

Apostila de Instalação do CUTer

Abel Soares Siqueira* Douglas Gonçalves† Leandro F. Prudente‡
Raniere Gaia Costa da Silva§

21 de março de 2013

Sumário

1	Introdução	2
1.1	Obtendo ajuda	2
2	Instalação	2
3	Interface	5
4	Exemplos	5
4.1	Instruções Gerais	6
4.2	Exemplos em Fortran	7
4.3	Exemplos em C	9
4.4	Exemplos em MATLAB	14
A	Licença	18

A biblioteca CUTer ([1, 2]) é de extrema importância para a realização de testes computacionais na área de otimização computacional. Tendo em vista que é necessário fazer comparações entre os algoritmos, Muitos deles atualmente têm uma interface que permite a execução dos testes do CUTer. Vamos mostrar nesse tutorial como instalar o CUTer e criar uma interface em Fortran, C e C++ para o mesmo. Algumas partes deste tutorial supõem que o leitor saiba o suficiente de linux.

*abel@ime.unicamp.br

†douglas@ime.unicamp.br

‡lfprudente@ime.unicamp.br

§ra092767@ime.unicamp.br

1 Introdução

1.1 Obtendo ajuda

Quando você precisar de problemas com este tutorial, por favor faça o seguinte:

1. Procure em <https://github.com/abelsiqueira/tutoriais/issues> pelo seu problema. Provável que alguém tenha reportado um erro similar.
2. Dê uma lida na documentação oficial do CUTER pois pode ser que este tutorial esteja desatualizado.
3. Entre em contato com um dos autores.

2 Instalação

O CUTER é um ambiente de testes para métodos de otimização. Ele foi feito em Fortran, porém tem suporte a C/C++. Para instalar o CUTER você irá precisar dos programas:

- gawk,
- gcc/g++ (ou outro compilador de C/C++),
- gfortran (ou outro compilador de fortran),
- svn (subversion),
- algum editor de texto (gedit, vim, etc.).

Com os pacotes instalados, iremos criar uma pasta para o CUTER. Você pode criar uma pasta separada no sistema ou instalar tudo no seu diretório principal. No meu caso, irei criar uma pasta chamada `libraries` na minha pasta pessoal, e criar uma pasta para o CUTER dentro dessa pasta. Ajuste os comandos de acordo com sua escolha.

```
$ mkdir -p $HOME/libraries/CUTer  
$ cd $HOME/libraries/CUTer
```

Agora faça o download do SifDec2 pelo comando a seguir

```
$ svn co http://tracsvn.mathappl.polytml.ca/SVN/cuter/sifdec/branches/SifDec2 ./sifdec2
```

E você precisa baixar o CUTER2 agora. Se quiser a versão 32 bits:

```
$ svn co http://tracsvn.mathappl.polytml.ca/SVN/cuter/cuter/branches/CUTer2 ./cuter2
```

E a versão 64 bits:

```
$ svn co http://tracsvn.mathappl.polytml.ca/SVN/cuter/cuter/branches/cuter64 ./cuter2
```

Faça o download da coleção de problemas SIF na página <http://cuter.rl.ac.uk/cuter-www/Problems/mastsif.shtml> ou use os comandos

```
$ wget ftp://ftp.numerical.rl.ac.uk/pub/cuter/mastsif_small.tar.gz
$ wget ftp://ftp.numerical.rl.ac.uk/pub/cuter/mastsif_large.tar.gz
```

Descompacte os problemas pequenos com o comando

```
$ tar -zxvf mastsif_small.tar.gz
```

Se quiser, descompacte os problemas grandes também. Note que estes problemas são muito mais pesados que os pequenos, então aconselho baixá-los apenas se for necessário.

```
$ tar -zxvf mastsif_large.tar.gz
```

Iremos começar a instalar o CUTer. Para isso, vamos instalar o decodificador dos problemas SIF (que faz parte da biblioteca CUTer). Entre na pasta `sifdec2` criada em `$HOME/libraries/CUTer`.

```
$ cd $HOME/libraries/CUTer/sifdec2
```

Agora digite o comando

```
$ ./install_sifdec
```

A instalação do `sifdec` irá pedir que você escolha

- Plataforma (no nosso caso será 5 - PC)
- Sistema Operacional (no nosso caso será 2 - linux)
- Compilador Fortran (no nosso caso será 7 - GNU gfortran)
- Precisão (no nosso caso será D - double)
- Tamanho (no nosso caso será L - large, porém pode ser necessário mudar para C - custom, dependendo do software e dos testes que você utilizará. Nesse caso edite o arquivo na pasta `build/arch/size.custom` antes de continuar.)

O instalador vai lançar a mensagem

```
By default, SifDec with your selections will be installed in
/home/abel/libraries/CUTer/sifdec2/SifDec.large.pc.lnx.gfo
Is this OK (Y/n)?
```

Anote o nome do diretório que será criado e aperte enter. No nosso caso, será `SifDec.large.pc.lnx.gfo`. Para a próxima mensagem, escreva 'n' e aperte enter.

Agora vamos acrescentar as variáveis de sistema para que o instalador e os softwares que formos utilizar saibam onde está o CUTer. Edite o arquivo `$HOME/.bashrc`. Para isso, você pode usar o `gedit` ou algum outro editor de texto. Por exemplo, com o `gedit`, para abrir o arquivo, basta usar o comando

```
$ gedit $HOME/.bashrc &
```

Adicione as seguintes linhas

```
ROOTCUTER="$HOME/libraries/CUTer"
export CUTER="$ROOTCUTER/cuter2"
export MYCUTER="$CUTER/CUTer.large.pc.lnx.gfo"
export SIFDEC="$ROOTCUTER/sifdec2"
export MYSIFDEC="$SIFDEC/SifDec.large.pc.lnx.gfo"
export MASTSIF="$ROOTCUTER/mastsif"
export MANPATH="$CUTER/common/man:$SIFDEC/common/man:$MANPATH"
export PATH="$MYCUTER/bin:$MYSIFDEC/bin:$PATH"
```

Se for necessário, mude a variável `ROOTCUTER`, e o diretório que eu mandei anotar. Note que já criamos um diretório para o CUTer, supondo que ele siga as mesmas opções que o SifDec. Para que essas mudanças façam efeito é necessário usar o comando

```
$ source $HOME/.bashrc
```

ou reiniciar o terminal. Agora entre na pasta que criamos anteriormente. Para isso basta fazer

```
$ cd $MYSIFDEC
```

Se deu algum erro, você pode ter errado o nome na definição da variável. Verifique que a pasta realmente existe e refaça os passos se necessário.

Agora rode o comando

```
$ ./install_mysifdec
```

Ao final desse comando, a frase

```
install_mysifdec : Do you want to 'make all' in
/home/abel/libraries/CUTer/sifdec2/SifDec.large.pc.lnx.gfo now (Y/n)?
```

aparecerá. Aperte enter. Agora vamos instalar o CUTer. Para isso, faça

```
$ cd $CUTER
$ ./install_cuter
```

A instalação do CUTer irá pedir que você escolha

- Plataforma (no nosso caso será 5 - PC)
- Sistema Operacional (no nosso caso será 2 - linux)
- Compilador Fortran (no nosso caso será 7 - GNU gfortran)
- Compilador C (no nosso caso será 4 - g++)
- Precisão (no nosso caso será D - double)
- Tamanho (no nosso caso será L - large, com a mesma observação sobre o tamanho.)

Quando necessário, aperte enter para confirmar a continuação do programa. Se tudo deu certo, o CUTer deve estar instalado. Caso tenha acontecido algum erro, reveja as variáveis e volte os passos. Pode ser necessário apagar tudo que você baixou e baixar novamente (em último caso). Para verificar que realmente está tudo funcionando, faça

```
$ mkdir -p $HOME/libraries/CUTEr/Testing
$ cd $HOME/libraries/CUTEr/Testing
$ runcuter -p gen -D ROSENB
```

Se nenhum erro aparecer, e aparecer várias informações do problema, então o CUTEr foi instalado corretamente.

3 Interface

A biblioteca CUTEr dá ao usuário um conjunto de funções para receber informações do problema. As funções são separadas para o caso irrestrito (precedidas pela letra *U*) ou restrito, (precedidas pela letra *C*). Se o usuário necessitar do valor da função objetivo num ponto dado, ele pode chamar UFN se o problema for irrestrito, ou CFN caso contrário. Note que a sintaxe das funções não é a mesma, pois o CUTEr tenta ser o mais prático possível. A sintaxe dessas funções é

- UFN (*N*, *X*, *F*), onde *N* é um inteiro indicando o número de variáveis do problema, *X* é um vetor de reais, e *F* é real que sai com o valor da função objetivo.
- CFN (*N*, *M*, *X*, *F*, *LC*, *C*), onde *N*, *X* e *F* indicam as mesmas coisas, *M* é um inteiro indicando o número de restrições, *C* é um vetor de reais, com os valores das restrições e *LC* é a dimensão real de *C*, que deve ser maior ou igual a *M*.

Existem muitas outras funções para interação com o problema. Elas podem ser vistas na documentação do CUTEr no arquivo `general.pdf` dentro da pasta `$CUTEr/common/doc` nas páginas 31 e 32.

Para rodar nossa biblioteca com o CUTEr, é necessário criar alguns arquivos extras. Um deles é o código que relaciona o nosso programa com essas funções do CUTEr. Além disso, também é necessário criar um arquivo que indica ao CUTEr quais as bibliotecas que nossa biblioteca precisa. A maneira tradicional de se trabalhar com o CUTEr (o novo CUTEr, pelo menos) é compilar nossa biblioteca inteira para um arquivo `.a`, compilar esse arquivo de ligação do CUTEr com nossa biblioteca, e passar tudo isso pro CUTEr. Quando necessário, rodamos o nosso pacote pelo comando do CUTEr `runcuter`. Por exemplo, para rodar o pacote `pack_exemplo`, usamos o comandos

```
$ runcuter -p pack_exemplo -D problema
```

onde `problema` é um dos problemas do CUTEr sem a extensão `.SIF`.

4 Exemplos

Como exemplos do CUTEr, vamos criar bibliotecas em diversas linguagens para resolver o problema de minimização irrestrita. Vamos implementar o método de máxima descida com busca linear utilizando o critério de Armijo.

Considere o problema

$$\min f(x), \quad x \in \mathbb{R}^n,$$

onde $f : \mathbb{R}^n \rightarrow \mathbb{R}$ é contínua e derivável. Vamos procurar um ponto estacionário para esse problema, isto é, um ponto $x^* \in \mathbb{R}^n$ tal que

$$\nabla f(x^*) = 0.$$

Obviamente, como vamos implementar este método, vamos parar quando encontrarmos um iterando x^k tal que $\|\nabla f(x^k)\| \leq \varepsilon$, onde $\varepsilon > 0$ é dado. O método está descrito a seguir.

Algorithm 1 Método de Máxima Descida

```

1: Dados  $x^0 \in \mathbb{R}^n$ ,  $\varepsilon > 0$ ,  $\alpha \in (0, 1)$ ,  $k = 0$ .
2: while  $\|\nabla f(x^k)\| > \varepsilon$  do
3:    $d^k = -\nabla f(x^k)$ 
4:    $\lambda_k = 1$ 
5:   while  $f(x^k + \lambda_k d^k) > f(x^k) + \alpha \lambda_k \nabla f(x^k)^T d^k$  do
6:      $\lambda_k = \lambda_k / 2$ 
7:   end while
8:    $x^{k+1} = x^k + \lambda_k d^k$ 
9:    $k = k + 1$ 
10: end while
11:  $x^* = x^k$ .
```

Vamos implementar este método em algumas linguagens, e às vezes, mais de uma vez, para exemplificar a interface CUTeR.

4.1 Instruções Gerais

Os passos a seguir são feitos normalmente após a criação da biblioteca, porque primeiro você precisa fazer com que ela funcione em outros problemas. Na dúvida, faça um dos exemplos a seguir e depois volte para ler esta seção.

Para que seu programa funcione com o CUTeR, você irá precisar de um arquivo de protótipo que pode ser copiado de `$CUTeR/build/prototypes/gen90.sh.pro`. Nesse arquivo, você irá mudar algumas variáveis para o compilador do CUTeR encontrar seus arquivos. As linhas sem comentários estão mostradas a seguir:

```

export PACK=gen90
export PACKAGE=gen
export PACK_PRECISION="single double"
export PACKOBS="cuterinter.o gen90.o"
export PACKLIBS=""
export SPECS="GEN.SPC"
```

Vamos supor que temos um pacote chamado `cutertest`, e que esse pacote precisa de um objeto `auxiliar.o`, e de uma biblioteca do sistema `libalgo.a`. Além disso, você deve ter criado (veja nos exemplos) um arquivo específico para fazer a interface entre o CUTeR e o pacote `cutertest`. Vamos supor que a linguagem em questão seja `fortran`, então você deve ter criado um arquivo `cutertestmain.f`. A variável `PACK` define um pequeno nome para o seu pacote, e `PACKAGE` é o nome de seu pacote. `PACK_PRECISION` é a precisão do seu pacote (muitas vezes

double). `PACKOBS` são os objetos que você deve colocar na pasta `$MYCUTER/(precision)/bin` onde `(precision)` é a precisão que você instalou o CUTer. `PACKLIBS` são as bibliotecas que o pacote precisa, e `SPECS` se refere a algum arquivo de especificações que o seu pacote pode depender. Esse arquivo pode ser local (isto é, na pasta onde você irá chamar o CUTer) ou na pasta `$CUTER/common/src/pkg/$PACKAGE/`.

Para o nosso exemplo deveríamos ter no arquivo chamado `cutertest.sh.pro` as seguintes definições:

```
export PACK=cutertest
export PACKAGE=cutertest
export PACK_PRECISION="double"
export PACKOBS="auxiliar.o"
export PACKLIBS="-lalgo"
export SPECS=""
```

Note que devemos copiar o arquivo `auxiliar.o` para `$MYCUTER/double/bin`. Agora, precisamos passar esse arquivo por um formatador de texto utilizando um script que o CUTer fornece e devemos habilitá-lo para execução. Para isso, os comandos são

```
$ sed -f $MYCUTER/double/config/script.sed cutertest.sh.pro > $MYCUTER/bin/cutertest
$ chmod a+x $MYCUTER/bin/cutertest
```

Agora passamos o arquivo de interface e o arquivo compilado para o CUTer.

```
$ cp -f cutertestmain.f $MYCUTER/double/bin/cutertestma.f
$ cp -f cutertestmain.o $MYCUTER/double/bin/cutertestma.o
```

Tendo feito isso, agora seu arquivo está pronto para ser executado. Use o comando

```
$ runcuter -p cutertest -D ROSENBR
```

4.2 Exemplos em Fortran

Fizemos uma implementação do método de máxima descida. Temos dois arquivos na implementação:

- `gradient.f`: Este arquivo contém a definição do método.
- `gradientmain.f`: Este arquivo contém a rotina principal do fortran.

Além desses arquivos também é necessário um arquivo com as definições das subrotinas

- `inip(n,x)`: Retorna `n`, a dimensão do problema, e `x`, o ponto inicial.
- `evalf(n,x,f)`: Recebe a dimensão do problema `n`, e o ponto `x`, e retorna o valor da função objetivo em `f`.
- `evalg(n,x,g)`: Recebe a dimensão do problema `n`, e o ponto `x`, e retorna o valor do gradiente da função objetivo em `g`.
- `endp(n,x)`: Imprime informações sobre a solução.

Para criar a interface em fortran, é necessário apenas criar um arquivo com as subrotinas acima. O arquivo com a interface (sem os comentários está abaixo:)

```

subroutine inip(n,x)

implicit none

integer n

double precision x(*)

integer i

integer err, ifile, nt, m, nmax
PARAMETER (nmax=10000)
double precision bl(nmax), bu(nmax)

ifile = 30
OPEN(ifile, FILE='OUTSDIF.d', FORM='FORMATTED',
$ STATUS='OLD', IOSTAT=err)
REWIND ifile
IF (err.NE.0) THEN
  WRITE(*,*) 'Could not open the OUTSDIF.d file'
  STOP
ENDIF

CALL cdimen(ifile, nt, m)

if (nt.GT.nmax) THEN
  WRITE(*,*) 'Increase nmax'
  STOP
ENDIF
if (m.GT.0) THEN
  WRITE(*,*) 'Cannot handle constraints'
  STOP
ENDIF

CALL usetup(ifile, 7, n, x, bl, bu, nmax)

DO i = 1,n
  IF ((bl(i).GT.-1.0D20).OR.(bu(i).LT.1.0D20)) THEN
    WRITE(*,*) 'Cannot handle boxes'
    STOP
  ENDIF
ENDDO

end

C *****
C *****

subroutine evalf(n,x,f)

implicit none

integer n
double precision f

double precision x(n)

CALL ufn(n, x, f)

```



```

end

C *****
C *****

subroutine evalg(n,x,g)

implicit none

integer n

double precision g(n),x(n)

CALL UGR(n, x, g)

end

C *****
C *****

subroutine endp(n,x)

implicit none

integer n
double precision x(n)
integer i

write(*,*)'Solution:'
do i = 1,n
    write(*,*)x(i)
end do

end

```

4.3 Exemplos em C

Fizemos três implementações do método de máxima descida. A primeira é uma implementação que não leva em conta o CUTer, e depois adapta o CUTer para o problema. A segunda já leva em conta o formato das funções do CUTer e faz pouca adaptação posteriormente. A terceira usa exatamente as funções do CUTer, não necessitando de adaptação.

Cada implementação do método de máxima descida consiste de dois arquivos: `steepest_descent.h` e `steepest_descent.c`. No `.h`, definimos que funções iremos chamar, e uma estrutura com as informações da execução. As funções para a primeira implementação são

- `double Norm (double * x, unsigned int n);`
Esta função calcula a norma 2 de um vetor `x` com tamanho `n`.
- `double NormSqr (double * x, unsigned int n);`
Esta função calcula o quadrado da norma 2 de um vetor `x` com tamanho `n`. É mais rápido que a função `Norm` pois não envolve raiz quadrada.
- `SteepestDescent (double * x, unsigned int n, Status * status);`
Esta é a função que encontra o ponto estacionário. `x` entra como ponto inicial e sai como

a solução. `n` é a dimensão do problema e `status` é um ponteiro para a estrutura de informações.

- `SD_Print (double * x, unsigned int n, Status * status);`

Esta função imprime o vetor `x` e as informações da execução do problema.

A estrutura do problema

```
typedef struct _Status {
    unsigned int iter;
    double f, ng;
    unsigned int n_objfun, n_gradfun;
} Status;
```

`iter` é o número de iterações que o algoritmo executou, `f` é o valor da função objetivo na solução, `ng` é a norma do gradiente da função objetivo, `n_objfun` é o número de cálculos da função objetivo e `n_gradfun` é o número de cálculos do gradiente. Mostraremos as diferenças das outras implementações posteriormente.

Nosso arquivo `.c` contém as definições das funções acima, e contém uma declaração de função usada para acessar a função objetivo e o gradiente. Na primeira implementação, essa declaração é

```
double objfun (double * x, unsigned int n);
void gradfun (double * x, unsigned int n, double * g);
```

As funções `Norm`, `NormSqr` e `SD_print` são idênticas para todas as implementações e serão deixadas de fora. A implementação do método em si encontra-se abaixo.

```
void SteepestDescent (double * x, uint n, Status *status) {
    double * g, f, fp;
    double * xp, lambda, ng_sqr;
    uint i;

    if ( (x == 0) || (status == 0) )
        return;

    g = (double *) malloc(n * sizeof(double) );
    xp = (double *) malloc(n * sizeof(double) );
    status->iter = 0;
    status->n_objfun = 0;
    status->n_gradfun = 0;

    f = objfun(x, n);
    status->n_objfun++;
    gradfun(x, n, g);
    status->n_gradfun++;

    status->ng = Norm(g, n);

    while (status->ng > EPSILON) {
        lambda = 1;

        for (i = 0; i < n; i++) {
            xp[i] = x[i] - g[i];
        }

        fp = objfun(xp, n);
        status->n_objfun++;
```

```

    ng_sqr = status->ng*status->ng;

    while (fp > f - 0.5 * lambda * ng_sqr) {
        for (i = 0; i < n; i++) {
            xp[i] = x[i] - lambda*g[i];
        }
        lambda = lambda/2;
        fp = objfun(xp, n);
        status->n_objfun++;
    }

    for (i = 0; i < n; i++)
        x[i] = xp[i];

    f = objfun(x, n);
    status->n_objfun++;
    gradfun(x, n, g);
    status->n_gradfun++;
    status->ng = Norm(g, n);
    status->iter++;
}

status->f = f;

free(xp);
free(g);
}

```

Um código de exemplo para esse teste é

```

#include <stdio.h>
#include "steepest_descent.h"

/*
 * Each file testx.c is a different problem. The user will have to
 * implement his own file, defining objfun and gradfun.
 */

/*
 * This problem is
 *
 *  $\min f(x) = 0.5*(x_1^2 + x_2^2)$ 
 *
 * starting from point  $x_0 = (1,2)$ ;
 */

double objfun (double * x, unsigned int n) {
    (void)n;
    return 0.5 * (x[0]*x[0] + x[1]*x[1]);
}

void gradfun (double * x, unsigned int n, double * g) {
    unsigned int i;

    for (i = 0; i < n; i++)
        g[i] = x[i];
}

int main () {
    double x[2];
    Status status;

```

```

x[0] = 1;
x[1] = 2;

SteepestDescent(x, 2, &status);

SD_Print(x, 2, &status);

return 0;
}

```

Este código implementa o problema de minimizar $f(x) = \frac{1}{2}\|x\|^2$ em duas dimensões. Note como temos que declarar as funções `objfun` e `gradfun`. Sem elas teríamos erros na compilação. Veja os arquivos `test2.c` e `test3.c` para outros exemplos.

A interface para o CUTer é o arquivo `c_example1main.c`:

```

#include "cuter.h"
#include "CExample1/steepest_descent.h"

double objfun (double * x, unsigned int n) {
    double F = 0;
    int N = n;
    UFN(&N, x, &F);
    return F;
}

void gradfun (double * x, unsigned int n, double * g) {
    int N = n;
    UGR(&N, x, g);
}

int MAINENTRY () {
    double *x, *bl, *bu;
    char fname[10] = "OUTSDIF.d";
    int nvar = 0, ncon = 0, nmax;
    int funit = 42, ierr = 0, fout = 6;
    int i;
    Status status;

    FORTRAN_OPEN(&funit, fname, &ierr);
    CDIMEN(&funit, &nvar, &ncon);

    if (ncon > 0) {
        printf("ERROR: Problem is not unconstrained\n");
        return 1;
    }

    x = (double *) malloc (sizeof(double) * nvar);
    bl = (double *) malloc (sizeof(double) * nvar);
    bu = (double *) malloc (sizeof(double) * nvar);

    USETUP(&funit, &fout, &nvar, x, bl, bu, &nmax);

    for (i = 0; i < nvar; i++) {
        if ( ( bl[i] > -CUTE_INF) || (bu[i] < CUTE_INF) ) {
            printf("ERROR: Problem has bounds\n");
            return 1;
        }
    }

    SteepestDescent(x, nvar, &status);
}

```

```

SD_Print(x, nvar, &status);

free(x);
free(bl);
free(bu);

return 0;
}

```

Note que também é necessário definir as funções `objfun` e `gradfun`. Essas funções, por sua vez, chamam as funções correspondentes em CUTer para esse serviço. A função `UFN` calcula o valor da função objetivo e a função `UGR` calcula o valor do gradiente. Note que quando compilamos uma função em Fortran, ele recebe um nome em letras minúsculas e com um `_` (underline) na frente (no caso do `gfortran`. Outros compiladores podem divergir). O arquivo `cuter.h` define macros para todas as funções do CUTer serem chamados com letras maiúsculas. Note ainda que a função `UFN` recebe um ponteiro para `int` e dois ponteiros para `double`, sendo o primeiro para o vetor `x` e o segundo para o valor da função objetivo. As funções do CUTer para C recebem ponteiros em todos os valores. Os valores que são vetores não precisam de um ponteiro adicional. Note também que como utilizamos `unsigned int` para os tamanhos, tivemos que converter os valores para `int`.

Veja agora o Exemplo 2. Nesse exemplo consideramos que as funções devem estar no mesmo formato que a função do CUTer.

```

void ufn (int * n, double * x, double * f);
void uofg (int * n, double * x, double * f, double * g, long int * grad);

```

O nome das funções foram escolhidas para seguir exatamente o formato do CUTer, mas aqui elas poderiam ser qualquer coisa. Note que a função `uofg` foi utilizada no lugar da função `ugr`. Essa função já calcula a função objetivo e o gradiente, sendo mais rápida que chamadas individuais. Com essa mudança, esse programa é levemente mais rápido que o outro. O resto do arquivo foi mudado de acordo a seguir essas mudanças.

A interface também teve uma mudança na definição das funções:

```

void ufn (int * n, double * x, double * f) {
    UFN(n, x, f);
}

void uofg (int * n, double * x, double * f, double * g, long int * grad) {
    UOFG(n, x, f, g, grad);
}

```

Essa interface fica muito mais natural de ser utilizada num problema com CUTer. Não precisamos converter nenhuma variável, simplesmente fazer a chamada da função com os parâmetros já dados. Note, no entanto, que estamos utilizando `long int` para as variáveis lógicas do CUTer. Essa é uma definição do arquivo `cuter.h`. Se mudarmos essa definição, então devemos criar uma variável `logical GRAD = *grad` e chamar a função com `UOFG(n, x, f, g, &GRAD)`.

A última interface já declara as funções que o CUTer irá definir. Então no arquivo `.c` temos

```

void ufn_ (int * n, double * x, double * f);
void uofg_ (int * n, double * x, double * f, double * g, long int * grad);

```

e no arquivo da interface não temos nenhuma definição de função. Lembre-se que o Fortran compilado com gfortran cria as funções com minúsculas e `_` na frente. Se mudarmos o compilador pode não funcionar. Além disso, a definição dessas funções é, na verdade

```
void UFN( integer *n, doublereal *x, doublereal *f );
void UOFG( integer *n, doublereal *x, doublereal *f, doublereal *g,
           logical *grad);
```

e esses tipos são definidos no arquivo `cuter.h`, podendo ser alterados pelo usuário. Se isso acontecer, é necessário mudar todo o programa.

Uma alternativa é utilizar `typedefs` para definir os tipos próprios, e deixar o acesso desses tipos para o usuário (assim como o arquivo `cuter.h`). Dessa maneira, se o usuário tiver necessidade de mudar o arquivo `cuter.h`, ele também pode (deve) mudar o arquivo com esses `typedefs`.

Para compilar a interface CUTER dos testes utilize

```
$ make cuter
```

Para rodar os testes utilize o comando

```
$ runcuter -p c_example# -D BARD
```

onde `#` é o número do exemplo e BARD é um dos problemas em que esse exemplo converge. Note que é preferível criar uma pasta separada para rodar os testes, já que eles geram lixo na pasta.

4.4 Exemplos em MATLAB

Com o CUTER devidamente instalado, vejamos como ocorre a interface com o MATLAB.

Primeiro, crie uma pasta **Test**, onde serão gerados os arquivos decodificados do problema e o arquivo `.mex` da interface para MATLAB.

No terminal, dentro da pasta **Test**, execute o comando:

```
$ runcuter -p mx -D ROSENBR
```

Além dos arquivos padrão, gerados pelo decoder para o problema ROSENBR, haverá também o arquivo: `mcuter.mex`. A extensão `.mex` poderá variar conforme a instalação do CUTER e o sistema operacional.

Em seguida, vá para o MATLAB. Adicione as pastas `$CUTER/common/src/matlab` e `$MYCUTER/bin` ao path do MATLAB. Isso pode ser feito através do menu **File > Set Path...** ou pelo prompt do MATLAB usando o comando `addpath`.

Dentro do MATLAB, vá para a pasta **Test**. Para verificar se está tudo certo, execute o comando:

```
>> prob = cuter_setup()

prob =

      n: 2
      m: 0
    nnzh: 3
    nnzj: 0
       x: [2x1 double]
      b1: [2x1 double]
```

```

bu: [2x1 double]
v: [0x1 double]
cl: [0x1 double]
cu: [0x1 double]
equatn: [0x1 logical]
linear: [0x1 logical]
name: 'ROSENBR '

```

Se estiver tudo correto, a saída deverá ser como acima.

O comando `cuter_setup`, que inicializa o problema e fornece suas características, se encontra na pasta `$CUTER/src/common/matlab`, dentro da qual estão as demais rotinas (arquivos `.m`) para acessar função objetivo, gradiente, restrições, etc.

Por exemplo, para saber o valor da função objetivo em um determinado ponto, usamos:

```

>> f = cuter_obj([-1 2]')

f =

    104

```

Abaixo temos um código em MATLAB que implementa o método do gradiente. Para acessar a função objetivo e o gradiente, fizemos uso da função `cuter_obj`.

```

function [x,f,gradnorm,iter,flag] = cute_gradient()
%
% [x,f,gradnorm,iter,flag] = cute_gradient()
%
% flags:
% -2      maximum number of iterations reached
% -1      too short step size
% 1       gradient norm less than eps0
%
% inicializando o problema
prob = cuter_setup();

% dimensao do problema
n = prob.n;

% ponto inicial
x0 = prob.x;

%=====
% Gradient method with linesearch
%=====
maxit=n*10000;
gamma=1e-4;
eps0=1e-5;
tmin=1e-8;
done=0;
flag=0;
k=0;
x=x0;

while ~done
    k=k+1;

    % maximum number of iterations test
    if k>maxit
        done=1;
    end
end

```

```

        flag=-2;
        continue
    end

    x0=x;

    [f,g] = cuter_obj(x);
    f0=f;

    gradnorm=norm(g,2);

    % gradient norm test
    if gradnorm<eps0
        done=1;
        flag=1;
        continue
    end

    % search direction
    d = -g;
    gtd = g'*d;
    t = 1;

    x = x0 + t*d;
    f = cuter_obj(x);

    % Armijo linesearch
    while f > f0 + t*gamma*gtd
        t = 0.5*t;
        if t<tmin
            done=1;
            flag=-1;
        end

        x = x0 + t*d;
        f = cuter_obj(x);
    end

end

iter=k;

end

```

E executando o código acima, obtemos

```

>> [x,f,gradnorm,iter,flag] = cute_gradient()

x =

    1.0000
    1.0000

f =

    6.1319e-11

gradnorm =

    9.9971e-06

```



```
iter =  
  
    10917  
  
flag =  
  
    1
```

É possível também utilizar as rotinas de otimização do próprio MATLAB. Por exemplo, para utilizar o `fminunc`, usamos:

```
>> prob = cuter_setup();  
>> [x,f,flag] = fminunc(@(x) cuter_obj(x),prob.x)  
Warning: Gradient must be provided for trust-region method;  
    using line-search method instead.  
> In fminunc at 356  
  
Local minimum found.  
  
Optimization completed because the size of the gradient is less than  
the default value of the function tolerance.  
  
<stopping criteria details>  
  
x =  
  
    1.0000  
    1.0000  
  
f =  
  
    2.8336e-11  
  
flag =  
  
    1
```

Lembrando que as opções do solver são ajustadas pelo comando `optimset`.

```
>> prob = cuter_setup();  
>> opts = optimset('GradObj','on');  
>> [x,f,flag] = fminunc(@(x) cuter_obj(x),prob.x,opts)  
  
Local minimum possible.  
  
fminunc stopped because the final change in function value relative to  
its initial value is less than the default value of the function tolerance.  
  
<stopping criteria details>  
  
x =  
  
    1.0000  
    1.0000
```

```
f =  
  
4.0035e-13  
  
flag =  
  
3
```

A Licença

Esta obra está licenciada sob a licença Creative Commons Atribuição 3.0 Não Adaptada. Para ver uma cópia desta licença, visite <http://creativecommons.org/licenses/by/3.0/>.



Referências

- [1] <http://cutter.rl.ac.uk/cuter-www>
- [2] <https://magi-trac-svn.mathappl.polymtl.ca/Trac/cuter>