Ambientes de Programação e Interface para Problemas de Optimização

Ismael Vaz[1], Teresa Monteiro[2], Edite Fernandes[3]
Departamento de Produção e Sistemas,
Escola de Engenharia,
Universidade do Minho

Resumo

A existência de inúmeros problemas de optimização na área da engenharia justifica a existência de ambientes que permitam formular e resolver esses mesmos problemas. A modelação de determinados problemas é tão fundamental como a sua formulação no computador, por forma a que o "software" existente ou a desenvolver possa apresentar uma solução aproximada para o problema formulado. A formulação de problemas de grande dimensão e complexidade é auxiliada pela existência de ambientes de programação que permitem que um determinado utilizador não só formule e resolva o seu problema, mas também teste os seus próprios códigos. Este trabalho tem como objectivo ilustrar dois ambientes e as suas interfaces que permitem a formulação de problemas matemáticos e a construção de bibliotecas de "software" para a sua resolução.

1. Introdução

Os problemas de optimização podem ser descritos na seguinte forma geral

$$\begin{aligned} &\textit{minimizar} \quad \mathrm{f(x)} \\ \text{sujeito a} \quad &\mathrm{g}_i(\mathrm{x}) = 0 \quad \text{com} \quad i = 1,...,m_1 \\ &\mathrm{g}_i(x) \le 0 \quad \text{com} \quad i = m_1 + 1,...,m_2 . \end{aligned}$$

No conjunto das restrições de desigualdade estão incluídas restrições de limites simples do tipo: $l_k \le x_k \le u_k, k = 1,...,n$.

[1] aivaz@ci.uminho.pt
[2] tm@ci.uminho.pt
[3] emgpf@ci.uminho.pt

A natureza das funções f(x) e $g_i(x)$ indicam se o problema é linear ou não linear. Após a sua modelação num problema de optimização, como o descrito anteriormente, há necessidade de codificar o problema de optimização no computador. Dependendo do método usado para resolver o problema, assim pode ter de se calcular as seguintes funções: f(x) e g(x), respectivamente a função objectivo e o vector das funções das restrições; ∇f(x), que é o vector gradiente da função objectivo; ∇g(x), que define a matriz do jacobiano das restrições; ∇^2f(x), que define a matriz hessiana de f(x); $\nabla^2 g_i(x), i = 1,...,m_2$, que definem as matrizes hessianas das funções das restrições.

A codificação destas funções e o próprio processo de derivação podem transformar-se num processo moroso, complicado e sujeito a erros.

O desenvolvimento de pacotes de "software" que permitam ao utilizador de um determinado algoritmo codificar o seu problema modelado de uma forma fácil e sem erros torna-se crucial. A existência de vários problemas (reais ou académicos) já codificados são uma ferramenta útil para os investigadores na área da optimização (linear ou não-linear), pois permite testar e comparar os algoritmos existentes com os desenvolvidos.

No sentido de evitar confusões entre os diversos códigos de "software" serão usadas, a partir de agora, as seguintes designações:

I. *Código existente* – código que foi desenvolvido no sentido de permitir a fácil e rápida codificação do modelo. Este código providencia ferramentas que podem ser utilizadas pelo utilizador;

II. *Código do utilizador* – código escrito que resolve o problema codificado. Este código é escrito pelos investigadores da área no sentido de testarem o seu próprio algoritmo. Existem vários pacotes comerciais nesta área (MATLAB[5], NPSOL[4], LANCELOT[2], MINOS[6]), pelo que este código não será necessariamente escrito pelo utilizador;

III. *Código gerado* – código que as ferramentas geram a partir da descrição do problema.

Dois dos ambientes de programação mais conhecidos em optimização não linear são o CUTE([1]) e o AMPL([3]). Como se verá nas secções 2 e 3, estes ambientes permitem a formulação de problemas de optimização e a construção de bibliotecas de

"software". A sua utilização exige do utilizador um conhecimento profundo das correspondentes linguagem de programação e da construção de "scripts", para a "interface" entre o pacote e o *código do utilizador*.

2. CUTE

O CUTE ("Constrained and Unconstrained Teste Environment") ([1]) é um "software" de domínio público que pode ser obtido através da "Internet" no endereço ftp://joyous-gard.cc.rl.ac.uk/pub/cute ou em ftp://thales.math.fundp.ac.be/cute . O CUTE encontra-se escrito em ANSI Fortran 77. Estão disponíveis versões em precisão aritmética simples e dupla. A instalação é automática para as seguintes plataformas: CRAY UNICOS, DEC (Ultrix, VMS e OSF/1), HP-UX, IBM AIX, DOS (com o compilador de Fortran da WATCOM) e SUN SunOS. Conseguiu-se com sucesso instalar o CUTE em Linux, embora a instalação não seja automática.

2.1 Estrutura de directórios

No final da instalação do CUTE a estrutura de directórios é apresentada na figura 1.

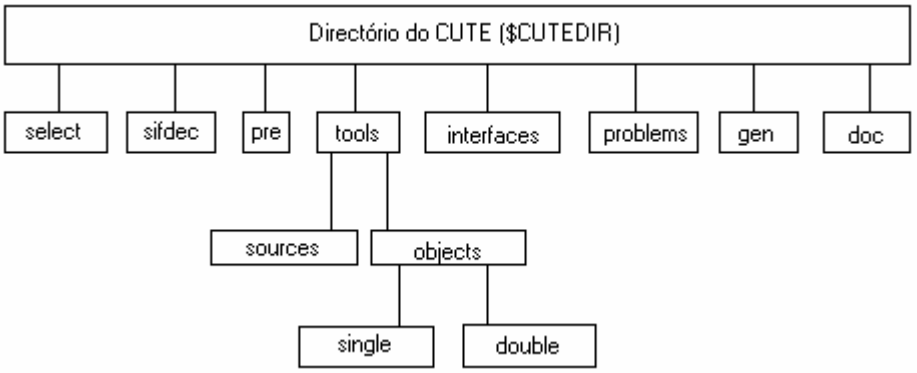

Figura 1: Estrutura de directórios do CUTE

Numa instalação típica em Linux o directório do CUTE será /usr/local/cute ($CUTEDIR=/usr/local/cute). Os subdirectórios têm os seguintes significados:

- doc – directório que contém a documentação do CUTE que acompanha a distribuição.
- gen – directório onde o utilizador colocará o seu código de resolução do problema. É da sua responsabilidade a criação de eventuais "makefiles", "scripts", etc, por

forma a que no final seja um módulo de código objecto (por exemplo gen.o). Este módulo de código deve ser independente do CUTE (não chamar as ferramentas) por forma a que seja portável. Na realidade existem mais alguns directórios deste tipo que se referem a módulos já existentes (MINOS[6], NPSOL[4], etc). Estes módulos não fazem parte da distribuição do CUTE, uma vez que estão sujeitos a direitos de autor.

- problems – directório com problemas já codificados. Existem duas bases de dados com problemas já codificados na linguagem usada pelo CUTE, "Standard Input Format" (SIF). O anexo 1 contém um exemplo de um problema em SIF. Uma base de dados contém problemas de grandes dimensões (mastsif_large) e a outra de pequenas dimensões (mastsif_small). Estas bases de dados podem ser obtidas do mesmo modo que o CUTE.
- interfaces – neste directório estão os "scripts" que procedem à compilação e geração do executável final que apresentará uma solução para o problema. É da responsabilidade do utilizador escrever estes "scripts". Já a seguir descrevemos o funcionamento da interface para o módulo genérico (gen). Os dois "scripts" existentes são **sdgen** e **gen** (note-se que o directório interfaces deve estar no "PATH" do utilizador). O "script" **sdgen** procede à verificação da linha de comandos (opções do utilizador), verifica se o ficheiro com o problema a resolver existe e procede à descodificação dos ficheiros necessários à fase seguinte. O **sdgen** termina com a execução do "script" **gen**. Este, por sua vez, procede à compilação dos ficheiros gerados pelo **sdgen**, junta todos os módulos (subrotinas do problema, programa principal, *código do utilizador* e ferramentas do CUTE) e gera o executável final (genmin). Posteriormente executa o binário gerado para resolver o problema.
- sifdec – neste directório estão alguns binários executáveis que permitem a descodificação do ficheiro que contém o problema (da linguagem SIF para a linguagem Fortran 77). A escolha do binário a executar é feita mediante as opções que o utilizador especificou (opções do "script" **sdgen**). Após a descodificação, o binário gera os ficheiros ELFUNS.f, GROUPS.f, RANGES.f, SETTYP.f, EXTERNAL.f e OUTSDIF.d. Os ficheiros de extensão .f serão posteriormente compilados e unidos ao módulo do utilizador. O ficheiro OUTSDIF.d contém os dados para o problema.

- select – directório com as ferramentas que permitem pesquisar e actualizar a base de dados dos problemas. A pesquisa é baseada no ficheiro CLASSF.DB. O executável **select** é interactivo com o utilizador por forma a que este especifique determinadas características (tipo de função objectivo, número de variáveis, etc) dos problemas que procura. O executável **select** termina imprimindo no ecrã uma lista de problemas que satisfazem as características pretendidas, existindo a possibilidade de guardar num ficheiro essa lista. Os utilitários **classify** e **classall** permitem classificar um problema (ou todos) e acrescentar a classificação no ficheiro CLASSF.DB.
- tools – este directório contém dois subdirectórios: object e source. O subdirectório object, por sua vez, contém outros dois onde residem as ferramentas já compiladas. Estas ferramentas têm duas versões, uma com aritmética de precisão simples (single) e outra de precisão dupla (double). O subdirectório source contém o código fonte (Fortran 77) das ferramentas. É neste directório que o utilizador poderá colocar a parte principal do seu módulo. Geralmente o nome deste módulo será o nome do módulo do *código do utilizador* acrescentando "ma" (por exemplo genma.f para o caso genérico). É neste pequeno módulo que o utilizador coloca a rotina principal ("main") e chama as ferramentas do CUTE. Este módulo principal deve passar como argumentos os apontadores de eventuais chamadas às ferramentas do CUTE, no sentido de que o módulo do *código do utilizador* seja independente do CUTE. O utilizador terá de garantir a compilação do código principal sempre que necessário.
- pre – os ficheiros com o código fonte das ferramentas possuem comentários que devem ser retirados de acordo com a precisão aritmética e o sistema operativo (compilador) utilizados. Neste directório residem os utilitários que mediante a precisão aritmética pretendida e o sistema operativo em causa fazem o pré-processamento do código fonte.

2.2 Ligação entre módulos

Ao "script" **gen** cabe a tarefa de juntar o código produzido pelo utilizador, as ferramentas e o código do problema. A relação entre os códigos é a descrita na figura 2.

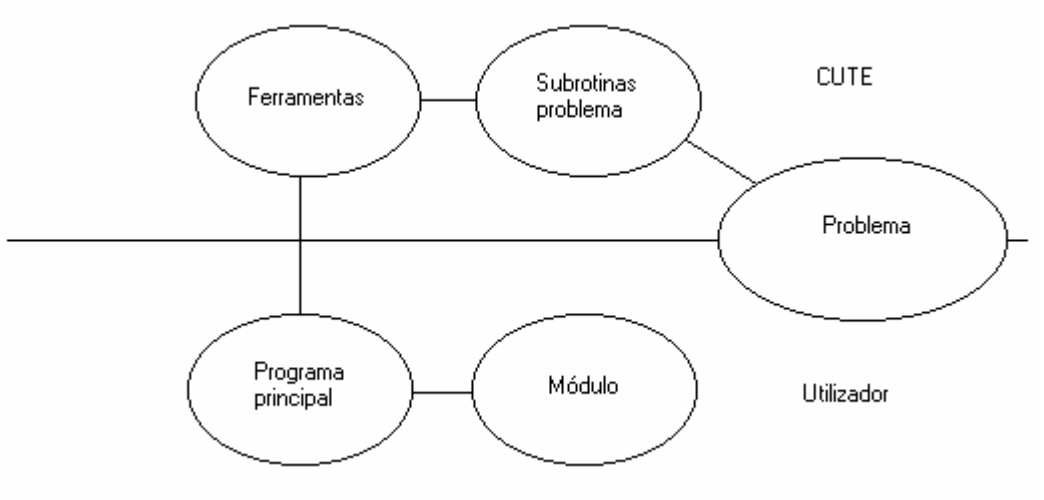

Figura 2: Relação entre códigos

2.3 Ferramentas

O CUTE possui duas classes de ferramentas. Uma para problemas com restrições e outra para problemas sem restrições. Estas ferramentas permitem avaliar, por exemplo, a função objectivo e o gradiente da função objectivo. Por questões de brevidade apenas será descrito uma subrotina para cada classe.

- *CALL UFN(N^i, X^i, F^o)* - permite avaliar a função objectivo. Nas ferramentas para a classe de problemas sem restrições os nomes das subrotinas começam sempre pela letra U ("Unconstrained"). Os argumentos com *i* são argumentos de entrada ("Input") e o argumento com *o* é de saída ("Output"). N é a dimensão do problema e X é o ponto onde se pretende avaliar a função (X é um "array" com N elementos). A subrotina retorna em F o valor da função objectivo.
- *CALL CFN(N^i, M^i, X^i, F^o, LC^i, C^o)* permite avaliar a função objectivo e as restrições. Os nomes das subrotinas para problemas com restrições começam sempre com a letra C ("Constrained"). N é a dimensão do problema, M é o número de restrições, LC é a dimensão em que o "array" C foi declarado e F é o valor da função objectivo. Note-se que a implementação em Fortran 77 obriga a que os "array" sejam declarados com valores máximos, originando três possibilidades de compilação das ferramentas –pequena, média e grande- que influenciam a pré-definição dos "arrays" e matrizes.

3. AMPL

O AMPL ("Algebraic Modeling Programming Language") ([3]) segue uma filosofia algo diferente do CUTE. O AMPL é um binário executável isolado (reside num directório à escolha do utilizador). O AMPL é um "software" comercial, sendo possível obter uma versão de demonstração limitada (máximo de 300 variáveis e 300 restrições). A partir da sua página na "Internet", cujo endereço é http://www.ampl.com, pode ser obtida a versão limitada ("Student Edition"). O binário em questão e a linguagem são ambos denominados de AMPL pelo que a referência ao binário **ampl** será em letra minúscula e a referência à linguagem AMPL será em letra maiúscula. O **ampl** permite ainda a análise das soluções do problema.

3.1 Funcionamento

O **ampl** tem um funcionamento à linha de comandos. Após a execução o **ampl** fica à espera de instruções do utilizador. A descrição da linguagem AMPL será omitida, uma vez que tornaria o texto demasiado longo. O anexo 2 contém um exemplo de um problema simples. Os seguintes comandos para o **ampl** são essenciais:

model XXX.mod; - indica ao **ampl** que o ficheiro *XXX.mod* contém uma descrição do modelo a ser usado,

data YYY.dat; - indica ao **ampl** que o ficheiro *YYY.dat* contém dados referentes ao modelo já definido,

option solver ZZZ; - indica ao **ampl** que deve executar o binário *ZZZ* para resolver o problema,

solve; - indica ao **ampl** que deve executar o binário que resolverá o problema.

Quando o **ampl** recebe instruções no sentido de resolver o problema, gera um ficheiro temporário que irá funcionar como canal de ligação entre o **ampl** e o binário que irá resolver o problema. A linha de comandos para o executável é *"ZZZ stub –AMPL"*, em que *ZZZ* representa o binário que resolve o problema, *"stub"* é o ficheiro que contém a descrição do problema e *"-AMPL"* indica ao binário *ZZZ* que este foi executado pelo **ampl** (note-se que o binário que resolve o problema é um binário

independente do **ampl**, ao contrário do CUTE em que era um módulo e não poderia ser executado directamente).

3.2 Ligação entre o ampl e o binário que resolve o problema

A ligação entre o ampl e o binário que resolve o problema é feita através do ficheiro *"stub"*. Existe um pacote de "software" (escrito na linguagem C) que contém as rotinas necessárias à leitura e processamento do ficheiro *"stub"*, bem como alguns exemplos de "interfaces" e "interfaces" para pacotes já existentes (MINOS[6], LOQO[7], NPSOL[4], MATLAB[5], etc). Este pacote pode ser obtido em ftp://netlib.bell-labs.com/netlib/ampl/solvers.tar ou em http://netlib.bell-labs.com/netlib/ampl (ficheiro solvers.tar).

Estas ferramentas são diferentes das do CUTE. No CUTE sempre que é necessário obter algum dado sobre o problema chama-se a respectiva função. No **ampl**, quando se chama uma função são preenchidas as estruturas de dados respectivas (estruturas definidas no ficheiro de "include" "asl.h").

Existe um conjunto de ferramentas que permite transformar o ficheiro *"stub"* em subrotinas (na linguagem C) que podem ser posteriormente compiladas com o *código do utilizador*.

3.3 Apresentação da solução

O binário executado escreve num ficheiro de extensão .sol a solução encontrada. Existem ferramentas que apoiam a escrita dos ficheiros que contêm as soluções. Estes ficheiros podem ser analisados pelo **ampl** através do comando *"display X;"* que apresenta no terminal o conteúdo da variável *X*.

3.4 AMPL

A linguagem AMPL é próxima da linguagem matemática. Esta proximidade permite que problemas de pequenas dimensões e complexidade sejam escritos num tempo razoável sem exigir um grande esforço de aprendizagem da linguagem. A linguagem faz distinção entre um determinado modelo e os seus dados (ficheiros .mod e .dat respectivamente), pois permite a definição de um modelo genérico e do conjunto de dados que instanciam os parâmetros declarados. Não é obrigatório a existência de dois ficheiros separados para o modelo e dados. Pode encontrar-se em [8] um conjunto de problemas já codificados.

4. Conclusões e comparações

O CUTE ([1]) e o AMPL ([3]) são dois ambientes de programação que permitem a codificação de problemas de optimização e fornecem as ferramentas necessárias para que os utilizadores possam resolver os problemas codificados, fornecendo apenas o código que resolve o problema (e algum código que serve de "interface"). Além de permitir a codificação, estes ambientes fazem a derivação automática das funções matemáticas. Este processo liberta o utilizador do seu cálculo e da possibilidade de cometer erros na derivação manual e na sua programação em computador. O quadro seguinte faz uma comparação entre o CUTE e o AMPL, evidenciando as suas potencialidades e diferenças.

	CUTE	AMPL
Linguagem de Codificação	SIF ("Standard Input Format")	AMPL ("Algebraic Modeling Programming Language")
Facilidade de Codificação	Difícil	Fácil
Base de dados de Problemas	Sim	Sim
"Sofware" existente para resolver problemas	MINOS, LANCELOT, NPSOL, MATLAB, etc	MINOS, LANCELOT, NPSOL, MATLAB, LOQO, etc
Instalação do pacote	Complexa	Simples
Linguagem de programação base	Fortran 77	C
Interface com o binário do utilizador	Fácil de usar	Mais complexa
Recompilação do problema	Sim (Código do problema e ligação com os restantes módulos)	Não

Quadro 1: Comparação entre CUTE e AMPL

A escolha do ambiente de programação apropriado a cada utilizador deve ter em conta as vantagens e desvantagens de cada ambiente. No entanto, as mais importantes, na opinião dos autores, são a facilidade de codificação de problemas e a existência ou não de problemas já codificados. Se o utilizador pretende codificar novos problemas deve usar o AMPL.

5. Referências

[1] I. Bongartz, R. Conn, N. Gould e Ph.L. Toint, *CUTE: Constrained and Unconstrained Testing Environment*, ACM Transactions on Mathematical Software, vol 21, nº1 (123-160), 1995

[2] A.R. Conn, N.I.M. Gould e Ph.L. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (release A)*, Springer Verlag, Heidelberg, New York, 1992

[3] R. Fourer, D.M. Gay e B.W. Kernighan, *AMPL: A Mathematical Programming Language*, Boyd & Fraser Publishing Company, Massachucetts, http://www.ampl.com, 1993

[4] P.E. Gill, W. Murray, M.A. Saunders e M.H. Wright, *User's Guide for NPSOL (Version 4.9): A Fortran Package for Nonlinear Programming*, Stanford Office of Technology Licensing, California, USA, 1986

[5] The MathWorks, Inc., *MATLAB Optimization Toolbox.*

[6] B.A. Murtagh e M.A. Saunders, *MINOS 5.4 User's Guide,* Technical Report SOL83-20R, Department of Operations Research, Stanford University, Stanford, USA, 1993

[7] R.J. Vanderbei, *LOQO: An Interior Point Code for Quadratic Programming,* Technical Report SOR 94-15, Princeton University, 1994

[8] R.J. Vanderbei, *Large-Scale Nonlinear AMPL Models,* http://www.princeton.edu/~rvdb/ampl/nlmodels/.

Anexos

1. Exemplo de problema em SIF

```
***************************
* SET UP THE INITIAL DATA *
***************************
NAME          HS10
*   Problem :
*   *********
*   Source: problem 10 in
*   W. Hock and K. Schittkowski,
*   "Test examples for nonlinear programming codes",
*   Lectures Notes in Economics and Mathematical Systems 187,
Springer
*   Verlag, Heidelberg, 1981.
*   SIF input: A.R. Conn March 1990
*   classification LQR2-AN-2-1
VARIABLES
    X1
    X2
GROUPS
 N  OBJ       X1        1.0            X2        -1.0
 G  CON1
CONSTANTS
    HS10      CON1      -1.0
BOUNDS
 FR HS10      'DEFAULT'
START POINT
    HS10      X1        -10.0
```

```
    HS10      X2        10.0
ELEMENT TYPE
 EV 2PROD     V1                      V2
 EV SQ        V1
ELEMENT USES
 T  E1        SQ
 V  E1        V1                      X1
 T  E2        2PROD
 V  E2        V1                      X1
 V  E2        V2                      X2
 T  E3        SQ
 V  E3        V1                      X2
GROUP USES
 E  CON1      E1        -3.0          E2        2.0
 E  CON1      E3        -1.0
OBJECT BOUND
*   Solution
*LO SOLTN               -1.0
ENDATA
***********************
* SET UP THE FUNCTION *
* AND RANGE ROUTINES  *
***********************
ELEMENTS      HS10
TEMPORARIES
 R  ZERO
INDIVIDUALS
 T  2PROD
 A  ZERO                0.0
 F                      V1*V2
 G  V1                  V2
 G  V2                  V1
 H  V1        V1        ZERO
 H  V2        V2        ZERO
 H  V1        V2        1.0
 T  SQ
 F                      V1 * V1
 G  V1                  2.0 * V1
 H  V1        V1        2.0
ENDATA
```

2. Exemplo de problema em AMPL

```
var x {1..2};
var t;
maximize fx :
   (1/3)*x[1]^2+x[2]^2+(1/2)*x[1];
subject to gxt:
   (1-x[1]^2*t^2)^2-x[1]*t^2-x[2]^2+x[2] <= 0;
subject to ubound:
   t <= 1;
subject to lbound:
   t >= 0;

data;
var x :=
1 1
2 2;
end;
```