pelo usuário e não devem ser menor do que a unidade de arredondamento definida por M.

Estas condições controlam o comprimento máximo de X e a maior distância permitida da função Lagrangeana e das restrições à origem. Podemos omitir uma destas condições tomando ε_X, ε_F ou ε_C como infinito.

Tomar $\varepsilon_X = \infty$ aumenta a eficiência do algoritmo nos casos em que sabemos que o problema tem tem muitas soluções em X_0. Não recomendamos tomar valores altos de ε_F ou ε_C pois isto introduz ruídos nos resultados. Outros critérios para aceitar X como solução são discutidos por Hansen e Walster (2004), Hargreaves (2002) e Ratschek e Rokne (1988).

Apenas nomeamos as caixas como candidatas a solução. Isto porque podemos excluir caixas em C durante o processo se descobrirmos que ela limita apenas pontos de mínimo local do problema. O conjunto solução S é dado pelas caixas que permaneceram em C até fim do programa. Este conjunto pode ter caixas que não contém soluções além de caixas que limitam mais de uma vez a mesma solução.

5.5.8 O Algoritmo

Apresentamos agora nosso algoritmo para problemas de otimização com restrições de igualdade. O algoritmo deve ser seguido em seqüência exceto quando indicamos o contrário. Omitimos os passos em que testamos se o número máximo de iterações ou avaliações de função ou restrição foi atingido. Não é difícil incorporar estes passos ao algoritmo.

Entradas

- A extensão $(M, \mathfrak{f}, \mathfrak{g}, \mathfrak{c}, \mathfrak{J})$
- A caixa inicial representável X_0.

- Um limitante superior do mínimo global μ. Pode ser ∞.
- Valores ε_X, ε_F e ε_C compatíveis com M. Veja a subseção anterior.

Saídas

- Um conjunto de caixas $S := \{S_1, \ldots, S_k\}$ tais que se x^* é solução do problema 5.1 - 5.3 então existe ao menos um índice k tal que $x^* \in S_k$.
- Um limitante superior do mínimo global μ.

Não há relação de $1-1$ entre as soluções de 5.1 - 5.3 e os elementos de S. É possível que uma mesma solução esteja em mais de uma caixa e também é possível que S tenha caixas que não contém soluções.

Passos do Algoritmo

1. Se P está vazia então encerre o programa. Caso contrário tome a primeira caixa de P e chame-a de X. Exclua X de P.
2. Avalie $\mathfrak{c}(X)$ e $\mathfrak{J}(X)$.
3. Se alguma entrada de $\mathfrak{c}(X)$ não contém zero então exclua X e volte ao passo 1.
4. Avalie $\mathfrak{f}(X)$ e $\mathfrak{g}(X)$.
5. Usando mínimos quadrados intervalares avalie $\mathfrak{J}^T(X)\lambda = g(X)$ para limitar os lagrangeanos λ.
6. Se não foi possível encontrar limitantes para todos os lagrangeanos então divida X na direção de maior comprimento. Guarde as duas caixas em P. Ordene P de acordo com $\underline{\mathfrak{f}}$. Volte ao passo 1.
7. Aplique o procedimento descrito na subseção 5.5.4.
8. Se pudermos afirmar que existem pontos viáveis do problema em X então $\mu = \min(\mu, \overline{\mathfrak{f}(X)})$.
9. Se atualizou μ então exclua todas as caixas Y em P e C tais que $\underline{\mathfrak{f}(Y)} > \mu$.
10. Avalie $\mathfrak{L}(X, \lambda)$ e sua matriz jacobiana $\mathfrak{J}\mathfrak{L}(X, \lambda)$.
11. Estime $A = mid(\mathfrak{J}\mathfrak{L}(X, \lambda))$.
12. Estime os valores singulares A com aritmética de ponto flutuante. Chame de σ_1 o menor destes valores e σ_n o maior.

13. Se $\frac{\sigma_n}{\sigma_1} < \nu$ então divida X na direção de maior comprimento. Guarde as duas caixas em P. Ordene P de acordo com $\underline{f}$ e volte ao passo 1. Em nossos experimentos usamos $\nu = 10^{-6}$.

14. Estime $\bar{x} = mid(X, \lambda)$.

15. Estime $M = A^{-1}\mathfrak{J}\mathfrak{L}(X, \lambda)$ e $b = -A^{-1}\mathfrak{L}(\bar{x}^I)$.

16. Aplique o estágio não estendido de Gauss Seidel ao sistema $M(z - \bar{x}) = b$ tomando (X, λ) como estimativa inicial. O resultado é uma nova caixa (X_1, λ_1).

17. Se alguma entrada de X_1 é vazia então exclua X e volte ao passo 1.

18. Estime $\bar{x} = mid(X_1, \lambda_1)$ e $\mathfrak{f}(X_1)$.

19. Se $\underline{\mathfrak{f}(X_1)} > \mu$ então exclua X e volte ao passo 1.

20. Avalie $\mathfrak{L}(X_1, \lambda_1)$.

21. Se alguma entrada de $\mathfrak{L}(X_1, \lambda_1)$ não contém zero então exclua X e volte ao passo 1.

22. Se $W(X_1) < \varepsilon_X$, $||\mathfrak{L}(X_1, \lambda_1)|| < \varepsilon_F$ e $||\mathfrak{c}(X_1)|| < \varepsilon_C$ então guarde X_1 em C e volte ao passo 1.

23. Estime $b = -A^{-1}\mathfrak{L}(\bar{x}^I)$.

24. Se $X_1 \neq X$ e aplicamos o passo 15 menos do que 6 vezes então substitua $X = X_1$. Volte ao passo 15.

25. Aplique o estágio estendido de Gauss Seidel ao sistema $M(z - \bar{x}) = b$ tomando (X_1, λ_1) como estimativa inicial. O resultado é uma nova caixa (X_2, λ_2). Salve a direção em que ocorre o maior salto e os intervalos resultantes nessa direção.

26. Se algum elemento de X_2 é vazio então exclua X e volte ao passo 1.

27. Avalie $\mathfrak{f}(X_2)$.

28. Se $\underline{\mathfrak{f}(X_2)} > \mu$ então exclua X e volte ao passo 1.

29. Avalie $\mathfrak{L}(X_2, \lambda_2)$.

30. Se alguma entrada de $\mathfrak{L}(X_2, \lambda_2)$ não contém zero então exclua X e volte ao passo 1.

31. Se $W(X_2) < \varepsilon_X$, $||\mathfrak{L}(X_2, \lambda_2)|| < \varepsilon_F$ e $||\mathfrak{c}(X_2)|| < \varepsilon_C$ então guarde X_2 em C e volte ao passo 1.

32. Se nenhum salto foi salvo então divida X_2 na direção de maior comprimento. Guarde as duas caixas em P. Ordene P de acordo com $\underline{f}$. Volte ao passo 1.

33. Se algum salto foi salvo então divida X_2 na direção deste salto. Utilize os intervalos resultantes nessa direção. Ordene P de acordo com $\underline{f}$. Volte ao passo 1.

5.6 A função intmincon

A função *intmincon* do *INTSOLVER* implementa o algoritmo descrito na seção anterior. Nesta seção ilustramos seu uso e algumas de suas opções.

O *INTSOLVER* deve ser simples para usuários acostumados ao Matlab. A função *intmincon* exige poucos parâmetros de entrada. Por exemplo, desejamos encontrar as soluções globais de

$$\begin{aligned} &\min x_1^2 + (x_2 - 1)^2 \\ &\text{S. a } x_2 - x_1^2 = 0 \end{aligned}$$

na caixa $-1 \leq x_1, x_2 \leq 1$. Para realizar esta tarefa com a função *intmincon* precisamos apenas das funções f e c e da caixa X_0.

```
>> f  = @(x) x(1)^2 + (x(2) - 1)^2;
>> c  = @(x) x(2) - x(1)^2;
>> x0 = [infsup(-1,1), infsup(-1,1)];

>>[x,iv] = intmincon(f, x0, c)
 ite   evf   evg   evh   evc   evcg   evch   min         nsol   nlist   completed

 0     1     0     0     0     0      0      Inf         0      1       0.000

 10    44    27    3     29    29     3      2.000e+00   0      7       0.171

 20    447   301   13    303   303    13     9.845e-01   2      7       0.941
>> x
intval x =
[0.7071,0.7072]   [0.5000,0.5001]
[-0.7072,-0.7071] [0.5000,0.5001]
[0.7071,0.7072]   [0.4999,0.5001]
[-0.7072,-0.7071] [0.4999,0.5001]
>> iv
intval iv =
[0.7499,0.7501]
[0.7499,0.7501]
[0.7499,0.7501]
[0.7499,0.7501]
```

A função *intmincon* assume os seguintes padrões,

- Número máximo de iterações: 1000.
- Número máximo de Avaliações de função: 10000.
- Número máximo de Avaliações de restrição: 10000.
- $\mu = \infty$.
- $\varepsilon_X = 10^{-8}$.
- $\varepsilon_F = 10^{-8}$.
- $\varepsilon_C = 10^{-8}$.
- Nível do Display: 10.
- Iterar até limitar todos os mínimos globais: Não.

Todas os parâmetros acima podem ser modificados. Em particular quando optamos por iterar até limitar todos os mínimos globais, a função ignora quaisquer valores em número máximo de iterações, número máximo de avaliações de função e número máximo de avaliações de restrição. Obviamente esta escolha deve aumentar o tempo de execução do algoritmo e cabe ao usuário o uso coerente desta opção.

No próximo exemplo mudamos as quantidades ε_X, ε_X e ε_C para 10^{-10}. Também atribuímos o valor 0 ao nível do display de forma que as informações sobre o processo não serão exibidas.

```
>> opt = intoptimset();
>> opt(1) = 10^-10;
>> opt(2) = 10^-10;
>> opt(3) = 10^-10;
>> opt(8) = 0;

>> [x, iv, ef] = intmincon(f, x0, c, opt)
>> x
intval x =
[0.7071,  0.7072] [0.5000,0.5001]
[-0.7072,-0.7071] [0.5000,0.5001]
[0.7071,  0.7072] [0.4999,0.5001]
[-0.7072,-0.7071] [0.4999,0.5001]
>> iv
intval iv =
```

```
[0.7499,0.7501]
[0.7499,0.7501]
[0.7499,0.7501]
[0.7499,0.7501]
>> ef
ef =
     0
```

A saída x mostra as caixas soluções encontradas pela função. Em iv temos $f(X_i)$ onde X_i são as caixas em x. Finalmente o parâmetro de saída $ef = 0$ nos diz que a função foi finalizada com sucesso.

As funções do *INTLAB* estão documentadas e o usuário que desejar maiores informações deve digitar

```
>> doc nome_da_função
```

no Matlab.

5.7 Estudos Computacionais

Para avaliar o desempenho da função *intmincon* escolhemos 3 problemas da literatura de otimização global. Levamos em conta a precisão e a eficiência da função. Com relação à precisão veremos que a função é capaz de encontrar todas as soluções dos problemas desde que o número máximo de iterações ou avaliações de função e restrição não sejam atingidos. Com relação à eficiência, apresentamos em cada caso o número de iterações, avaliações de função, gradiente, Hessiana da função f e avaliações de função, gradiente e Hessiana de c. Reportamos também o tempo de execução em segundos em cada caso.

A menos de menção contrária, o número máximo de iterações em cada teste é 1000, o número máximo de avaliações de função e restrição é 10000, $\mu = +\infty$ e os valores ϵ_X, ϵ_F e ϵ_C são iguais a 10^{-8}. As funções foram implementadas em Matlab exatamente como aparecem descritas abaixo. A extensão é feita apenas trocando variáveis reais por intervalos. Os testes foram realizados em um AMD Sempron de $800MHZ$ e $1GB$ de memória RAM com sistema operacional Linux.

5.7.1 Runarsson

Este problema foi extraído de Runarsson e Yao (2000). Desejamos encontrar todos os mínimos globais regulares de

$$\min \; -\sqrt{(n)}^n \prod_{i=1}^{n} x_i$$

$$\text{S. a} \sum_{i=1}^{n} x_i^2 - 1 = 0$$

na caixa $0 \leq x_1, \ldots, x_n \leq 1$.

Este problema tem uma única solução dada por

$$x^* = \left(\frac{1}{n^{0.5}}, \ldots, \frac{1}{n^{0.5}}\right)$$

e sabemos que $f(x^*) = -1$. Usando a função *intmincon* obtemos

#Variáveis	#Iterações	#Aval. f	#Aval. c	#Aval. ∇f	#Aval. ∇c.
2	14	543	368	368	368
3	81	806	574	580	580
4	381	10001	6746	6833	6833

#Variáveis	Tempo(s)	μ	#Soluções	Concluído(%)
2	10.448	-1	1	100
3	25.887	-0.9164	1	100
4	312.81	-0.7545	8	50

Observe que para $n = 4$ a função alcança o número máximo de avaliações de função. Nos demais casos encontramos caixas que contém o ponto

$$x^* = (\frac{1}{n^{0.5}}, \ldots, \frac{1}{n^{0.5}}).$$

A tabela abaixo traz estatísticas sobre estas caixas

#Variáveis	$\|\|\mathfrak{L}(X,\lambda)\|\|$	$\mathbb{W}(X)$	$\|\|\mathfrak{c}(X)\|\|$
2	3.7442e-11	9.7256e-14	4.4018e-9
3	4.5498e-9	2.3414e-9	1.6880e-9

5.7.2 Runarsson 2

Este problema foi extraído de Runarsson e Yao (2000). Desejamos encontrar todos os mínimos globais regulares de

$$\begin{aligned} \min\ & x_1^2 + (x_2 - 1)^2 \\ \text{S. a}\ & x_2 - x_1^2 = 0 \end{aligned}$$

na caixa $-1 \leq x_1, x_2 \leq 1$.

Este problema tem duas soluções dadas por

$$x^* = \left(\pm\frac{1}{2^{0.5}}, \frac{1}{2}\right)$$

e sabemos que $f(x^*) = 0.75$. Usando nossa função obtemos

#Iterações	#Aval. f	#Aval. c	#Aval. ∇f	#Aval. ∇c.
25	534	365	367	367

Tempo(s)	μ	#Soluções	Concluído(%)
11.100	0.75	4	100

Encontramos mais de um limitante para os mínimos globais x^*. Isto se deve ao passo de divisão do algoritmo, veja a subseção 3.9.2 para maiores detalhes. Todas as caixas soluções deste exemplo limitam x^*. A tabela abaixo traz estatísticas sobre elas

$\|\|\mathfrak{L}(X,\lambda)\|\|$	$\mathbb{W}(X)$	$\|\|\mathfrak{c}(X)\|\|$
2.5634e-9	2.072e-9	1.0e-9
4.7344e-9	2.072e-9	1.0e-9
5.432e-10	7.79e-10	1.0e-9
3.715e-10	7.79e-10	1.0e-9

5.7.3 Runarsson 3

Este problema foi extraído de Runarsson e Yao (2000). Desejamos encontrar todos mínimos globais regulares de

$$\begin{aligned} \min\ & \exp^{x_1 x_2 x_3 x_4 x_5} \\ \text{S. a}\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ & x_2 x_3 - 5 x_4 x_5 = 0 \\ & x_1^3 + x_2^3 + 1 = 0 \end{aligned}$$

na caixa $-2 \leq x_1, x_4, x_5 \leq 0$, $1 \leq x_2, x_3 \leq 2$.

Este problema tem uma única solução dada por

$$x^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)$$

e sabemos que $f(x^*) = 0.0539498$. Usando a função *intmincon* obtemos

#Iterações	#Aval. f	#Aval. c	#Aval. ∇f	#Aval. ∇c.
25	534	365	367	367

Tempo(s)	μ	#Soluções	Concluído(%)
11.100	0.75	4	100

Encontramos mais de um limitante para os mínimos globais x^*. Isto se deve ao passo de divisão do algoritmo, veja a subseção 3.9.2 para maiores detalhes. Todas as caixas soluções deste exemplo limitam x^*. A tabela abaixo traz estatísticas sobre elas

$\|\|\mathfrak{L}(X)\|\|$	$\mathbb{W}(X)$	$\|\|\mathfrak{c}(X)\|\|$
2.5634e-9	2.072e-9	1.0e-9
4.7344e-9	2.072e-9	1.0e-9
5.432e-10	7.79e-10	1.0e-9
3.715e-10	7.79e-10	1.0e-9

s

Capítulo 6

Conclusões

6.1 Considerações Finais

Neste trabalho apresentamos métodos capazes de limitar todas as soluções de três problemas de otimização global. A ferramenta que tornou possível a construção destes métodos é a aritmética intervalar. Ela permite realizar cálculos verificados de expressões complexas e dá informações sobre o comportamento global da função em uma caixa.

Um dos principais resultados da aritmética intervalar é a inclusão. Esta propriedade tem implicações importantes em nosso trabalho. Dentre elas destacamos a capacidade de excluir caixas que não contém raízes ou mínimos globais em nossos algoritmos. Outras propriedades importantes dessa aritmética são o teorema do valor médio intervalar e o método de Newton intervalar. Eles foram explorados ao longo dos capítulos e o método de Newton intervalar é a base para os algoritmos estudados.

O principal resultado do nosso trabalho é o *INTSOLVER*, um pacote escrito em Matlab que implementa os algoritmos discutidos nos capítulos anteriores e que resolve, sob certas condições, problemas de otimização global em dimensão baixa. Testamos o pacote com problemas clássicos da otimização global e a última seção de cada capítulo deste texto dedica-se a apresentação dos principais resultados dos testes. O *INTSOLVER* e os arquivos de teste estão disponíveis em

http://www.mathworks.com/matlabcentral/fileexchange/25211

O *INTSOLVER* é simples e usuários do Matlab não terão dificuldades em entede-lo. A sintaxe das funções é semelhante àquelas do *optimization toolbox* da *mathworks* e ao longo da dissertação ilustramos o uso de suas principais funções. Além disso o *INTSOLVER* é um pacote pequeno, com aproximadamente 2500 linhas de código, e está aberto para usuários interessados em modifica-lo.

Embora a teoria que apresentamos seja válida para problemas em qualquer dimensão, não conseguimos resolver problemas médios e grandes usando nosso pacote. Esta limitação é a principal preocupação para trabalhos futuros. Em particular, as próximas implementações do *INTSOLVER*

devem considerar:

- Estudar, implementar e comparar outros algoritmos de otimização restrita com o implementado no *INTSOLVER*.
- Incluir uma função no *INTSOLVER* capaz de resolver problemas com restrições de desigualdade.
- Comparar a performance do *INTSOLVER* com outros pacotes de otimização global.
- Paralelizar nossos algoritmos para aumentar sua eficiência. Permitir que problemas maiores sejam resolvidos.
- Estudar a possibilidade de substituir a hessiana por aproximações mais baratas nos algoritmos de otimização.
- Implementar o *INTSOLVER* em uma linguagem mais eficiente como $C++$ ou $FORTRAN$.

Apêndice A

Diferenciação Automática

O cálculo de derivadas é essencial em computação científica. Como vimos nos capítulos anteriores, nossos algoritmos dependem do cálculo de gradientes e hessianas para funcionar corretamente. Por outro lado, o usuário do *INTSOLVER* deve notar que suas funções não exigem expressões das derivadas como parâmetro de entrada. Isto acontece porque nossas funções avaliam automaticamente todas as derivadas necessárias.

Neste apêndice discutimos as idéias principais da diferenciação automática e ilustramos seu uso. As funções de diferenciação automática que usamos foram escritas por Rump (1999) e estão presentes no *INTLAB*. Nossa exposição segue o trabalho de Hargreaves (2002).

A diferenciação automática não está ligada exclusivamente à aritmética intervalar. Podemos usá-la para calcular derivadas e hessianas de funções com aritmética de ponto flutuante. No entanto, neste caso os resultados estão sujeitos a erros de arredondamento. Usando aritmética intervalar os cálculos são verificados e garantimos que os intervalos obtidos contém os verdadeiros valores das derivadas. Para facilitar as idéias apresentamos a diferenciação automática usando o conjunto do números reais R. A passagem para o caso I(M) deve respeitar as regras expostas no capítulo 2.

Seja $f : \mathrm{R}^m \to \mathrm{R}^n$ uma função que depende somente de operações elementares com derivadas conhecidas, por exemplo, $+, -, *, /, \sin, \cos, \ln, \exp$. A expressão para $f(x)$, $x = (x_1, \ldots, x_m)$ pode ser decomposta em uma lista de equações representando a função,

$$t_i(x) = g_i(x) = x_i \qquad i = 1, \ldots, m \tag{A.1}$$

$$t_i(x) = g_i(t_1(x), \ldots, t_{i-1}(x)) \qquad i = m+1, \ldots, l. \tag{A.2}$$

onde t_i e g_i são funções escalares e somente uma operação elementar ocorre em cada g_i. As funções g_i são diferenciáveis pois são compostas de operações elementares. Usando a regra da cadeia nas relações acima temos

$$\frac{\partial t_i}{\partial t_j} = \delta_{ij} + \sum_{k=j}^{i-1} \frac{\partial g_i}{\partial t_k} \frac{\partial t_k}{\partial t_j} \text{ onde } \delta_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{caso contrário} \end{cases}$$

Escrevendo $t = (t_1, \ldots, t_l)$ e $g = (g_1, \ldots, g_l)$, podemos escrever A.1 - A.2 como

$$Dt = I + DgDt$$

onde

$$Dg = \frac{\partial g_i}{\partial t_j} = \begin{pmatrix} 0 & \ldots & & \\ \frac{\partial g_2}{\partial t_1} & 0 & \ldots & \\ \frac{\partial g_3}{\partial t_1} & \frac{\partial g_3}{\partial t_2} & 0 & \ldots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix},$$

e

$$Dg = \frac{\partial t_i}{\partial t_j} = \begin{pmatrix} 1 & 0 & \ldots & \\ \frac{\partial t_2}{\partial t_1} & 1 & \ddots & \\ \frac{\partial t_3}{\partial t_1} & \frac{\partial t_3}{\partial t_2} & 1 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}.$$

Isto implica que $(I - Dg)Dt = I$. Este sistema pode ser resolvido para qualquer coluna de Dt através de substituição para frente. Para encontrar todas as derivadas parciais de f devemos resolver o sistema para as m primeira colunas de Dt.

Para usar diferenciação automática no *INTLAB* usamos o comando *gradientinit*. Por exemplo desejamos calcular a derivada de $f(x) = 2x + cos(2x)$ no intervalo $x = [3, 3]$. Com o *INTLAB* temos

```
>> f = @(x) 2*x + cos(2*x);
>> x = gradientinit(infsup(3,3));
>> f(x)
intval gradient value ans.x =
    6.9601
intval gradient derivative(s) ans.dx =
    2.5588
```

Referências Bibliográficas

754, I. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic.* [S.l.], 1985.

ALEFELD, G.; HERZBERGER, J. *Introduction to Interval Computations.* Second. New York: Academic Press, 1983.

BENTBIB, A. H. Solving full rank interval least squares problem. *Applied Numerical Mathematics*, v. 41, p. 283–294, 2002.

BOYD, S. et al. *Branch and Bound Methods.* [S.l.], 2003.

EASOM, E. E. *A survey of global optimization techniques.* [S.l.], 1990.

FLOUDAS, C. et al. *Handbook of Test Problems in Local and Global Optimization.* First. Dordrecht: Kluwer, 1999.

FLOUDAS, C. A. *Deterministic Global Optimization.* First. Dordrecht: Kluwer Academic, 2000.

HAMMER, R. et al. *Numerical Toolbox for Verified Computing I.* First. Heidelberg: Springer-Verlag, 1993.

HANSEN, E. Global optimization using interval analysis. the multi-dimensional case. *Numerical Mathematics*, v. 34, p. 247–270, 1980.

HANSEN, E.; GREENBERG, R. An interval newton method. *Applied Mathematics And Computation*, v. 12, p. 89–98, 1983.

HANSEN, E.; SENGUPTA, S. Bounding solutions of systems of equations using interval analysis. *BIT*, v. 21, p. 203–211, 1981.

HANSEN, E.; SMITH, R. Interval arithmetic in matrix computations, part ii. *SIAM Journal of Numerical Analysis*, v. 4, p. 1–9, 1967.

HANSEN, E. R.; WALSTER, W. G. *Global Optimization Using Interval Analysis.* Second. New York: Marcel Dekker and Sun Microsystems, 2004.

HARGREAVES, G. I. *Interval analysis in Matlab.* [S.l.], 2002. `http://www.ma.man.ac.uk/~nareports`.

IZMAILOV, A.; SOLODOV, M. *Otimização - Volume 1.* First. Rio de Janeiro: IMPA, 2005.

KEARFOTT, R. B. Interval arithmetic: A fortran 90 module for an interval data type. *ACM Trans. Math. Software*, v. 22, p. 385–392, 1996.

_____. *Rigorous Global Search: Continuous Problems.* First. Dordrecht: Kluwer Academic, 1996.

KEARFOTT, R. B. et al. Intlib - a portable fortran 77 interval standard function library. *ACM Trans. Math. Software*, v. 20, n. 4, p. 447–459, 1994.

KRAWCZYK, R. Newton-algorithmen zur bestimmung von nulstellen mit fehlerschranken. *Computing*, v. 4, p. 187–201, 1969.

LEVY, A. V. et al. *Topics in Global Optimization.* First. [S.l.]: Springer-Verlag, 1981.

LUENBERGER, D. G. *Linear and Nonlinear Programming.* Second. New York: Addison-Wesley, 1989.

MARANAS, C. D.; FLOUDAS, C. Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, v. 7, n. 2, p. 143–182, 1995.

MARTINEZ, J. et al. On an efficient use of gradient information for accelerating interval global optimization algorithms. *Numerical Algorithms*, v. 37, p. 61–69, 2004.

MICHALEWICZ, Z. *Genetic Algorithm + Data Structures = Evolution Programs.* First. Heideberg: Springer-Verlag, 1996.

MOORE, R. E. A test for existence of solutions to nonlinear systems. *SIAM Journal of Numerical Analysis*, v. 14, n. 4, p. 611–615, 1977.

_____. *Methods and Aplications of Interval Analysis.* First. Philadelphia: SIAM, 1979.

_____. Global optimization to prescribed accuracy. *Computers and Mathematical Applications*, v. 21, n. 6, p. 25–39, 1991.

MORé, J. J. et al. Testing unconstrained optimization software. *ACM Transaction Mathematical Software*, v. 7, n. 1, p. 17–41, 1981.

NOCEDAL, J.; WRIGHT, J. *Numerical Optimization.* Second. New York: Springer, 2006.

PEDAMALLU, C. S. et al. Efficient interval partitioning for constrained global optimization. *Journal of Global Optimization*, v. 42, p. 369–384, 2008.

RALL, L. B. A comparison of the existence theorems of kantorovich and moore. *SIAM Journal of Numerical Analysis*, v. 17, n. 1, p. 148–161, 1980.

RATSCHEK, H.; ROKNE, J. *New Computer Methods for Global Optimization.* First. Chichester: Horwood Limited, 1988.

RATZ, D. *Inclusion isotone extended interval arithmetic.* [S.l.], 1996.

RATZ, D.; CZENDES, T. On the selection of subdivision directions in interval branch and bound methods for global optimization. *Journal of Global Optimization*, v. 7, p. 183–207, 1995.

RUMP, S. M. *INTLAB - INTerval LABoratory.* [S.l.], 1999. `http://www.ti3.tu-harburg.de/rump/`.

_____. Rigorous and portable standard functions. *BIT*, v. 41, n. 3, p. 540–562, 2001.

RUNARSSON, T. D.; YAO, X. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, v. 4, n. 3, p. 274–283, 2000.

TORN, A.; ZILINSKAS, A. *Global Optimization.* First. [S.l.]: Springer-Verlag, 1989.

TUY, H. *Convex Analysis and Global Optimization.* First. Dordrecht: Kluwer Academic, 1997.

WOLFE, M. A. A modification of krawczyk's algorithm. *SIAM Journal of Numerical Analysis*, v. 17, n. 3, p. 376–379, 1980.

Índice Remissivo

Aritmética
  De Ponto Flutuante, 5, 6, 16, 24, 27, 77
  Intervalar, 5, 11, 17, 18, 24, 27

C/C++, 6
Cálculo Verificado, 11, 19
Caixa
  Solução, 50
Caixas, *veja* Vetor Intervalar
Condições de Lagrange, 78

Decomposição QR Intervalar, 83
Diferenciação Automática, 3
Divisão Entendida, 22

Erro de Arredondamento, 11, 12
Extensão Intervalar, 20
Extensões Intervalares, 20

Fortran, 6
Função Intervalar, 17
Função Lagrangeana, 79
Função Racional Intervalar, 18

IEEE754, 6, 19, 22
Inclusão Monotônica, 16
Intervalos, 7–10, 15, 18
  Comprimento, 9
  Módulo, 9
  Ponto Médio, 9
  Raio, 9
INTLAB, 28, 82
  Intminunc, 61
  Verifylss, 82
Intlab, 5, 6, 10, 13, 14, 16, 23, 24
Intlib, 6
INTSOLVER, 2, 23, 27, 36, 42, 53, 77, 89
  Intksolve, 42
  Intmincon, 77, 89
  Intminunc, 53
  Intsolve, 36

Java, 6

Mínimos Quadrados Intervalar, 82
Método
  De Convexificação, 27
  De Gauss Seidel, 31
    Estendido, 31
    Não Estendido, 31
  De Gauss Seidel Intervalar, 29
  De Hansen e Greenberg, 32
  De Krawczyk, 40, 83
  De Newton Intervalar, 28, 77
Métodos
  De Convexificação, 77
Maple, 6
Matlab, 5, 6, 24
Modo De Arredondamento, 6, 18, 19
Modo de Arredondamento, 22
Multiplicadores De Lagrange, 79
Mupad, 6

Números de Máquina, 19

Operações Aritméticas
  Diferença, 10
  Potência, 11
  Produto, 10
  Razão, 10
  Soma, 10
Operador
  De Givens, 83
  De Householder, 83

Propriedade de Inclusão, *veja* Propriedade de Inclusão Monotônica
Python, 6

Qualificação Das Restrições, 77, 78

Rump, 6

Subdistributiva, 16, 22

Teorema
De Moore, 38
Existência, 39
Unicidade, 40
Do Valor Médio Intervalar, 29
Fundamental Da Aritmética Intervalar, 14

Vetor Intervalar, 13