UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS FÍSICAS E MATEMÁTICAS

O Ambiente CUTE para Problemas de Otimização

TRABALHO DE CONCLUSÃO DE CURSO

Juliana Araujo Paz

FLORIANÓPOLIS - SC
DEZEMBRO - 2004

Juliana Araujo Paz

O Ambiente CUTE para Problemas de Otimização

Trabalho de Conclusão de Curso apresentado ao
Curso de Matemática - Habilitação Bacharelado
Departamento de Matemática
Centro de Ciências Físicas e Matemáticas
Universidade Federal de Santa Catarina

Orientador: Mário César Zambaldi

FLORIANÓPOLIS - SC
DEZEMBRO - 2004

O Ambiente CUTE para Problemas de Otimização

Esta monografia foi julgada adequada como TRABALHO DE CONCLUSÃO DE CURSO no Curso de Matemática - Habilitação Bacharelado e aprovada em sua forma final pela Banca Examinadora designada pela Portaria nº 79/SCG/04.

Profª Carmem Suzane Comitre Gimenez
Professora responsável pela disciplina

Banca Examinadora:

Prof Mário César Zambaldi
Orientador

Prof Daniel Noberto Kozakevich

Prof Márcio Rodolfo Fernandes

Florianópolis, 7 de dezembro de 2004.

Sumário

Introdução

O CUTE, Constrained and Unconstrained Testing Environment, desenvolvido por N. I. M. Gould, (Computational Science and Engineering Department - England), D. Orban (CERFACS, Parallel Algorithms Project - France) e Ph. L. Toint (Facultés Universitaires Notre-Dame de la Paix - Belgium) é um ambiente de testes para problemas de otimização.

Durante a criação de um software, é inevitável uma certa preocupação dos próprios autores com a validação do mesmo. Deste modo, é importante considerar como obter e especificar uma coleção de problemas-teste adequada. É de se presumir também que pesquisadores gostariam de comparar diferentes algoritmos para um certo problema. Muitas destas situações ocorreram durante as próprias pesquisas dos autores do CUTE e muitas facilidades foram originalmente produzidas e testadas em conjunção com o LANCELOT[5], um pacote Fortran para Otimização Não Linear para problemas em grande escala. Desde 1993, o CUTE tem sido muito usado por autores de softwares de otimização.

Recentemente, surgiu uma nova versão do CUTE, denominada CUTEr, em que foram incluídas novas ferramentas e novas interfaces. Trata-se de uma revisão do projeto inicial. O CUTEr é um ambiente versátil de testes para otimização e pacotes de álgebra linear e contém uma coleção de problemas-teste, ferramentas do Matlab, etc, com a intenção de auxiliar autores de softwares, comparar e melhorar os novos pacotes de otimização e os já existentes. As novas características incluem uma reorganização do ambiente permitindo simultânea instalação em multi-plataformas, novas ferramentas, um conjunto de novas interfaces para pacotes de otimização adicionais, além de pacotes de otimização, algum suporte para Fortran 90/95 e um procedimento simplificado e automático de instalação.

Os problemas-teste fornecidos são escritos no chamado Standard Input Format

(SIF)[4] exigido pelo LANCELOT[5]. Um decodificador é fornecido para converter os arquivos em formato SIF em subrotinas com o problema decodificado. Uma vez traduzidos, estes arquivos podem ser manipulados para produzir ferramentas adequadas para testes de pacotes de otimização. Interfaces prontas para pacotes existentes também são fornecidas. O decodificador de arquivos em formato SIF, que é para ser usado juntamente com o ambiente, reside em um pacote isolado chamado SifDec por várias razões. Entre elas está a fácil manutenção e o fato que o decodificador pode se desenvolver separadamente. O CUTEr pode ser utilizado em uma variedade de plataformas UNIX, incluindo LINUX e foi desenvolvido para ser acessível e facilmente manipulado em redes de computadores heterogêneas.

Um problema de otimização começa com um conjunto de variáveis independentes ou parâmetros e frequentemente inclui condições que definem valores aceitáveis das variáveis. Tais condições são chamadas de *restrições* do problema. Outro componente de um problema de otimização é o que chamamos de função objetivo, que depende de algum modo das variáveis. A solução do problema de otimização é um conjunto de valores na qual a função objetivo assume um valor "ótimo". Matematicamente, um problema de otimização geralmente envolve maximização ou minimização. A forma geral de um problema de otimização pode ser expressa do seguinte modo:

$$\min_{x \in \mathbb{R}^n} f(x)$$

sujeito às restrições

$$\begin{cases} \text{c}_i\text{(x)=0, i=1,2,..., m} \\ \text{c}_i\text{(x)} \geqslant 0\text{, i=m+1,..., m'} \end{cases}$$

Alguns casos particulares do problema acima e que são muito relevantes são:

1. Minimização irrestrita:

   $$\min_{x \in \mathbb{R}} f(x)$$

   em que f pode ser uma função linear ou não linear;

2. Programação linear:

   $$\min_{x \in \mathbb{R}^n} c^T x$$

   sujeito à restrição

   $$Ax = b$$

3. Programação quadrática:

$$\min_{x \in \mathbb{R}^n} q(x)$$

sujeito à restrição

$$Ax \geqslant 0$$

em que q é uma função quadrática;

Em geral, para cada tipo de problema específico (função objetivo, tipo de restrições) emprega-se técnicas e algorítmos também específicos para resolvê-los.

Os problemas de otimização aparecem em muitas áreas e aplicações, como engenharias, economia, física, estatística, etc. É importante sabermos avaliar a eficiência de um dado método. O ambiente CUTEr foi criado nesse sentido e neste trabalho vamos mostrar como o uso deste ambiente pode nos ajudar a testar métodos de otimização.

O trabalho apresenta-se como segue: o primeito capítulo fornece uma breve introdução ao formato de entrada de dados padrão (SIF), com a construção de alguns arquivos e exemplos dos mesmos. Além disso, faremos uma introdução também ao decodificador destes arquivos, que tem papel fundamental no utilização do ambiente. O segundo capítulo apresenta o modo de classificação dos arquivos SIF e descrição das ferramentas fornecidas pelo CUTEr. No capítulo 4 estão todas as informações de uso e instalação do CUTEr e o modo de obtenção do pacote. O capítulo seguinte contém um exemplo da utilização do ambiente. Neste capítulo está a implementação do método de Newton para minimização irrestrita e alguns testes realizados.

Esperamos que este trabalho possa ser muito útil a todos aqueles interessados na análise e desenvolvimento de métodos numéricos em otimização assim como na utilização de um ambiente computacional especialmente criado para esta finalidade.

Capítulo 1

O SIF e o SifDec

1.1 Introdução ao Formato SIF

A modelagem matemática de muitas aplicações reais envolve a maximização ou a minimização de uma função de parâmetros conhecidos ou variáveis. Frequentemente estes parâmetros tem limites conhecidos e algumas vezes existem mais relações gerais entre esses parâmetros. Quando o número de variáveis de um problema é modesto, digamos 10 variáveis, a entrada de dados para tal problema é geralmente direta. Infelizmente, muitas áreas de aplicação exigem a solução de um problema de otimização com milhares de variáveis. Neste caso, a entrada de dados consome muito tempo e está propensa a erros. Além disso, muitos algoritmos estão sendo desenvolvidos agora para resolver problemas desta escala.

O formato que descreveremos aqui foi motivado diretamente pelas dificuldades que os autores do LANCELOT enfrentaram quando efetuando testes do pacote. Disto, fez-se necessária a criação de um formato de entrada de dados padrão para problemas de programação não linear: o Standard Input Format (SIF)[2].

Existem muitas razões para o propósito de um formato de entrada de dados padrão. Um deles é a consistência em codificar problemas de programação não linear e o melhoramento resultante da confiabilidade em códigos. Como todo problema é tratado de um modo similar e padronizado, é muito difícil negligenciar certos aspectos da definição do problema. Outra vantagem de ter um formato padrão é a possibilidade de ter um leque de problemas significativos e contar com o crescimento do mesmo. Claramente, o formato SIF deve cobrir uma grande parte de problemas práticos de otimização que os

usuários podem querer especificar. Esta especificação não deve ser feita somente para problemas irrestritos mas também para problemas com restrições de diferentes tipos e complexidade: limites das variáveis, equações lineares ou não lineares e desigualdades devem ser manuseadas sem problemas. Uma estrutura especial do problema é parte obrigatória de um arquivo SIF. Por exemplo, a esparsidade de uma matriz deve ser incluída na descrição do problema quando ela é conhecida.

A existência e o sucesso de um formato para entrada de dados padrão para problemas de programação linear, o formato MPS (Mathematical Programming System)[9], pode ser considerada como base para qualquer tentativa de se definir um SIF para problemas não lineares. O número de problemas disponíveis neste formato é grande, e muitos problemas não lineares aparecem como um refinamento dos lineares existentes, e espera-se que a parte linear seja descrita em formato MPS[9]. Por isso, é razoável exigir que um problema descrito em SIF tenha conformidade com um em formato MPS[9].

O SIF não pode ser dependente de um sistema operacional específico. Com respeito a isso, ele deve evitar fiar-se a ferramentas que podem ser excelentes mas muito específicas. Em princípio, o SIF também não pode ser dependente de uma linguagem de programação particular. Entretanto, como a intenção é que o SIF possa ser convertido dentro de programas executáveis, restrições sobre nomes simbólicos permitidas por diferentes linguagens podem influenciar a escolha de nomes dentro da descrição do próprio problema.

Adiante daremos mais detalhes do modo que tal estrutura pode ser expressa em um formato de entrada de dados padrão.

1.1.1 Começando com o SIF

Vamos nesta seção discutir alguns pré-requisitos sobre o modo como palavras e números são entendidos dentro da sintaxe do SIF.

Quando especificamos um problema em SIF, fazemos isso escrevendo um ou mais arquivos. Vamos começar considerando o promeiro arquivo, o Standard Data Input Format (ou SDIF). O SDIF contém um número de seções ordenadas usando cinco tipos de objetos: palavras-chave, códigos, números, nomes e nomes Fortran.

- As *palavras-chave* são os títulos das diferentes seções que juntas constituem o problema SDIF;

- Os *códigos* consistem de uma ou duas letras maisúsculas. O propósito é especificar, dentro das seções, vários tipos de informação do problema;

- *Números* consistem de no máximo doze caracteres que podem incluir um ponto decimal ou um sinal (um número é assumido ser positivo a menos que o sinal (-) seja dado). O valor pode ser seguido por um expoente decimal, escrito como um E ou D seguido ou não por um sinal ou dois dígitos inteiros. Espaços em branco não são permitidos entre números;

- Os *nomes* são feitos de caracteres válidos. Eles incluem caracteres alfabéticos, os dígitos de 0 a 9, caracteres em branco e outros símbolos gramaticais e matemáticos. O comprimento para um nome varia entre 6 e 10 caracteres. Por exemplo, "NAME", "3.141592" e "@$#+*" são nomes permitidos. Comumente escolhemos nomes que significam algo para nós e que possam nos ajudar a entender as relações entre as partes do problema;

- A interface que conecta o SIF e o LANCELOT produzem subrotinas Fortran que são subsequentemente usadas pelos otimizadores do LANCELOT. Estas rotinas contém variáveis que são nomeadas dentro da especificação do problema em SIF. Por causa disto, elas não aparecem somente no arquivo SIF mas também em subrotinas Fortran; estes nomes são chamados *nomes Fortran*. Eles estão restritos a conter no máximo 6 caracteres e não devem começar com um dígito.

Também podemos inserir comentários dentro da descrição do problema. Isto pode ser obtido começando a linha com um * e qualquer coisa que o segue será tido como comentário. Códigos, nomes e números devem estar bem definidos em *campos* dentro da linha. A definição de 6 campos pode ocorrer como segue:

	Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
colunas	2-3	5-12	15-22	25-36	40-47	50-61

Os campos contém diferentes tipos de informações:

- o campo 1 contém um código;

- os campos 2, 3 e 5 contém names;

- e os campos 4 e 6 contém números.

- Esquematicamente, uma linha do SDIF é organizada como:

Campo 1	Campo 2	Campo 3	Campo 4	Campo 5	Campo 6
código	nome	nome	número	nome	número

Obs.: Por convenção, as *palavras-chave* devem começar na coluna 1.

1.1.2 Exemplos

Podemos começar agora a especificar nosso primeiro problema. Começaremos com um exemplo simples: encontrar o menor número real não negativo, ou, na forma matemática

$$\min_{x \in \mathbb{R}} x$$

sujeito à restrição $x \geqslant 0$.Vamos ver como podemos especificar este problema em SDIF.

A primeira coisa que devemos fazer é definir um nome para este problema, "SIMPLES" por exemplo. Então fornecemos o seu nome escrevendo

NAME SIMPLES

Nesta linha, a palavra-chave NAME começa na primeira coluna e o nome deve estar no campo 3. Temos que especificar também a variável do problema. Fazemos isto na seção VARIABLES. Como dito anteriormente, o SIDF está dividido em seções e cada uma tem seu propósito bem definido. Depois de especificarmos a variável, nossa descrição do problema se torna:

NAME SIMPLES

* O mais simples problema de programação linear

VARIABLES

x

Devemos definir agora a função objetivo, que é simplesmente 1.0x. Então, completando nossa especificação:

NAME SIMPLES

* O mais simples problema de programação linear

VARIABLES

x

GROUPS

XN Objetivo x 1.0

```
ENDATA
```

Como antes, inicamos mais uma seção chamada GROUPS e seu propósito é declarar a função objetivo. Isto é feito usando o código XN, que por ser um código, deve estar na coluna 2. Nós dizemos então que a variável x aparece na função objetivo (que chamamos de Objetivo) com coeficiente igual a 1.0. Note que 1.0 aparecerá no campo 4, que é um dos campos numéricos. A palavra-chave ENDATA inidica o final do arquivo SDIF.

Poderíamos agora nos perguntar: com um pacote de otimização, como por exemplo o LANCELOT, entenderia que a variável x deve ser não negativa, se não a especificamos em nenhum lugar? Na verdade, por convenção, o SDIF assume que todas as variáveis são não negativas, a menos que declarado o contrário. Este tipo de informação também aparece em outra seção chamada BOUNDS, e então nosso problema poderia ser especificado como segue:

```
NAME              SIMPLES
* O mais simples problema de programação linear
VARIABLES
              x
GROUPS
XN Objetivo     x              1.0
BOUNDS
XL Simples      x               0.0
ENDATA
```

Adicionamos aqui a seção BOUNDS que especifica que a variável x tem limite inferior 0, usando o código XL. Outros tipos de limites podem ser impostos sobre uma variável. Veja na Tabela 1.1.

Existe outra parte do problema que evitamos acima, mas que pode ser vital para especificação de problemas não lineares: a especificação de um ponto inicial para minimização. Se nada for dito explicitamente, o SDIF assume que o ponto inicial é a origem. Podemos então introduzir um valor diferente de ponto inicial para x (usando o código XV, em que V é de Variável, numa seção chamada START POINT).

Código	Significado
XU	limitado inferiormente pelo valor...
XL	limitado superiormente por ...
XP	ilimitado superiormente ($+\infty$)
XM	ilimitado inferiormente ($-\infty$)
XR	ilimitado (variável livre)
XX	limitado superior e inferiormente pelo mesmo valor

Tabela 1.1: Códigos para limitantes das variáveis do problema

Nosso arquivo SIDF então se torna assim:

```
NAME                SIMPLES
* O mais simples problema de programação linear
VARIABLES
        x
GROUPS
XN Objetivo      x              1.0
START POINT
XV Simples       x              3.141592
ENDATA
```

Vamos analisar agora problemas com restrições um pouco mais complexas. Consideremos o problema:

$$\min_{x,y\in\mathbb{R}} x + 1$$

sujeito às restrições

$$\begin{cases} x \geqslant 0 \\ x + 2y = 2 \end{cases}$$

Este problema pode ser traduzido para

```
NAME            EXTRASIM
* Um programa linear simples
VARIABLES
    x
    y
```

```
GROUPS
   XN Objetivo          x          1.0
   XE Restr1            x          1.0          y          2.0
CONSTANTS
        EXTRASIM     Objetivo     -1.0
        EXTRASIM     Restr1       2.0
   XR                y
ENDATA
```

A novidade na especificação deste problema é o código XE na seção GROUPS que especifica as restrições de igualdade. O código XE é deve ser seguido pelo nome na restrição, que aqui chamamos Restr1, e pelas variáveis da restrição juntamente com seus coeficientes. O lado direito da restrição também foi especificado na seção CONSTANTS, declarando o valor da constante associada a restrição. A diferença entre os campos 2 e os campos 3 e 5 é que o primeiro é usado para indicar nomes do objeto que estão sendo definidos na afirmação corrente, enquanto que os últimos contém nomes que já foram definidos anteriormente. Outra novidade é que adicionamos a constante 1 na função objetivo. Esta constante foi especificada do seguinte modo:

```
EXTRASIM     Objetivo     -1.0
```

É importante notar que a função objetivo não tem propriamente um lado direito; se a função objetivo tivesse, a constante poderia ser colocada no lado esquerdo, o que implica a mudança do sinal. Por esta razão, na descrição do arquivo foi colocado -1.0 no lugar de 1.0.

Existem outros códigos para especificar os tipos de restrições de um problema. Veja na Tabela 1.2:

Código	**Significado**		
XN	função objetivo - constante		
XL	restrição	$\leqslant$	constante
XE	restrição	$=$	constante
XG	restrição	$\geqslant$	constante

Tabela 1.2: Códigos para especificação das restrições do problema

Resumindo: na descrição de um problema em SDIF, *palavras-chave* delimitam as seções, *códigos* indicam todo tipo de informação que são relevantes para uma dada

seção, *nomes* podem ser dados para variáveis, função objetivo e restrições, e *números*. Até aqui introduzimos as seções:

NAME: define o nome do problema;

GROUPS: começa a seção em que a função objetivo e as restrições recebem um nome e em que a contribuição linear de cada variável é especificada.

CONSTANTS: começa a seção em que as constantes são nomeadas e definidas. Existem constantes na função objetivo e no lado direito das restrições (o zero no lado direito é assumido a menos que outro valor seja declarado).

BOUNDS: começa a seção em que os limites das variáveis são nomeados e definidos (o limite inferior igual a zero é assumido se nada for declarado).

START POINT: inicia a seção em que o ponto inicial proposto para o problema é nomeado e especificado usando o código XV.

ENDATA: declara o fim do arquivo SDIF para o problema corrente.

Vamos agora considerar um problema não linear muito simples: minimizar o quadrado da diferença entre duas variáveis não negativas x e y sujeito a restrição de igualdade em que x e y somados resulta em 1.0. Formalmente, o problema pode ser escrito como

$$\min_{x,y\in\mathbb{R}} (x-y)^2$$

sujeito a

$$\begin{cases} x+y=1.0 \\ x \geqslant 0 \\ y \geqslant 0 \end{cases}$$

Se assumirmos por um instante que o quadrado não está presente na função objetivo e usando o conhecimento de SIDF para programas lineares, poderíamos escrever a especificação do problemas do seguinte modo:

```
NAME          WRONG
VARIABLES
    x
    y
GROUPS
 XN  Objetivo      x        1.0      y       -1.0
 XE  Constr        x        1.0      y        1.0
```

```
CONSTANTS
   X[1]  WRONG              Constr        1.0
ENDATA
```

Infelizmente não podemos esquecer do quadrado na função objetivo. A idéia agora é especificar uma transformação não linear do grupo objetivo. Tais transformações são chamadas *transformações do grupo* ou *funções do grupo* no SIF. Cada transformação é dita ser de um *tipo de grupo* específico e todos os tipos de grupo são introduzidos em uma seção especial chamada GROUP TYPE. Para o nosso problema, precisamos de uma seção GROUP TYPE da forma

```
GROUP TYPE
   GV    SQUARE             ALPHA
```

em que o primeiro uso da palavra-chave GROUP TYPE indica o início da nova seção, e o código GV especifica o tipo do grupo, chamado SQUARE, que envolve a variável do grupo (**G**roup **V**ariable), chamada ALPHA, que é a variável necessária para especificar a transformação do grupo (aqui, a transformação é α^2). É claro que ainda temos que dizer que o grupo objetivo é do tipo SQUARE, enquanto que a restrição linear do problema Constr é do tipo usual, chamado tipo trivial O tipo trivial usa a transformação identidade da variável do grupo, daí o nome trivial. Determinar tipos para grupos é o propósito de uma nova seção, chamda GROUP USES, que pode ser escrita como segue.

```
GROUP USES
   XT    Objetivo           SQUARE
```

Temos especificado então que o grupo objetivo é do tipo SQUARE, usando o código XT. Não determinamos nenhum tipo para o grupo Constr, então o tipo trivial é assumido ($g(\alpha) = \alpha$). Ainda temos que definir qual o significado que queremos dar quando dizemos que o tipo do grupo SQUARE da variável do grupo ALPHA é igual, em nosso exemplo, ao quadrado de tal variável. Também temos que especificar as derivadas da função associada em uma forma que ela possa ser convertida para subrotinas (Fortran). A especificação da forma funcional atual e das derivadas das funções do grupo não linear é feita em outro arquivo do nosso arquivo SIF: o SGIF (Standard Group

[1]Em geral, o SDIF usará códigos começando com X quando o valor associado com a ação especificada pelo código é encontrada em um dos campos numéricos 4 ou 6. Códigos começando com Z serão usados quando este valor é daquele parâmetro cujo nome aparece no campo de nomes 5.

Input Format). O SGIF começa com a palavra-chave GROUPS seguida do nome do problema (no campo 3), ou seja,

```
GROUPS          TAME
```

Vamos especificar agora os grupos não triviais individuais do nosso problema na seção do SGIF chamada INDIVIDUALS como abaixo:

```
INDIVIDUALS
   T      SQUARE
   F                              ALPHA**2
   G                              2.0*ALPHA
   H                              2.0
ENDATA
```

O código T indica o tipo do grupo que estamos interessados em escrever: no nosso caso o tipo é SQUARE ($g(\alpha) = \alpha^2$) . A expressão funcional para o grupo é então especificada na linha começando com o código F (como em **F**unção). Seu gradiente com relação a variável é especificado na linha começando com o código G (como em **G**radiente). Suas derivadas segundas são especificadas na linha começando com o código H (como em **H**essiana). Como em SDIF, o SGIF é finalizado com a palavra-chave ENDATA. Juntando tudo que temos até agora, podemos obter o seguinte arquivo de especificação do problema:

```
NAME            TAME
VARIABLES
        x
        y
GROUPS
   XN  Objetivo              x          1.0          y         -1.0
   XE  Constr                x          1.0          y          1.0
CONSTANTS
   X    TAME                 Constr     1.0
GROUP TYPE
   GV     SQUARE             ALPHA
GROUP USES
   XT     Objetivo           SQUARE
```

```
ENDATA

GROUPS          TAME
INDIVIDUALS
   T      SQUARE
   F                                  ALPHA**2
   G                                  2.0*ALPHA
   H                                  2.0
ENDATA
```

Agora estamos prontos para introduzir problemas não tão suaves, ou seja, problemas envolvendo mais sérias não linearidades. Em geral, o uso de funções de um grupo somente não será suficiente para para cobrir os casos mais complexos. Vamos precisar de maneiras adicionais para especificar componentes não lineares de uma função objetivo e suas restrições.

Começaremos a introdução dos conceitos necessários com a ajuda de um exemplo. Escolhemos então um problema clássico de minimização irrestrita de duas variáveis: a famosa função de Rosenbrock, ou comumente conhecida, função "banana". O problema é o seguinte:

$$\min_{x,y \in \mathbb{R}} 100(y - x^2)^2 + (x - 1)^2$$

A função é às vezes chamada de "vale das bananas" porque o desenho de suas curvas de nível revela vales curvados estreitos, cuja base tem uma curvatura suave, resultando assim curvas que tomam uma forma parecida com bananas. Especificando o problemas com técnicas que aprendemos até agora, obtemos:

```
NAME            ROSENBR
* Primeira tentativa de especificação da função de Rosenbrock
VARIABLES
        x
        y
GROUPS
   XN  Obj1                y          1.0
   XN  Obj1                'SCALE'    0.01
   XN  Obj2                x          1.0
CONSTANTS
```

```
    X    ROSENBR          Obj2          1.0
BOUNDS
    XR   ROSENBR          x
    XR   ROSENBR          y
GROUP TYPE
    T    SQUARE           ALPHA
GROUP USES
    T    Obj1             SQUARE
    T    Obj2             SQUARE
ENDATA

GROUPS          ROSENBR
INDIVIDUALS
    T       SQUARE
    F                                 ALPHA * ALPHA
    G                                 ALPHA + ALPHA
    H                                 2.0
ENDATA
```

Na seção GROUPS, a linha

```
XN Obj1                 'SCALE'     0.01
```

indica que o grupo Obj1, ou seja, a expressão $(y - x^2)^2$ é dividida pelo fator 0.01, ou ainda, é multiplicada pelo coeficiente 100.

Uma função elemento não linear f_j é dita ser uma função do problema de variáveis x_j, um subconjunto sobre todas as variáveis x. Vamos supor que x_j tem n_j componentes. Então podemos considerar a função elemento não linear como sendo da forma estrutural $f_j(v_1, ..., v_{n_j})$ em que $v_1 = x_{j1}, ..., v_{n_j} = x_{jn_j}$. As variáveis elementares para a função elemento f_j são as variáveis v. Como exemplo, suponha que a primeira função elemento não linear para um problema particular seja

$$(x_{29} + x_3 - 2x_{17})e^{x_{29}-x_{17}}$$

A forma estrutural para esta função é

$$f_1(v_1, v_2, v_3) = (v_1 + v_2 - 2v_3)e^{v_1-v_3}$$

em que $v_1 = x_{29}$, $v_2 = x_3$, $v_3 = x_{17}$. Neste exemplo temos então três variáveis elementares.

Voltando à especificação do nosso problema, gostaríamos então de indicar que o grupo Obj1 contém uma função elemento não linear, ou seja, x^2. Vamos especificar o tipo de elemento em uma seção chamada ELEMENT TYPE. Mais precisamente, vamos escrever o seguinte:

```
ELEMENT TYPE
   EV    SQ              V
```

Deste modo, especificamos o tipo do elemento, chamado SQ[2], dependendo de uma única *variável elementar* V. Neste estágio, devemos definir a variável do problema que será denominada por V. Isto é feito na seção ELEMENT USES, que em nosso caso, é do seguinte modo:

```
ELEMENT USES
   T     XSQ             SQ
   ZV    XSQ             V
```

A linha começando com o código T determina o tipo de elemento SQ para o elemento nomeado XSQ, enquanto que a linha começando com o código ZV determina a variável do problema x para a variável elementar V do elemento paticular XSQ.

A próxima tarefa é dizer que o grupo Obj1 envolve o elemento XSQ, a qual é feita na seção GROUPS USES. Então temos:

```
GROUPS USES
   T     Obj1            SQUARE
   XE    Obj1            XSQ             -1.0
   T     Obj2            SQUARE
```

em que o código XE indica que determinamos a função elemento para o grupo (em nosso caso, XSQ para o grupo Obj1), depois da multiplicação pelo fator -1.0. Resta somente especificar o comportamento não linear. Isto é feito por outro arquivo para o arquivo de especificação do problema, o chamado Standard Element Input Format (SEIF), cuja sintaxe é similar àquela em SGIF. Para a função de Rosenbrock, este arquivo SEIF é como a seguir:

[2]Usamos SQ no lugar de SQUARE somente para evitar confusão entre o tipo do grupo e o tipo do elemento. O mesmo nome poderia ser usado legalmente.

```
ELEMENTS          ROSENBR
INDIVIDUALS
   T      SQ
   F                                  ALPHA * ALPHA
   G      V                           ALPHA + ALPHA
   H      V           V               2.0
ENDATA
```

Portanto, obtemos assim a completa especificação da função de Rosenbrock em arquivo SIF que é dada abaixo.

```
NAME              ROSENBR
* O famoso problema de Rosenbrock
VARIABLES
        x
        y
GROUPS
   XN  Obj1              y          1.0
   XN  Obj1              'SCALE'    0.01
   XN  Obj2              x          1.0
CONSTANTS
   X    ROSENBR          Obj2       1.0
BOUNDS
   XR  ROSENBR           x
   XR  ROSENBR           y
START POINT
   XV  ROSENBR           x          -1.2
   XV  ROSENBR           y          1.0
ELEMENT TYPE
   EV    SQ              V
ELEMENT USES
   T     XSQ             SQ
   ZV    XSQ             V
GROUP TYPE
```

```
    T     SQUARE              ALPHA
GROUP USES
    T     Obj1                SQUARE
    T     Obj2                SQUARE
ENDATA

ELEMENTS          ROSENBR
INDIVIDUALS
    T     SQ
    F                                 ALPHA * ALPHA
    G     V                           ALPHA + ALPHA
    H     V           V               2.0
ENDATA

GROUPS            ROSENBR
INDIVIDUALS
    T     SQUARE
    F                                   ALPHA * ALPHA
    G                                   ALPHA + ALPHA
    H                                   2.0
ENDATA
```

Especificamos também o ponto inicial para este problema. Note que as seções ELEMENT TYPE e ELEMENT USES imediatamente precedem as seções GROUP TYPE e GROUP USES, e também por convenção, o SEIF é escrito entre o SDIF e o SGIF. Maiores informações sobre este tipo de arquivo e como escrever arquivos SIF de funções mais complicadas, ver [4].

1.2 SifDec

SifDec é um decodificador. Ele traduz problemas-teste, escritos no chamado Standart Input Format (SIF), em arquivos Fortran 77 e arquivos de dados. Uma vez traduzidos, estes arquivos podem ser manipulados para fornecerem ferramentas adequadas para testes de pacotes de otimização. O SifDec costumava ser parte do ambiente de

teste CUTE e agora é um componente vital no ambiente de testes CUTEr, que inclui interfaces prontas para o uso para pacotes existentes.

SifDec é agora distribuído como um pacote separado por razões convenientes e para encorajar o uso do decodificador em conjunção com outros softwares e pacotes. Uma das razões é que o decodificador SIF será parte vital da segunda versão do pacote LANCELOT[6], LANCELOT-B. Outra razão é a facilidade de manutenção e consistência quando aprimorando o decodificador. Finalmente, o decodificador SIF pode se desenvolver separadamente.

1.2.1 Instalação do SifDec

Assim como o pacote CUTEr, a versão corrente de SifDec também vem na forma de um arquivo de extensão *tar* compactado através do programa gzip. Para descompactar e extrair a distribuição do SifDec deste arquivo, basta movê-lo para um novo diretório, que aqui nos referiremos por $*SIFDEC*, e lançar os comandos:

```
prompt%    gunzip sifdec.tar.gz
prompt%    tar xvf sifdec.tar
```

ou na forma mais compactada

```
prompt%    gunzip -c sifdec.tar.gz | tar xvf -
```

O script de instalação principal é install_sifdec e também iterage com um número de scripts auxiliares. Os scripts fornecidos são:

1. install_sifdec: instala um modelo de SifDec no sistema;
2. update_sifdec: atualiza arquivos de um modelo de SifDec instalado;
3. uninstall_sifdec: remove um modelo particular de instalação.

Estes scripts podem ser encontrados no diretório $*SIFDEC*/*build*/*scripts*. Os próximos passos da instalação seguem o modelo de instalação do CUTEr, descrito no Capítulo 3, exceto pela mudança da palavra CUTER para SIFDEC em todos os passos.

O SifDec pode ser obtido através da internet, acessando a página

http://cuter.rl.ac.uk/cuter-www/sifdec

e baixando o arquivo sifdec.tar.gz.

1.2.2 Estrutura dos Diretórios do SifDec

A estrutura dos diretórios do SifDec é bem parecida com a estrutura dos diretórios do CUTEr, como poderemos observar comparando as Figuras 1.1 e 3.1, exceto pela ausência de alguns subdiretórios como, por exemplo, *include* e *sif* do diretório $*CUTER/common*. O subdiretório *man* de *$SIFDEC/common* possui agora somente a pasta man1 contendo as descrições das ferramentas do SifDec e scripts das mesmas. O subdiretório *src* do diretório $*SIFDEC/common* possui apenas dois diretórios: *select* e *sifdec*, que contém o codigo-fonte do decodificador.

Outra diferença está nos subdiretórios *$MYSIFDEC/single* e *$MYCUTER/double*. Estes contém apenas dois diretórios cada, *bin* e *config*, que contém o decodificador SIF executável na precisão correspondente intalada e os arquivos de arquitetura-dependente que foram usados para construir a árvore corrente do diretório *$MYSIFDEC,* respectivamente. Veja a estrutura dos diretórios do SifDec na Figura 1.1.

1.2.3 O Comando sifdecode

Em alguns casos, é muito útil decodificar um arquivo escrito em formato SIF sem rodar um pacote de otimização ou simplesmente para checar a sintaxe do arquivo SIF. O comando sifdecode reconhece um número de opções escritas nas linhas de comando do arquivo SIF, muitas das quais só fazem sentido quando o sifdecode é usado em conjunção com um pacote de otimização. Com relação a isso, o comando sifdecode pode ser lançado independentemente do seguinte modo:

prompt% sifdecode PROBLEMA.SIF

Geralmente, lançamos este comando na pasta em que se encontra o problema em arquivo SIF que queremos decodificar. Deste modo, o problema, as subrotinas e arquivos de dados gerados pelo decodificador ficam juntos em um mesmo diretório.

O comando sifdecode também pode ser lançado com algumas opções. A sintaxe do sifdecode é similar à sintaxe das interfaces que veremos na seção 3.4.

sifdecode [-s] [-h] [-k] [-l secs] [-show]
[-debug] probname[.SIF]

Se o arquivo estiver correto, isto resulta na criação dos arquivos:

ELFUN.f, GROUPS.f, RANGE.f, OUTSDIF.d, AUTOMAT.d

contendo a decodificação do problema.

O executável principal para o decodificador deve ser encontrado em *$MYSIFDEC/precisão/bin/sifdec*, em que $MYSIFDEC é a variável do ambiente apontando para o modelo de SifDec instalado. Por isso, a instalação do SifDec deve ser feita antes do uso de qualquer interface do CUTEr.

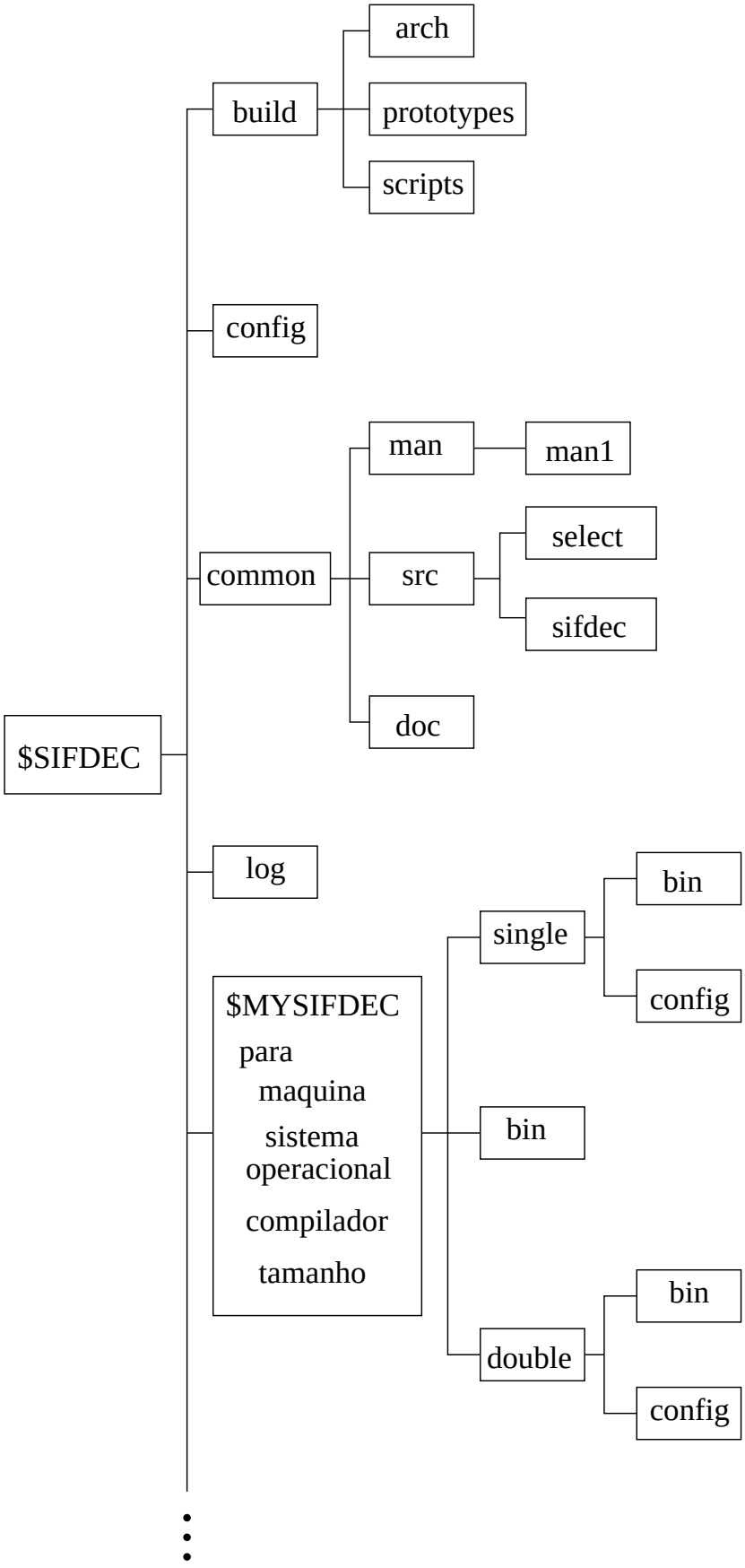

Figura 1.1: Distribuição dos arquivos do SifDec

Capítulo 2

Método de Classificação e Ferramentas

2.1 Classificação dos Arquivos SIF

Cada um dos problemas no conjunto de dados do CUTE vem com uma simples classificação inspirada pelo esquema de Hock e Schittkowski (Test Examples for Nonlinear Programming Codes, Lectures Notes in Economics and Mathematical Systems 187, Springer Verlag, 1981)[8], que era uma modesta extensão do esquema adotado por Bus (1977)[7]. O alvo de tal classificação poderia ser muito amplo de fato e percebeu-se que um esquema geral mais abrangente poderia ser muito útil para bases de dados grandes.

Um problema é classificado por uma série de caracteres

XXXr-XX-n-m

por exemplo, QUR2-AN-4-0 ou LNR0-MY-3-5.

Esta série não deve conter nenhum espaço em branco. Além disso, todas as letras na série devem ser maiúsculas. Vamos ver abaixo as letras e valores inteiros admissíveis, juntamente com sua interpretação, para cada caracter na série.

O primeiro caracter na sequência define o tipo da função objetivo. As possíveis letras são:

N	nenhuma função objetivo está definida
C	a função objetivo é constante
L	a função objetivo é linear

Q a função objetivo é quadrática

S a função objetivo é uma soma de quadrados

O a função objetivo é nenhuma das funções acima

O segundo caracter na série define os tipo de restrições do problema. As possíveis letras são:

U o problema é irrestrito

X as restrições são variáveis fixas

B as restrições são limitadas nas variávies

N as restrições representam a matriz de um grafo (linear)

L ar restrições são lineares

Q as restrições são quadráticas

O as restrições são mais gerais que qualquer uma das acima

O terceiro caracter na sequência indica a suavidade do problema. Há duas opções:

R o problema é regular

I o problema é irregular

Um problema é dito ser regular se as primeira e segunda derivadas existem e são contínuas em todo ponto.

O inteiro (r) que corresponde ao quarto caracter da sequência é o grau da maior derivada fornecida dentro da descrição do problema. Ele pode assumir apenas os valores 0, 1 ou 2.

O caracter após o primeiro hífen indica a origem principal e/ou utilidade do problema. As possíveis letras são:

A o problema é acadêmico, ou seja, foi construído especificamente por pesquisadores para testar um ou mais algorítmos.

M o problema é parte de um exercício de modelagem, onde os valores reais da solução não são usados em uma aplicação prática.

R a solução do problema é (ou tem sido) geralmente usada em aplicações reais para outros propósitos que não sejam testar algorítmos.

O próximo caracter indica quando ou não a descrição do problema contém variáveis internas explícitas. Existem dois valores possíveis:

Y a descrição do problema contém variáveis internas explícitas, ou

N a descrição do problema não contém variáveis internas explícitas.

O símbolo entre o segundo e o terceiro hífen indicam o número de variáveis no problema. Possíveis valores são:

V o número de variáveis no problema pode ser escolhido pelo usuário, ou

n um número inteiro positivo dando o número (fixado) de variáveis do problema.

O símbolo depois do terceiro hífen indica o número de restrições (exceto variáveis fixadas e limites) no problema. Note que variáveis fixadas não são consideradas como restrições gerais. As duas possibilidades são:

V o número de restrições no problema pode ser escolhido pelo usuário, ou

m um inteiro não negativo dando o número (fixado) de restrições do problema.

Vejamos alguns exemplos:

Problema	**Arquivo SIF**	**Classificação**
Engenharia		
de torsão elástica-plástica	TORSION1	QBR2-MY-V-0
otimização de grafo elétrico	HS87	OOI2-MN-6-4
de oscilação em mecânica estrutural	SVANBERG	OOR2-MN-V-V
tempo ótimo de condução de calor	HS88	QOR2-MN-2-1
Matemática Aplicada		
de geometria aplicada	PENTAGON	OLR2-AY-6-15
função banana	ROSENBR	SUR2-AN-2-0
de cálculos de autovalor	VAREIGVL	SUR2-AN-V-0
raíz de matriz quadrada	MSQRTA	NQR2-AN-V-V
de controle ótimo não linear	HAGER4	OLR2-AN-V-V
de teoria de números	LEWISPOL	QOR2-AN-6-9
de regressão ortogonal	ORTHREGA	QQR2-AN-V-V
de aproximação racional	EXPFITC	OLR2-AN-5-502
Física		
modelagem de termistores	MEYER3	SUR2-RN-3-0
de trasferência radioativa	CHANDHEQ	OQR2-AY-V-V
análise de semicondutores	SEMICON2	NOR2-AN-V-V

Tabela 2.1(a): Exemplos típicos de problemas

Problema	**Arquivo SIF**	**Classificação**
Química e Biologia		
de equilíbrio químico	HIMMELBJ	OLR2-MY-45-14
cinética química	PALMER1C	QUR2-RN-8-0
de reação química	CHEMRCTA	NOR2-MN-V-V
modelo dipolo do coração	HEART8	NOR2-MN-8-8
Economia e Pesquisa Operacional		
de equilíbrio econômico	MANNE	OOR2-MN-V-V
de geração de energia elétrica	SSEBNLN	LQR2-RN-194-96
de equilíbrio de tráfego	STEENBRE	ONR2-MY-540-126
de distribuição de água	HIMMELBI	OLR2-MN-100-12

Tabela 2.1 (b): Exemplos típicos de problemas

2.2 Ferramentas do CUTEr

O CUTEr fornece uma coleção de subrotinas que servem como ferramentas. Elas permitem uma manipulação dos dados de um determinado problema, uma vez que ele tenha sido decodificado a partir de um arquivo SIF. As ferramentas do CUTEr estão descritas nos manuais, categoria 3, e podem ser vistos usando o comando *man* ou ,no sistema LINUX, lançando o comando *less manpage*, em que manpage é manual a ser visualizado. Os manuais estão armazenados no subdiretório $\$CUTER/common/man$.

Vejamos abaixo as classes de problemas para as quais as ferramentas podem ser usadas pelo usuário para manipular os dados decodificados.

Problema irrestrito e com restrições-limite

Exitem ferramentas especificamente relacionadas ao problema com restrições-limite.

$$otimizar \quad f(x)$$

sujeito aos limites simples

$$l_i \leq x_i \leq u_i, \ (i = 1, ..., n)$$

em que $x \in \mathbb{R}^n$, f é assumida ser uma função suave de $\mathbb{R}^n$em $\mathbb{R}$. Claro que, se todos os limites são infinitos, então o problema é irrestrito.

Problema restrito geral

O CUTEr também fornece ferramentas para o problema restrito geral:

$$otimizar \quad f(x)$$

sujeito às equações gerais

$$c_i(x) = 0, i \in E$$

às restrições de desigualdade gerais

$$(c_l)_i \leq c_i(x) \leq (c_u)_i, i \in I$$

e simples limites

$$l_i \leq x_i \leq u_i, (i = 1, ...n)$$

Aqui, $x \in \mathbb{R}^n$, f, c_i são assumidas serem funções suaves de $\mathbb{R}^n$em $\mathbb{R}$. Além disso, I, E são cunjuntos de índices tais que $I \cup E = 1, 2, ...m$, com I e E disjuntos.

Associado com o problema acima está a chamada função Lagrangiana

$$L(x, \lambda) = f(x) + \lambda^T c(x)$$

em que $c(x)$ é um vetor cuja componente i é $c_i(x)$ e as componentes do vetor λ são conhecidos como multiplicadores de Lagrange.

A Tabela 3.2 contém uma lista de ferramentas para minimização irrestrita, junto com uma breve descrição. As Tabelas 3.3 (a) e 3.3 (b) descrevem as ferramentas para minimização restrita.

Ferramenta	Descrição
ubandh	extrai uma matriz associada de uma matriz Hessiana
udh	calcula a matriz Hessiana
udimen	obtém o número de variáveis envolvidas
udimse	determina o número de não zeros exigidos para armazenar a matriz Hessiana esparsa em formato elemento finito
udimsh	mesmo que udimse, em formato esparso
ueh	calcula a matriz Hessiana esparsa em formato elemento finito
ufn	calcula o valor da função
ugr	calcula o gradiente
ugrdh	calcula o gradiente e a matriz Hessiana
ugreh	calcula o gradiente e a matriz Hessiana em formato elemento finito
ugrsh	calcula o gradiente e a matriz Hessiana em formato esparso
unames	obtém os nomes dos problemas e suas variáveis
uofg	calcula o valorda função e possivelmente o gradiente
uprod	forma o produto matriz-vetor de um vetor com a matriz Hessiana
usetup	inicializa as estruturas de dados para minimização irrestrita
ush	calcula a matriz Hessiana esparsa
uvarty	determina o tipo de cada variável
ureprt	obtém estatísticas relacionadas ao cálculo de função e tempo de CPU usado

Tabela 2.2: Ferramentas para Minimização Irrestrita

Ferramenta	Descrição
ccfg	calcula valores das funções restrições e possivelmente gradientes
ccfsg	mesmo como ccfg, em formato esparso
ccifg	calcula o valor de uma única função restrição e o gradiente
ccifsg	mesmo como ccifg, em formato esparso
cdh	calcula a Hessiana do Lagrangiano
cdimen	obtém o número de variáveis e restrições envolvidas
cdimse	determina o número de não zeros para armazenar a Hessiana Lagrangiana em formato elemento finito
cdimsh	determina o número de não zeros para armazenar a Hessiana Lagrangiana, em formato esparso
cdimsj	determina o número de não zeros para armazenar a matriz de gradientes da função objetivo e restrições, em formato esparso
ceh	calcula a Hessiana Lagrangiana esparsa em formato elemento finito
cfn	calcula valores da função e restrições
cgr	calcula gradiente das restrições e gradiente da função objetivo ou da função Lagrangiana
cgrdh	mesmo que cgr, mais Hessiana Lagrangiana
cidh	calcula a Hessiana de uma função de um problema
cish	mesmo que cidh, em formato esparso
cnames	obtém os nomes do problemas e suas variáveis
cofg	calcula o valor da função e possivelmente gradiente
cprod	forma o produto matriz-vetor de um vetor com a Hessiana Lagrangiana
cscfg	calcula os valores das funções restrições e possivelmente gradientes em formato esparso
cscifg	mesmo que cscfg, para uma única restrição
csetup	inicializa as estruturas de dados para minimização irrestrita

Tabela 2.3 (a): Ferramentas para Minimização Restrita

Ferramenta	Descrição
csgr	calcula gradientes das funções restrições e objetivo/Lagrangiana
csgreh	calcula ambos os gradientes das restrições, a Hessiana Lagrangiana em formato elemento finito e o gradiente da função objetivo/Lagrangiana em formato esparso
csgrsh	mesmo que csgreh, em formato esparso no lugar do formato elemento finito
csh	calcula a Hessiana do Lagrangiano, em formato esparso
cvarty	determina o tipo de cada variável
creprt	obtém estatísticas relacionada ao cálculo da função e tempo de CPU usado

Tabela 2.3 (b): Ferramentas para Minimização Restrita

As ferramentas *creprt* e *ureprt* produzem estatísticas sobre a execução de um programa particular, como por exemplo, número vezes que o cálculo da função objetivo em um ponto foi efetuado, número de vezes que o gradiente (da função objetivo ou das restrições) em um ponto foi calculado, etc.

Vamos agora dar a completa lista de argumentos para algumas das subrotinas resumidas nas tabelas acima. Existem dois tipos de conjuntos de ferramentas: um para problemas irrestritos ou com restrições-limite, e outro para problemas com restrições gerais. Note que estes dois conjuntos não podem ser misturados. O sobrescrito i em um argumento significa que o ele é de entrada, ou seja, deve ser fornecido. O sobrescrito o significa que o argumento é de saída, isto é, será dado pela subrotina.

Subrotinas para problemas irrestritos e com restrições-limites.

- CALL USETUP(INPUT^i, IOUT^i,N^o, X^o, BL^o, BU^o, NMAX^i) : inicializa os dados para cálculos subsequentes. Note que a chamada de USETUP deve preceder as chamadas de outras subrotinas de cálculo;

- CALL UNAMES(N^i, PNAME^o, XNAMES^o) : obtém o nome do problema e de suas variáveis;

- CALL UFN(N^i, X^i, F^o) : calcula o valor da função da função objetivo;

- CALL UGR(N^i, X^i, G^o) : calcula o gradiente da função objetivo;
- CALL UOFG(N^i, X^i, F^o, G^o, GRAD^i) : calcula o valor da função objetivo e possivelmente seu gradiente. Note que chamar UOFG é mais eficiente que chamar separadamente UFN e UGR;
- CALL UDH(N^i, X^i, LH1^i, H^o) : calcula a matriz Hessiana da função objetivo (quando armazenada como uma matriz densa);
- CALL UGRDH(N^i, X^i, G^o, LH1^i, H^o) : calcula ambos o gradiente e a Hessiana da função objetivo (quando armazenada como uma matriz densa). Note que chamar UGRDH é mais eficiente que chamar separadamente UGR e UDH;
- CALL USH(N^i, X^i, NNZSH^o, LSH^i, SH^o, IRNSH^o, ICNSH^o) :calcula a matriz Hessiana da função objetivo (quando armazenada como uma matriz esparsa);
- CALL UGRSH(N^i, X^i, G^o, NNZSH^o, LSH^i, SH^o, IRNSH^o, ICNSH^o) : calcula ambos o gradiente e a matriz Hessiana da função objetivo. Note que chamar UGRSH é mais eficiente que chamar separadamente UGR e USH;
- CALL UPROD(N^i, GOTH^i, X^i, P^i, Q^o) : calcula o produto de um vetor com a matriz Hessiana;

Subrotinas para problemas com restrições gerais.

- CALL CSETUP(INPUT^i, IOUT^i, N^o, M^o, X^o, BL^o, BU^o, NMAX^i, EQUATN^o, LINEAR^o, V^o, CL^o, CU^o, MMAX^i, EFIRST^i, LFIRST^i, NVFRST^i) : inicializa a estrutura de dados para cálculos subsequentes. Note que a subrotina CSETUP deve ser chamada antes de qualquer outra subrotina para problemas com restrições gerais;
- CALL CNAMES(N^i, M^i, PNAME^o, XNAMES^o, GNAMES^o) : obtém o nome do problema, suas variáveis e restrições gerais;
- CALL CFN(N^i, M^i, X^i, F^o, LC^i, C^o) : calcula o valor da função objetivo e o valor das restrições gerais;
- CALL CGR(N^i, M^i, X^i, GRLAGF^i, LV^i, V^i, G^o, JTRANS^i, LCJAC1^i, LCJAC2$^\iota$, CJAC^o) : calcula o gradiente das funções restrições gerais;

- CALL COFG(N^i, X^i, F^o, G^o, GRAD^i) : calcula o valor da função objetivo e possivelmente seu gradiente. Note que chamar COFG é mais eficiente que chamar separadamente CFN e CGR;

- CALL CSGR(N^i, M^i, GRLAGF^i, LV^i, V^i, X^i, NNZSCJ^o, LSCJAC^i, SCJAC^o, INDVAR^o, INDFUN^o) : calcula os gradientes das funções restrições gerais (quando estas estão armazenadas em formato esparso);

- CALL CCFG(N^i, M^i, X^i, LC^i, C^o, JTRANS^i, LCJAC1^i, LCJAC2^i, CJAC^o, GRAD^i) : calcula os valores das funções restrições e possivelmente seus gradientes;

- CALL CSCFG(N^i, M^i, X^i, LC^i, C^o, NNZSCJ^o, LSCJAC^i, SCJAC^o, INDVAR^o, INDFUN^o, GRAD^i) : calcula os valores das funções restrições e possivelmente seus gradientes (quando estas estão armazenadas em formato esparso);

- CALL CDH(N^i, M^i, X^i, LV^i, V^i, LH1^i, H^o) : calcula a matriz Hessiana da Lagrangiana (quando armazenada como uma matriz densa);

- CALL CGRDH(N^i, M^i, X^i, GRLAGF^i, LV^i, V^i, G^o, JTRANS^i, LCJAC1^i, LCJAC2^i, CJAC^o, LH1^i, H^o) : calcula ambos os gradientes das funções restrições gerais e a matriz Hessiana da Lagrangiana. Note que chamar CGRDH é mais eficiente que chamar separadamente CGR e CDH;

- CALL CSH(N^i, M^i, X^i, LV^i, V^i, NNZSH^o, LSH^i, SH^o, IRNSH^o, ICNSH^o) : calcula a matriz Hessiana da Lagrangiana (quando armazenada como uma matriz esparsa);

- CALL CSGRSH(N^i, M^i, X^i, GRLAGF^i, LV^i, V^i, NNZSCJ^o, LSCJAC^i, SCJAC^o, INDVAR^o, INDFUN^o, NNZSH^o, LSH^i, SH^o, IRNSH^o, ICNSH^o) : calcula ambos os gradientes das funções restrições gerais e a matriz Hessiana da Lagrangiana. Note que chamar CSGRSH é mais eficiente que chamar separadamente CSGR e CSH;

- CALL CPROD(N^i, M^i, GOTH^i, X^i, LV^i, V^i, P^i, Q^o) : calcula o produto de um vetor com a matriz Hessiana da Lagrangiana.

Capítulo 3

Instalação e Uso do CUTEr

3.1 Procedimento de Instalação

A versão corrente do CUTEr vem na forma de um arquivo de extensão tar compactado através do programa ZIP. Para descompactar e extrair a distribuição do CUTEr, basta mover o arquivo para um novo diretório - que aqui será referenciado como $\$CUTER$ - e lançar os comandos

```
prompt%    gunzip cuter.tar.gz
prompt%    tar xvf cuter.tar
```

ou na forma mais compactada

```
prompt%    gunzip -c cuter.tar.gz | tar xvf -
```

A instalação é feita via uma interface que trabalha apenas em modo textual, na qual o usuário é induzido a fazer escolhas pertinentes a instalação desejada. O script de instalação principal é install_cuter e iterage com um número de scrips auxiliares. Vamos examinar estes scripts em breve, usando um exemplo de uma instalação do CUTEr . Os scripts dados são:

1. install_cuter: instala o CUTEr no sistema
2. update_cuter: atualiza arquivos de uma instalação já feita do CUTEr
3. unistall_cuter: remove uma instalação particular do CUTEr

Estes scripts podem ser encontrados em

$\$CUTER/build/$scripts

Vamos supor, como exemplo, que o nosso computador tem as seguintes características: um computador pessoal intel com sistema operacional LINUX, no qual o compilador instalado é o Gnu Fortran 77, g77. Assim, desejamos instalar a seguinte configuração do CUTEr nesta máquina.

Máquina	Compilador	Tamanho	Precisão
Intel	g77	medium	double

O script install_cuter serve com o propósito dual de instalação de um exemplo inicial do CUTEr no sistema e de exemplos adicionais, para uma diferente arquitetura, onde por arquitetura entendemos como sendo a combinação máquina - sistema operacional - compilador - tamanho - precisão.

Como ilustração, vamos supor que desejamos instalar um exemplo de configuração do CUTEr no diretório *$CUTER = /usr/shr/cuter/*. Descompactada a distribuição do CUTEr no diretório *$CUTER*, começamos a instalação lançando o comando

prompt% install_cuter

Uma vez lançado o comando, estamos prontos para fornecer informações relativas à instalação desejada. A primeira questão está relacionada a máquina. Neste caso selecionaremos a opção "Intel". Depois, selecionamos o sistema operacional existente na máquina. Em nosso exemplo, "LINUX". A terceira questão é em relação ao compilador para a máquina (não há garantia que algum compilador esteja instalado na máquina, simplesmente sabe-se quais são disponíveis para a combinação máquina - sistema operacional escolhida); então selecionamos "g77". As próximas escolhas são em relação a precisão das ferramentas (simples ou dupla) e o seu tamanho (pequeno, médio, grande ou padrão).

Uma vez esta informação tenha sido dada para a instalação do script, está determinado um nome de diretório padrão onde o exemplo de CUTEr será instalado. Este diretório é um subdiretório de *$CUTER* que escolhemos anteriormente (no nosso exemplo: /usr/share/*cuter*). Para o exemplo que estamos considerando, o diretório pardrão será

/usr/share/cuter/CUTEr.medium.pc.lnx.g77

que reflete as seleções feitas durante a instalação. Este diretório tem por objetivo ser auto-explicativo e pode ajudar tanto a nós quanto a outros usuários a determinar onde cada tipo de exemplo de CUTEr está armazenado. É importante notar que a precisão não aparece no nome do diretório. A razão para isto é que ambas (dupla e simples) podem ser instaladas para a mesma combinação máquina-sistema operacional-compilador-tamanho. Isto será armazenado nos diretórios *single/* e *double/* do diretório acima.

O script install_cuter criou então a estrutura de diretório necessária, *Umakefiles* e arquivos de configuração. O passo final da instalação será descrito abaixo.

Uma vez esta fase completada, install_cuter lembra o que deve ser adicionado ao *.cshrc, .bashrc* ou qualquer arquivo de configuração UNIX correspondente ao que será usado pelo usuário. No caso ao qual estamos interessados, as variáveis do ambiente CUTEr devem ser adaptadas: *CUTER* para

/usr/share/cuter

e *MYCUTER* para

/usr/share/cuter/CUTEr.medium.pc.lnx.g77

(ou o diretório alternativo especificado durate a instalação).

Vamos agora descrever o passo final da instalação usando *Umakefiles.* Exite um *Umakefile* em cada subdiretório de *$MYCUTER.* Cada um destes *Umakefiles* precisa usar apropriadamente os arquivos de configurações armazenados em *$MYCUTER/config* de forma a gerar *Makefiles* adequados para o sitema local. Este processo é chamado *bootstrapping*[3]. Isto é feito alterando *$MYCUTER* e lançando o comando

prompt% ./install_mycuter

Note que se ambas precisões, simples e dupla, foram instaladas, o script exige um argumento dizendo para que precisão o script deve desenvolver os *Umakefiles.* Mais precisamente, se o usuário deseja reproduzir a versão do CUTEr dupla precisão, o comando é

prompt% ./install_mycuter -DDoublePrecision

[3]Esta terminologia é geralmente usada para *Imakefiles* - arquivos de configuração geralmente usados para instalar o sitema X-Windows. *Umakefiles* são de fato uma versão simplificada de *Imakefiles.*

e similarmente, para a versão precisão simples

prompt% ./install_mycuter -DsinglePrecision

É importante consultar o arquivo IMPORTANT para os últimos detalhes. Um arquivo README acompanha todo *Makefile* para descrever o que ele faz e quais alvos ele reconhece. Uma vez os *Makefiles* gerados, a única coisa restante a ser feita é o usual *make all.* O comando -s para o *make* permite que apenas informações básicas da construção do processo de instalação sejam impressas na tela . Assim, podemos contruir o CUTEr usando o comando

prompt% make -s all

E isto completa a instalação do CUTEr usando *Umakefiles.*

Durante esta fase, é importante manter os olhos na tela e olhar para os indicadores [**Ok**]. A sequência de indicadores que aparecem pode ser interrompida por uma mensagem de erro. Para saber mais sobre o problema, deve-se ler o arquivo leia-me (readme) no diretório onde ocorreu o problema e tentar identificar o caminho que o comando *make* estava tentando construir, e rodar novamente o comando *make* sem a opção -s.

Um novo exemplo de CUTEr pode ser instalado, contendo uma diferente arquitetura ou com uma arquitetura correspondente a um modelo já instalado, com uma precisão ou tamanho diferente. Em todos os casos, a variável do ambiente *MYCUTER* deve apontar para o modelo corrente de CUTEr.

O script install_cuter mantém o caminho de de todos os modelos de CUTEr instalados no sistema no arquivo .log $*CUTER/log/install.log*. Este arquivo pode ser usado, por exemplo, para ter MY*CUTER* apontando para a distribuição correta. Como ilustração disto, assumindo que o modelo de CUTEr em nosso exemplo foi instalado no diretório padrão, dentro do arquivo $*CUTER/log/install.log* será encontrado

double medium Intel-like-PC lnx g77 ! $*cuter*/*CUTEr*.medium.pc.lnx.g77

em que o símbolo (!) atua como um separador. Se, na mesma máquina, estiverem instalados outros modelos de CUTEr, o arquivo .log conterá as informações destes modelos, como descrito acima.

A descrição completa do script update_cuter e do script uninstall_cuter pode ser encontrada em [3].

O CUTEr pode ser obtido através da internet. Basta acessar a página

http://cuter.rl.ac.uk/cuter-www

e baixar o arquivo cuter.tar.gz.

3.2 Estrutura dos Diretórios do CUTEr

Um dos defeitos do CUTE é que ele não foi desenvolvido para suportar simultaneamente um ambiente de multi-plataforma, que são modelos de ambientes que poderiam ser usados simultaneamente de um servidor central em várias máquinas, possivelmente diferentes, ao mesmo tempo. Além disso, usar CUTE em uma única máquina em conjunção com vários compiladores diferentes é impossível. Mais ainda, lidar com diferentes modelos de ambiente correspondentes a tamanhos diferentes de ferramentas (que é o tamanho do problema-teste que eles podem manipular) é também impossível. A razão para estas dificuldades é que a estrutura dos arquivos do CUTE, não permitia a ele mesmo tal uso, visto que ele continha apenas uma única subárvore de arquivos objetos. Se uma combinação é o conjunto máquina, sistema operacional, compilador e tamanho das ferramentas de uma arquitetura, a solução é então permitir várias dessas subárvores na instalação, uma para cada arquitetura usada.

A organização do diretório escolhida para o CUTEr é mostrada na Figura 3.1. Vamos descrever rapidamente seus componentes.

Começando do topo da figura, a primeira subárvore abaixo do diretório principal $\$CUTER$ (raíz principal do ambiente CUTEr) é *build*, que contém todos os arquivos necessários para instalação e manutenção. Seu subdiretório *arch* contém o arquivos definindo todas as possíveis arquiteturas suportadas pelo CUTEr, permitindo ao usuário instalar uma nova subárvore dependendo da arquitetura. O subdiretório *prototypes* contém as partes do ambiente que devem ser especificadas para uma arquitetura antes dela poder ser usada. Chamamos tais arquivos de *prototypes* (protótipos). Os arquivos *prototypes* incluem um número de ferramentas e scripts cuja forma final depende tipicamente das opções do compilador e do tamanho escolhido das ferramentas. O

último subdiretório de *build*, chamado *scripts* contém as ferramentas de manutenção do ambiente e vários arquivos de documentação.

A segunda subárvore abaixo de $*CUTER* é chamada *common* e contém os arquivos de dados do ambiente que são relevantes para seu objetivo, testar os pacotes de otimização, mas que são independentes da arquitetura. Seu primeiro subdiretório, *doc*, contém um número de arquivos de documentação relativas ao ambiente (tais como descrição de sua estrutura, descrição do procedimento a seguir para conectar por meio de interfaces os pacotes de otimização, o completo documento de referência para arquivos SIF, entre outros), mas não possui uma descrição das ferramentas do CUTEr e scripts das mesmas. Isto está documentado no subdiretório *man* (e como é comum nos sistemas UNIX, nos subdiretórios *man1* e *man*3). O subdiretório *src* contém um número de subdiretórios que por sua vez contém os códigos-fonte para muitas das utilidades do ambiente: tools contém os códigos-fonte das ferramentas Fortran usadas nos programas, enquanto *matlab* contém todos os m-files que fornecem uma interface MATLAB para o ambiente. O subdiretório *pkg* de *src* é usado para armazenar as informações relacionadas aos vários pacotes de otimização para os quais o CUTEr fornece uma interface. Existe um subdiretório para cada pacote (representado pelos pacotes COBYLA e VE12), tipicamente incluindo um arquivo de especificações algorítmica ou o código-fonte do pacote se disponível. O subdiretório *include* de *common* contém os cabeçalhos necessários para as interfaces entre CUTEr e códigos C. O último subdiretório de *common*, *sif*, contém alguns problemas-teste em formato SIF.

A próxima subárvore abaixo de $*CUTER* é chamada *config* e contém todos os arquivos de configurações.

O subdiretório log de $*CUTER* contém um log ("anotação de atividades ocorridas no computador"), das várias instalações (e possivelmente, subsequentes desinstalações), do ambiente para várias arquiteturas.

Os subdiretórios remanescentes de $*CUTER* são todos dependentes da arquitetura: cada um deles depende da instalação de CUTEr em uma específica máquina, para dados sistema operacional e compilador e um dado tamanho de ferrramenta. A figura representa somente um, mas os pontos de continuação na linha vertical mais a esqueda indicam que pode haver mais de uma. O nome destes diretórios são, por padrão, escolhidos automaticamente na instalação, mas o usuário de umas destas subárvores

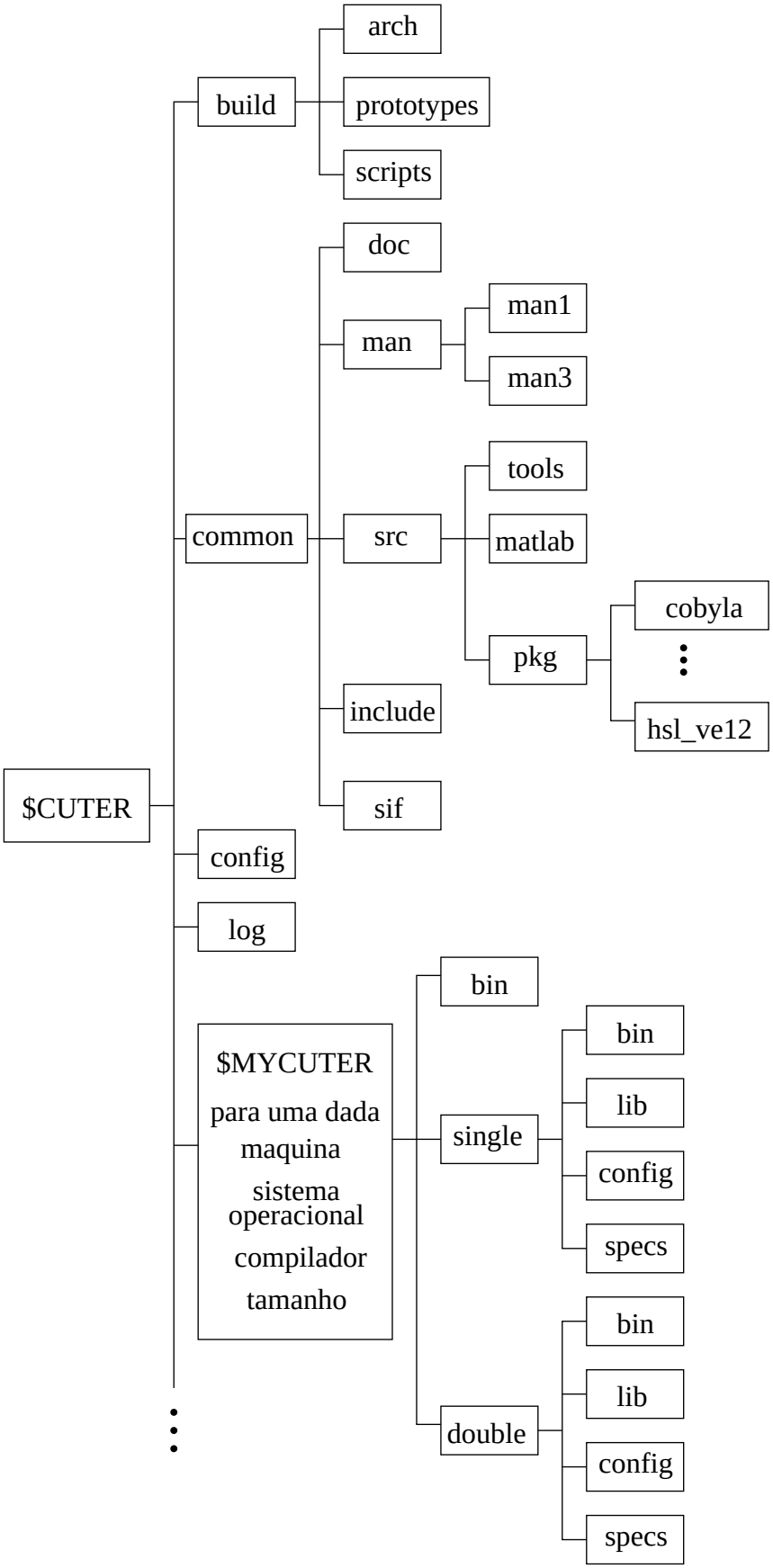

Figura 3.1: Distribuição dos arquivos do CUTEr

pode dar um nome simbólico, como *$MYCUTER*, referindo-se ao modelo de CUTEr em uso. Cada subárvore dependente da arquitetura é dividida em precisão simples e dupla (*single* e *double*, respectivamente), cada uma dessas instâncias contém quatro subdiretórios. O primeiro, *bin*, contém os arquivos objeto correspondentes aos programas principais de pacotes de otimização (drivers). O segundo, *lib*, contém a biblioteca de ferramentas do CUTE e, se relevante, livrarias associadas com pacotes de otimização conectados por meio de interface. O subdiretório *config* contém os arquivos dependentes da arquitetura que foram usados para construir a corrente subárvore *$MYCUTER* (eles são reutilizados quando uma ferramenta ou pacote de otimização é adicionado ou atualizado), enquanto que o subdiretório *specs* contém arquivos de especificação algorítmica para pacotes de otimização que são dependentes da arquitetura. Finalmente, *$MYCUTER/bin* contém aqueles scripts que são dependentes da arquitetura, mas não dependentes da precisão.

O fato das ferramentas CUTEr serem armazenadas na forma de livrarias permite um desenho mais simples de novas interfaces para pacotes de otimização, visto que as intefaces não necessitam especificar a lista exata de ferramentas que devem ser carregadas (loaded) junto com o pacote.

Uma característica final da organização do ambiente é que a documentação está disponivel via o comando usual *man* para os scripts e ferramentas.

3.3 Drivers

Drivers são programas, códigos-fonte Fortran que chamam subrotinas relevantes, de um pacote de otimização particular ou de um pacote de álgebra linear, fornecidas pelo usuário, e que obtém função, derivada e outras informações do problema diretamente das subrotinas do CUTEr. Um driver é compilado e executado pela interface de tal pacote.

Por exemplo, a distribuição do CUTEr inclui uma interface para o pacote PRAXIS, que é uma pacote para minimização irrestrita sem o uso de derivadas.. Esta interface é fornecida por dois scripts UNIX, *sdprx.pro* e *prx.pro*, armazenados em $*CUTER/build/prototypes*, que são lançados em *sdprx* e *prx* e subsequentemente armazenados em *$MYCUTER*/bin. A função destes scripts é decodificar o problema dado

em subrotinas Fortran apropriadas, junto com as livrarias necessárias e arquivos objeto, lincá-los e compilá-los e finalmente executar o driver PRAXIS, cujo código-fonte *prxma.f* é lançado e lincado gerando o arquivo objeto *prxma.o*, armazenado em *$MYCUTER/precisão*/bin. O driver inicializa todas as estruturas de dados e ambiente exigidos por PRAXIS e chama as subrotinas PRAXIS para resolver o problema dado.

Todos os pacotes são representados por um nome abreviado. Como ilustração, vamos assumir que a abreviação é *pack*. Os scripts para esta interface são chamados *sdpack.pro* e *pack.pro* e o programa principal Fortran (driver) *packma.f*.

Os pacotes não são fornecidos na distribuição do CUTEr. O usuário deve obter o pacote ou os arquivos objeto e lincá-los propriamente.

Mais detalhes relativos a um pacote específico *pack* pode ser encontrado em $*CUTER/common*/src/pkg/*pack*/README.*pack*.

3.4 Interfaces Existentes

Nesta seção vamos descrever interfaces CUTEr existentes para pacotes de otimização e de álgebra linear.

Informações e uso das diferentes interfaces para pacotes de otimização e de álgebra linear existentes podem ser encontradas nos manuais armazenadas em

$$\$CUTER/\text{commom/man/man1}.$$

O manual para o script genérico *script* pode ser visualizado lançando o comando man *script*, ou, no sistema operacional LINUX, executando o comando less *script.1*. A tabela 3.1 mostra as interfaces fornecidas e os pacotes aos quais elas correpondem.

Como ilustração, vamos considerar as interfaces *unc* e *sdunc* para o pacote de minimização irrestrita UNCMIN. As opções que podem ser lançadas com *sdunc* e *unc* são:

```
sdunc    [-s]  [-h]  [-k]  [-l secs]  [-show]
                     [-debug]  probname[.SIF]
unc      [-n]  [-h]  [-s]  [-k]  [-r]  [-l secs]  [-debug]
```

O propósito de *sdunc* é decodificar o arquivo probname.SIF, usar as variáveis do ambiente definindo o objeto e arquivos de especificação necessária para compilar o

Interface	Pacote
cob/sdcob	COBYLA (Powell)
fil/sdfil	FilterSQP (Fletcher e Leyffer)
gen/sdgen	Interface Genérica para Fortran 77
gen90/sdgen90	Interface Genérica para Fortran 90
genc/sdgenc	Interface Genérica para C/C++
hrb/sdhrb	Conversor para formato matriz esparsa SIF-Harwell ou Rutherford-Boeing (Gould)
ipopt/sdipopt	IPOPT (Wächter)
knit/sdknit	KNITRO (Byrd, Nocedal e Waltz)
la04/sdla04	LA04 (Reid)
lmb/sdlmb	L-BFGS-B (Nocedal)
lqo/sdlqo	LOQO (Benson, Shanno e Vanderbei)
mns/sdmns	MINOS (Murtagh e Saunders)
nps/sdnps	NPSOL (Gill, Murray, Saunders e Wright)
osl/sdosl	OSL (IBM)
pds/sdpds	PDS (Torczon)
prx/sdprx	PRAXIS (Brent e Chandler)
snp/sdsnp	SNOPT (Gill, Murray e Saunders)
stn/sdstn	Stenmin (Bouaricha)
tao/sdtao	TAO (Benson, Curfman McInnes, Moré e Sarich)
ten/sdten	Tenmin (Schnabel e Chow)
unc/sdunc	Uncmin (Koontz, Schnabel e Weiss)
va15/sdva15	VA15 (Nocedal)
ve09/sdve09	VE09 (Gould)
ve12/sdve12	HSL_VE12 (Gould)
ve14/sdve14	VE14 (Gould)
vf13/sdvf13	VF13 (Powell)

Tabela 3.1: Interfaces para pacotes de otimização e de álgebra linear existentes

executável UNCMIN principal. O script *unc* é similar, exceto pelo fato que ele assume que o problema já foi decodificado pelo decodificador SIF. As opções acima de *sdunc* e *unc* estão perfeitamente descritas nos manuais. Vamos revisar algumas rapidamente aqui:

-s linca, compila e executa a instância em precisão simples. Dupla precisão é o padrão;

-h imprime uma mensagem de ajuda;

-k mantém o módulo carregado antes do uso;

-r (*unc* somente) desencoraja recompilação do problema-teste;

-l secs limita o tempo de execução do CPU em *secs* segundos;

-show (*sdunc* somente) exibe possíveis configurações de parâmetro para probname[.SIF]. Outras opções são ignoradas;

-debug linca as livrarias e compila com a opção -g de forma a permitir nos livrar dos defeitos;

-n (*unc* somente) usa o módulo carregado se ele existe. O padrão é recompilar.

Os arquivos objeto principais para os pacotes suportados (isto é, neste caso, uncmin.o) devem ser colocados no diretório *$MYCUTER*/precisão/bin.

Uma breve descrição destes pacotes pode ser encontrada em

http://cuter.rl.ac.uk/cuter-www/interfaces.html.

3.5 Criando uma Nova Interface - Procedimento Geral para Fortran

O propósito desta seção é explicar como podemos construir uma interface para outro pacote de otimização, similar àquelas interfaces *sdunc* e *unc* fornecidas para o pacote UNCMIN. O CUTEr fornece os scripts genéricos *sdgen* e *gen* para este processo ser feito mais facilmente. Estes scripts podem ser encontrados no diretório $*CUTER/build/prototypes*.

Como ilustração, vamos supor que o pacote para o qual desejamos fornecer uma interface é chamado *pack*. Vamos supor também que ambas precisões (simples e dupla) do pacote estão disponíveis e que interfaces em ambas as precisões são exigidas. Uma interface para somente uma das precisões pode ser obtida ignorando qualquer

comentário relativo a outra. Então, sugerimos os seguintes passos para a criação da interface para o pacote *pack*:

1. Construa um programa principal, que aqui receberá o nome de packma.f, chamando o novo pacote e, se necessário, use as ferramentas Fortran fornecidas (para cálculo de função objetivo, seu gradiente, etc.)[4]. Os programas driver existentes no diretório $\$CUTER/common/src/tools$ podem ajudar o usuário a iniciar a escrita deste novo driver.

2. Crie um arquivo contendo **todas** as ferramentas de precisão dupla usadas pelo programa principal packma.f (as subrotinas _reprt e _names podem permanecer no programa principal); salve, por exemplo, como packd.f, em que d indica a precisão usada. Repita este processo para as ferramentas de precisão simples, substituindo o d do nome do arquivo packd.f por s. Temos assim, dois aquivos contendo ferramentas, packd.f e packs.f.

3. Crie o novo diretório *pack* em $\$CUTER/common$/src/pkg e guarde o programa principal e o arquivo contendo as subrotinas neste diretório.Agora compile o programa principal e o arquivo de subrotinas em arquivos objeto lançando o comando

   prompt% g77 -c packma.f

   prompt% g77 -c packd.f (e/ou packs.f)

4. Copie o arquivos objeto para os diretórios

   $\$MYCUTER/double$/bin e/ou \$MY$CUTER/single$/bin.

5. Agora, no diretório $\$CUTER/build/prototypes$, copie o arquivo *sdgen.pro* e *gen.pro* para *sdpack.pro* e *pack.pro* respectivamente. Isto pode ser feito do seguinte modo: no terminal, lance os comandos

   prompt% cd $\$CUTER/build/prototypes$

   prompt% cp sdgen.pro sdpack.pro

   prompt% cp gen.pro pack.pro

[4]É importante notar que os programas driver recebem nomes com a expressão *ma.f, se escritos em código fonte Fortran 77, somente para facilitar seu reconhecimento. Isto não impede que qualquer outro nome seja dado ao programa pelo usuário.

6. Neste mesmo diretório, edite os novos arquivos *sdpack.pro* e *pack.pro* e modifique-os como abaixo:

   (a) Mude o nome do pacote de *gen* para *pack*. No início dos scripts s*dpack.pro* e *pack.pro* aparecem as variáveis PAC e PACKAGE. Basta então mudar o nome do pacote *gen* em

   setenv PAC gen para setenv PAC pack
   setenv PACKAGE gen para setenv PACKAGE pack

   (b) No final do script *pack.pro* existem as variáveis PAKOBJ e SPECS, relativas a arquivos objeto/livrarias (conjunto de subrotinas) para o pacote e o arquivo de especificação, respectivamente. Nestas variáveis, o usuário deve especificar os arquivos objeto em PACKOBJ e as livrarias em SPECS do pacote que queremos criar a nova interface. Em nosso exemplo, teremos:

   PACKOBJ "packd.o" "packs.o"
   SPECS " "

pois não estamos usando nenhuma livraria além daquelas fornecidas pelo ambiente (BLAS). Os arquivos objeto devem estar no diretório $*MYCUTER/precisão*/bin, em que precisão é dupla ou simples, e o arquivo de especificação deve estar em $*CUTER/common*/src/pkg/$PACKAGE; neste caso, $*CUTER/common*/src/pkg/*pack*.

7. Os scripts precisam agora ser lançados de acordo com as dependências da máquina. Isto é feito lançando os comandos:

prompt% sed -f *$MYCUTER/precisão/config/script.sed arquivo*.pro > *$MYCUTER/bin/arquivo*

prompt% chmod a+x *$MYCUTER/bin/arquivo*

onde *arquivo* é sucessivamente *sdpack* e *pack*. Se o script *runpackage.pro* foi alterado, o mesmo deve ser feito para o *arquivo=runpackage*.

8. Agora, vá ao diretório $*CUTER/common*/src/sif, escolha um problema adequado para testar o programa e lance o comando:

prompt% pack -decode ARQUIVO.SIF

Se não houver nenhum problema adequado, o usuário pode fazer o download de problemas em formato SIF diretamente do site do CUTEr, de acordo com a classificação desejada.

O CUTEr contém um conjunto de problemas-teste para otimização linear e não linear. O conjunto completo dos problemas está disponível em um par de arquivos tar compactados (gzipped). Os problemas cujo arquivos SIF são menores que 100 Kbytes estão em um arquivo tar (mastsif_small.tar.gz; 1.2 Mbytes no total), enquanto que os problemas grandes estão contidos em outro (mastsif_large.tar.gz; aproximadamente 44.2 Mbytes no total). Uma lista de novos problemas adicionados recentemente está disponível no endereço eletrônico:

http://ftp.numerical.rl.ac.uk/pub/cuter/mastsif.updates.

Como cada problema é classificado pelo tipos de função objetivo e de restrições, é possível selecionar subconjuntos de problemas-teste de acordo com o critério pré-definido. Rolf Felkel (felkel@mathematik.tu-darmstadt.de), da Universidade de Darmstadt na Alemanha, tem fornecido on-line uma fácil seleção de problemas-teste. Isto torna possível fazer o download de todos os problemas-teste selecionados de uma vez. O endereço é http://numawww.mathematik.tu-darmstadt.de:8081/opti/select.html. Os problemas-teste também podem ser adquiridos individualmente pelo site

http://cuter.rl.ac.uk/cuter-www/Problems/mastsif.shtml

O conjunto de problemas-teste NETLIB LP é uma coleção de exemplos de programação linear da vida real dos mais variados tipos. Eles estão disponíveis em formato MPS, que é um subconjunto de formato SIF usado pelo CUTEr. O conjunto completo de problemas-teste está disponível em um arquivo de extensão tar compactado (netlib.tar.gz, aproximadamente 6.3 Mbytes no total). Também podem ser obtidos individualmente, através do endereço eletrônico

http://cuter.rl.ac.uk/cuter-www/Problems/netlib.shtml

Um conjunto de problemas-teste para programação quadrática convexa está disponível em um arquivo compactado (aproximadamente 29.6 Mbytes). Para fazer o download de problemas individualmente, basta acessar o endereço

http://cuter.rl.ac.uk/cuter-www/Problems/marmes.shtml

Capítulo 4

Implementações

Nesta seção vamos implementar o método de Newton para minimização irrestrita. O método de Newton consiste em resolver

$$\min_{x \in \mathbb{R}^n} f : \mathbb{R}^n \longrightarrow \mathbb{R}$$

para f duas vezes continuamente diferenciável. Vejamos o algoritmo do método de Newton para minimização irrestrita:

Algorítmo 1 *Dados $x^0 \in \mathbb{R}^n$ e $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ duas vezes continuamente diferenciável; para cada iteração k, resova*

$$\begin{aligned} \nabla^2 f(x^k) s^k &= -\nabla f(x^k) \\ x^{k+1} &= x^k + s^k \end{aligned}$$

$\nabla^2 f(x^k)$ denota a Hessiana da f calculada em x^k, $\nabla f(x^k)$ o gradiente da f calculado em $x^{k\cdot}$ e s^k solução do sistema.

O programa principal para o método de Newton escrito em Fortran 77 está descrito abaixo:

```
C       DRIVER PARA NEWTON IRRESTRITO
          PROGRAM NEWTONMA
C       DECLARAÇÃO DAS VARIÁVEIS
          PARAMETER(INPUT=47, IOUT=6, NMAX=1000)
          PARAMETER(ONE=1.0D+0, EPS=0.0000001)
          DOUBLE PRECISION X(NMAX), G(NMAX), F, NORMA, DNRM2, ONE
          DOUBLE PRECISION BU(NMAX), BL(NMAX), H(NMAX,NMAX)
```

```
      INTEGER N, M, K, NMAX, INPUT, IOUT
      REAL CALLS(7), CPU(2), EPS
      CHARACTER*10 PNAME, XNAMES(NMAX)
      OPEN ( INPUT, FILE = 'OUTSDIF.d', FORM = 'FORMATTED',
      .STATUS = 'OLD' )
      REWIND INPUT
C     INICIALIZACAO DOS DADOS
      CALL USETUP(INPUT, IOUT, N, X, BL, BU, NMAX)
C     NOME DO PROBLEMA E DAS VARIAVEIS
      CALL UNAMES(N, PNAME, XNAMES)
C     INICIO DAS ITERACOES
      K=0
C     CALCULO DO GRADIENTE E HESSIANA DA FUNÇAO OBJETIVO
      CALL UGRDH(N, X, G, NMAX, H)
      WRITE(*,*)' PONTO INICIAL', (X(I),I=1,N)
C     CALCULO DA NORMA DO GRADIENTE
      NORMA=DNRM2(N, G, 1)
      WRITE(*,*)'NORMA DO GRAD NO PONTO INICIAL= ',NORMA
10    IF (NORMA.GT.EPS) THEN
        DO I=1,N
          G(I)=(-1)*G(I)
        END DO
C     CALCULO DO SISTEMA HX=-G
        CALL DGESV(N, 1, H, NMAX, IPIV, G, NMAX, INFO)
C     CALCULA O NOVO PONTO DO NEWTON
        CALL DAXPY(N, ONE, G, 1, X, 1)
C     CALCULA O GRADIENTE E HESSIANA DO NOVO PONTO
        CALL UGRDH(N,X,G,NMAX,H)
        WRITE(*,*)' NOVO PONTO= ', (X(I),I=1,N)
C     CALCULO DA NORMA DO GRADIENTE
        NORMA=DNRM2(N, G, 1)
        WRITE(*,*)'NORMA DO GRADIENTE= ',NORMA
```

```
            K=K+1
            WRITE(*,*)'ITERACAO= ', K
            GO TO 10
            ENDIF
C     SOLUCAO DO PROBLEMA
      WRITE(*,*)'SOLUCAO FINAL= ', (X(I),I=1,N)
C     CALCULO DA FUNCAO NO PONTO FINAL
      CALL UFN(N, X, F)
      WRITE(*,*)'VALOR DA FUNCAO NO PONTO FINAL', F

      CLOSE(INPUT)
C     CALCULA AS ESTATISTICAS DO DRIVER NEWTONMA
      CALL UREPRT(CALLS, TIME)
      WRITE (IOUT, 2000 ) PNAME, N, EXITCODE, FINAL, OPT,
     *            CALLS(1), CALLS(2), CALLS(3), CALLS(4), CALLS(5),
     *            CALLS(6), CALLS(7), CPU(1), CPU(2)

2000  FORMAT( /, 24('*'), ' CUTEr statistics ', 24('*') //
     *       ,' Code used                :       NEWTON',
     *       ,' Problem                  :            ', A10, /
     *       ,' # variables              =              ', I10, /
     *       ,' Exit code                =              ', I10, /
     *       ,' Final f                  =       ', F15.7 /
     *       ,' OPT                      =              ', I10, /
     *       ,' # objective functions    =                ', F8.2 /
     *       ,' # objective gradients    =                ', F8.2 /
     *       ,' # objective Hessians     =                ', F8.2 /
     *       ,' Set up time              =       ', 0P, F10.2, ' seconds'/
     *       ,' Solve time               =       ', 0P, F10.2, ' seconds'/
     *       ,' 66('*') / )              =
      STOP
      END
```

Além do programa principal, que salvamos no diretório

$CUTER/common/src/pkg/newton

como newtonma.f, temos também o arquivo que nominanos de newtond.f, contendo a subrotina DGESV, uma subrotina do pacote LAPACK[6], e todas as subrotinas usadas pela mesma (lembrando que o *d* no nome do arquivo refere-se a precisão dupla).

O passo seguinte é compilar estes dois arquivos em arquivos objeto e copiá-los para o diretório*$MYCUTER/double/bin.* Depois, no diretório $*CUTER/build/prototypes* criamos os scripts *sdnewton.pro* e *newton.pro* a partir dos scripts genéricos *sdgen.pro* e *gen.pro* e editamos as variáveis destes scripts. Lançamos os comandos como descrito no passo 7 na seção 3.5 e agora estamos aptos a testar nosso método com problemas-teste fornecidos pelo ambiente.

Faremos o primeiro teste com um clássico da minimização irrestrita: minimizar a funo de Rosenbrock ou função "banana". As Figuras 4.2 e 4.3 mostram o comportamento da função e suas curvas de nível, respectivamente.

O problema está classificado como SUR2-AN-2-0, ou seja, a função objetivo é uma soma de quadrados, sem restrições, regular, possui derivada segunda, é um problema acadêmico e de duas variáveis.

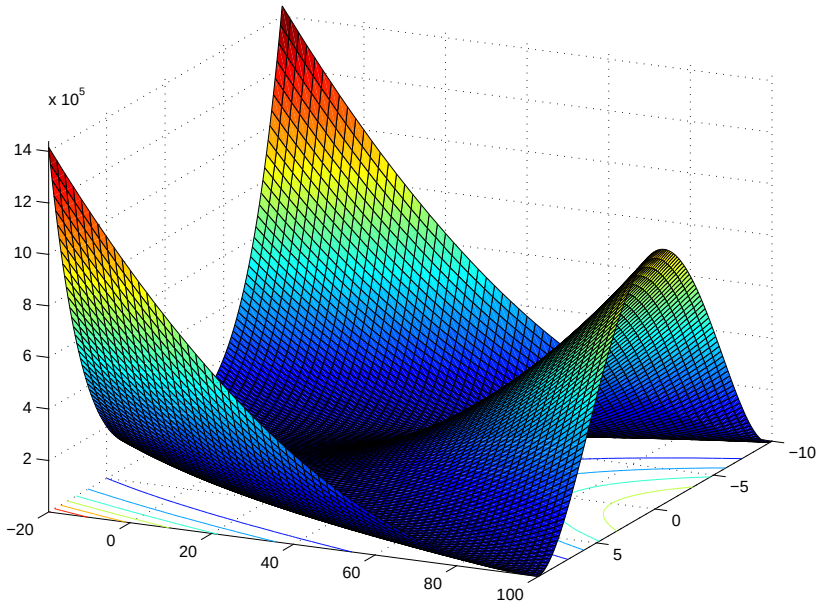

Figura 4.2: função "banana"

Então, lançando o comando

prompt% newton -decode ROSENBR.SIF

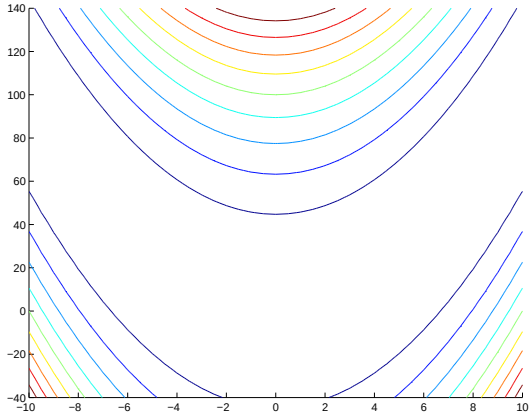

Figura 4.3: Curvas de nível da função "banana"

obtemos os seguintes resultados:

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
0	232.867688	$x_1 = -1.2 x_2 = 1$	24.2
1	4.63942621	$x_1 = -1.1752809, x_2 = 1.38067416$	4.73188433
2	1370.78985	$x_1 = 0.763114871, x_2 = -3.17503385$	1411.84518
3	0.473110379	$x_1 = 0.763429679, x_2 = 0.582824775$	0.0559655168
4	25.0274456	$x_1 = 0.999995311, x_2 = 0.944027324$	0.313189076
5	$8.60863357E-06$	$x_1 = 0.999995696, x_2 = 0.999991391$	$1.85273976E-11$
6	$8.28570579E-09$	$x_1 = 1 x_2 = 1.$	$3.43264619E-20 \simeq 0$

Resultados numéricos para o problema ROSENBR.SIF

Vejamos outro exemplo. Para o problema S308.SIF, cuja classificação é SUR2-AN-2-0, o método de Newton forneceu os seguintes resultados:

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
0	127.922215	$x_1 = 3.2 x_2 = 0.1$	87.6860481
1	38.1290555	$x_1 = 2.0382145, x_2 = 0.0295410992$	19.5657178
2	11.8751262	$x_1 = 1.356712, x_2 = 0.0236328517$	5.46351682
3	3.76337741	$x_1 = 0.717036979, x_2 = 0.246141861$	1.93681197
4	8.36421853	$x_1 = -0.437930971, x_2 = 1.46626077$	3.07930343
5	2.17005922	$x_1 = -0.258094397, x_2 = 1.02226698$	1.05580571

...

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
6	0.487567245	$x_1 = -0.183139494, x_2 = 0.797509098$	0.795114946
7	0.0655389423	$x_1 = -0.159270173, x_2 = 0.710614075$	0.773681026
8	0.00201171849	$x_1 = -0.155548813, x_2 = 0.695066958$	0.773199524
9	$2.10396262E-06$	$x_1 = -0.155437347, x_2 = 0.694564299$	0.773199056
10	$2.29834781E-12$	$x_1 = -0.155437236, x_2 = 0.694563775$	0.773199056

Resultados numéricos para o problema S308.SIF

Faremos agora alguns testes com funções quadráticas. Para isso escolhemos os problemas ZANGWIL2.SIF, DIXON3DQ.SIF e PALMER2C.SIF, que tem a seguinte classificação: QUR2-AN-2–0, QUR2-AN-10-0 e QUR2-RN-8-0, respectivamente. Os resultados obtidos foram:

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
0	2.2627417	$x_1 = 3, x_2 = 8$	-16.6
1	0	$x_1 = 4, x_2 = 9$	-18.2

Resultados numéricos para o problema ZANGWIL2.SIF

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
0	5.65685425	$x_1 = -1, x_2 = -1, x_3 = -1, x_4 = -1, x_5 = -1$ $x_6 = -1, x_7 = -1, x_8 = -1, x_9 = -1, x_{10} = -1$	8
1	0	$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1$ $x_6 = 1, x_7 = 1, x_8 = 1, x_9 = 1, x_{10} = 1$	0

Resultados numéricos para o problema DIXON3DQ.SIF

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
0	38861813	$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1,$ $x_5 = 1, x_6 = 1, x_7 = 1, x_8 = 1$	26894034.4
1	$1.74480291E-08$	$x_1 = 19.3895711, x_2 = -55.9990053$ $x_3 = 66.3248755, x_4 = -56.267078$ $x_5 = 41.016833, x_6 = -18.0397177$ $x_7 = 4.22476476, x_8 = -0.401307358$	0.0143688857

Resultados numéricos para o problema PALMER2C.SIF

Agora vamos testar nosso método para um problema cuja função objetivo não é quadrática nem soma de quadrados. Escolhemos aleatoriamente os problemas DENSCHNA.SIF e BRKMCC.SIF, classificados como OUR2-AN-2-0. Vejamos os resultados obtidos:

k	$\nabla f(x^k)$	x^k	$f(x^k)$
0	15.5562501	$x_1 = 1, x_2 = 1$	7.95249244
1	5.08250234	$x_1 = 0.496028029$ $x_2 = 0.527803798$	1.59207944
2	1.17471204	$x_1 = 0.130845867$ $x_2 = 0.164169739$	0.119158994
3	0.0782385013	$x_1 = -0.0155198542$ $x_2 = 0.0260747736$	0.000809358927
4	0.00196678938	$x_1 = -0.00100880474$ $x_2 = 0.000995309842$	$9.91811398E-07$
5	$2.96765205E-06$	$x_1 = -1.49201201E-06$ $x_2 = 1.48791453E-06$	$2.21390973E-12$
6	$6.64177165E-12$	$x_1 = -3.32091862E-12$ $x_2 = 3.32090534E-12$	$1.10283709E-23 \simeq 0$

Resultados numéricos para o problema DENSCHNA.SIF

k	$\|\|\nabla f(x^k)\|\|$	x^k	$f(x^k)$
0	24.174162	x_1= 2, x_2= 2	5.99
1	0.0183426504	x_1= 1.80153428, x_2= 1.38120027	0.169091488
2	$6.10531063E-06$	x_1= 1.79540472, x_2= 1.79540472	0.169042679
3	$5.88003336E-13$	x_1= 1.79540285, x_2= 1.37785978	0.169042679

Resultados numéricos para o problema BRKMCC.SIF

As soluções dos problemas estão no arquivo SIF dos mesmos. Basta então abrir o arquivo e verificar se a solução está correta. Para os nossos exemplos, todas as soluções obtidas coincidiram com as soluções exatas de cada problema.

Conclusão

O desenvolvimento de algoritmos para problemas de otimização é um campo de pesquisa muito relevante e promissor. A compreensão de um ambiente computacional com tantas possibilidades, como descritas neste trabalho, consiste em numa ferramenta muito útil para pesquisa de algoritmos e validação dos modelos matemáticos em diversas áreas das ciências aplicadas.

O ambiente CUTE, desenvolvido por pesquisadores de primeira linha em otimização é um instrumento computacional muito dinâmico, permitindo o desenvolvimento de interfaces ou novos métodos mátemáticos e ao mesmo tempo disponibiliza um grande número de problemas-teste para validação de algoritmos.

Este trabalho teve como objetivo principal descrever e divulgar o ambiente CUTE para a comunidade acadêmica assim como nos preparar para um futuro trabalho em otimização em nível mais avançado. Este último objetivo foi plenamente alcançado pela compreensão e utilização do ambiente CUTE, finalizando com uma implementação típica.

Referências Bibliográficas

[1] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *CUTE: Constrained and Unconstrained Testing Environmen*t. ACM Transactions on Mathematical Software, 21(1):123-160, 1995.

[2] N. I. M. Gould, D .Orban, and Ph. L. Toint. *CUTEr and SifDec: Constrained and Unconstrained Testing Environment, revisited*. ACM Transactions on Mathematical Software, 29(4):373-394, 2003.

[3] N. I. M. Gould, D .Orban, and Ph. L. Toint. *General CUTEr documentation*. Rutherford Appleton Laboratory, UK, CERFACS, France and Facultés Universitaires Notre-Dame de la Paix, Beulgium, 2003. ver http://cuter.rl.ac.uk/cuter-www/

[4] N. I. M. Gould, D .Orban, and Ph. L. Toint. *General SifDec documentation*. Rutherford Appleton Laboratory, UK, CERFACS, France and Facultés Universitaires Notre-Dame de la Paix, Beulgium, 2004. ver http://cuter.rl.ac.uk/CUTER-www/sifdec

[5] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: a Fortran package for Large-scale Nonlinear Optimization (Release A)*. Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[6] F. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, 2ed*. SIAM Society for Industrial and Applied Mathematics, University City Center, Philadelphia, Pennsylvania, 1995.

[7] J.C.P. Bus. A proposal for the classification and documentation of test problems in the field of nonlinear programming. Technical report, Mathematisch Centrum, Amsterdam, 1977.

[8] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Springer Verlag, Berlin, 1987. Lectures Notes in Economics and Mathematical Systems 282.

[9] *Program Reference Manual*. Mathematical Programming System Extended/370 (MPSX/370)