Apostila de Instalação do CUTEst

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

24 de junho de 2014

Sumário

A biblioteca CUTEst ([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 dos mais famosos atualmente têm uma interface que permite a execução dos testes do CUTEst. Vamos mostrar nesse tutorial como instalar o CUTEst e criar uma

*abel.s.siqueira@gmail.com
†douglas@ime.unicamp.br - desatualizado
‡lfprudente@ime.unicamp.br - desatualizado
§ra092767@ime.unicamp.br

interface em Fortran e C para o mesmo. Algumas partes deste tutorial supõem que o leitor saiba o suficiente de linux.

Para uma interface em Windows ou Mac, leia veja a Seção 5.

1 Introdução

1.1 Obtendo ajuda

Quando você precisar de ajuda com assuntos relacionados a este tutorial, 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 CUTEst pois pode ser que este tutorial esteja desatualizado.
3. Entre em contato com um dos autores.
4. Procure no irc, servidor `freenode` canal `##cutest`.

Tenha em mente que estamos fazendo esse serviço de bondade. Não temos nenhuma relação direta com o projeto ou seus autores. Nosso conhecimento sobre o assunto é limitado.

2 Instalação

O CUTEst é um ambiente de testes para métodos de otimização. Ele foi feito em Fortran, com suporte a C. Para usar em C++, o trabalho é um pouco maior. Depois de entender o processo descrito aqui para C, verifique o trabalho de [**?**] como exemplo. Para instalar o CUTEr você irá precisar dos programas:

- gawk,
- gcc (ou outro compilador de 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 CUTEst. 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 CUTEst dentro dessa pasta. Ajuste os comandos de acordo com sua escolha.

2.1 Preparação e download

O CUTEst trabalha com quatro diretórios: archdefs, cutest, sifdecode e sif. Você precisa definir uma variável de sistema para cada um desses diretórios. Vamos fazer isso logo.

Inclua no seu arquivo `$HOME/.bashrc` as seguintes linhas:

```
LIBSDIR="$HOME/libraries"
export ARCHDEFS="$LIBSDIR/archdefs"
export SIFDECODE="$LIBSDIR/sifdecode"
export CUTEST="$LIBSDIR/cutest"
export MASTSIF="$LIBSDIR/sif"
```

Feche o terminal e abra um novo, e vamos começar a instalação.

Digite

```
$ mkdir -p $LIBSDIR
$ cd $LIBSDIR
```

Agora você precisa fazer o download das bibliotecas usando o subversion. Para detalhes veja [?]. Sem detalhes, copie o arquivo abaixo para um arquivo `downcutest.sh` e salve-o na pasta `$LIBSDIR`.

```
#!/bin/bash
cmd="svn checkout -q --username anonymous"
url="http://ccpforge.cse.rl.ac.uk/svn/cutest/"
for name in archdefs sifdecode cutest sif
do
  echo "Downloading $name"
  $cmd $url/$name/trunk ./$name
done
```

Faça esse arquivo ficar executável e execute-o com os comandos

```
$ chmod a+x downcutest.sh
$ ./downcuter.sh
```

Caso não funcione alguma coisa, você pode retirar o `'-q'` da chamada do svn para verificar qual o erro.

2.2 Instalação

Para instalar o CUTEst, comece entrando no diretório do archdefs

```
$ cd $ARCHDEFS
```

Daí, execute o script `install_optsuite` e siga as instruções

```
$ ./install_optsuite
```

As perguntas feitas são:

```
Do you wish to install GALAHAD (Y/n)?
Do you wish to install CUTEst (Y/n)?
Do you require the CUTEst-Matlab interface (y/N)?
Select platform
Select operating system
Would you like to review and modify the system commands (y/N)?
Select fortran compiler
```

```
Would you like to review and modify the fortran compiler settings (y/N)?
Select C compiler
Would you like to review and modify the C compiler settings (y/N)?
Would you like to compile SIFDecode ... (Y/n)?
Would you like to compile CUTEst ... (Y/n)?
Which precision do you require for the installed subset (D/s/b) ?
```

As etapas acima são totalmente dependentes da sua configuração. A maior parte vem com o default que você provavelmente gostará de escolher. No caso, o GALAHAD eu não selecionei, e as outras opções foram default. A plataforma que eu escolhi foi um PC com processador de 64 bits genérico. O sistema operacional é linux. Os compiladores de fortran e C são o gfotran e o gcc, respectivamente.

Ao fim da instalação, o script imprime uma linha tipo

```
export MYARCH="xxxxxxxxxxx"
```

Copie essa linha para o seu `$HOME/.bashrc` e reinicie o terminal. No meu caso, de acordo com minhas escolhas para o instalar, seu MYARCH é `pc64.lnx.gfo`. Se nenhum erro apareceu, você deve ter o CUTEst instalado.

2.3 Testando o CUTEst

Para testar, vamos criar um diretório temporário e rodar o CUTEst com pacotes genéricos, feitos para teste.

```
$ cd $(mktemp -d)
$ runcutest -p genc -D ROSENBR
$ runcutest -p gen77 -D ROSENBR
$ runcutest -p gen90 -D ROSENBR
```

Cada comando destes 3 últimos deve retornar uma tela com estatísticas CUTEst. Eles são, respectivamente, para C, fortran 77 e fortran 90.

3 Interface

A biblioteca CUTEst dá ao usuário um conjunto de funções para receber informações do problema. As funções são do tipo `cutest_nome`, e são separadas para o caso irrestrito ou restrito, onde o nome é iniciado com as letras U e C, respectivamente. Por exemplo, se o usuário necessitar do valor da funcão objetivo num ponto dado, ele pode chamar `cutest_ufn` se o problema for irrestrito, ou `cutest_cfn` caso seja restrito. Note que a sintaxe das funções não é a mesma.

- `cutest_ufn (status, N, X, F)`, onde `N` é um inteiro indicando o número de variáveis do problema, `X` é um vetor de reais, `F` é real que sai com o valor da função objetivo, e `status` é uma variável de saída indicando se houve erro.
- `cutest_cfn (status, N, M, X, F, 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 `status` é uma variável de saída indicando se houve erro.

Existem muitas outras funções de interface do CUTEst. Você vai ter que procurar um pouco até achar o que você quer. Veja o arquivo `cutest.pdf` dentro da pasta `/doc/pdf` do CUTEst.

4 Exemplos

Como exemplos do CUTEst, 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

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

onde $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ 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: $\quad d^k = -\nabla f(x^k)$
4: $\quad \lambda_k = 1$
5: $\quad$ **while** $f(x^k + \lambda_k d^k) > f(x^k) + \alpha\lambda_k \nabla f(x^k)^T d^k$ **do**
6: $\quad\quad \lambda_k = \lambda_k/2$
7: $\quad$ **end while**
8: $\quad x^{k+1} = x^k + \lambda_k d^k$
9: $\quad k = k + 1$
10: **end while**
11: $x^* = x^k$.

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

4.1 Instruções Gerais

Algumas mudanças consideráveis aconteceram do CUTEr para o CUTEst. Recomendamos que você escolhe uma das interfaces apresentadas e investigue o diretório do mesmo. Os arquivos principais de uma interface `pack` são o `packmain.c`, o `Makefile`, o `makemaster` e o `pack.sh.pro`.

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

C     SCALAR ARGUMENTS
      integer n

C     ARRAY ARGUMENTS
      double precision x(*)

C     This subroutine must set some problem data.
C
C     Parameters of the subroutine:
C
C     On Entry:
C
C     This subroutine has no input parameters.
C
C     On Return
C
C     n       integer,
C             number of variables,
C
C     x       double precision x(n),
C             initial point.

C     LOCAL SCALARS
      integer i

C     CUTER STUFF - Include by Abel
      integer err, ifile, m
      double precision bl(n), bu(n)
      integer st

      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 cutest_cdimen(st, ifile, n, m)
```

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

      CALL cutest_usetup(st, ifile, 7, 11, n, x, bl, bu)

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

C     Removed some lines.
C     End of CUTER STUFF

      end

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

      subroutine evalf(n,x,f)

      implicit none

C     SCALAR ARGUMENTS
      integer n, st
      double precision f

C     ARRAY ARGUMENTS
      double precision x(n)

C     This subroutine must compute the objective function.
C
C     Parameters of the subroutine:
C
C     On Entry:
C
C     n       integer,
C             number of variables,
C
C     x       double precision x(n),
C             current point,
C
C     On Return
C
C     f       double precision,
C             objective function value at x,

C     Objective function
      CALL cutest_ufn(st, n, x, f)

      end

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

      subroutine evalg(n,x,g)

      implicit none
```

```
C     SCALAR ARGUMENTS
      integer n, st

C     ARRAY ARGUMENTS
      double precision g(n),x(n)

C     This subroutine must compute the gradient vector of the objective
C     function.
C
C     Parameters of the subroutine:
C
C     On Entry:
C
C     n       integer,
C             number of variables,
C
C     x       double precision x(n),
C             current point,
C
C     On Return
C
C     g       double precision g(n),
C             gradient vector of the objective function evaluated at x,

C     Gradient vector

      CALL cutest_ugr(st, n, x, g)

      end

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

      subroutine endp(n,x,f,g,exitflag)

      implicit none

C     SCALAR ARGUMENTS
      integer n

C     ARRAY ARGUMENTS
      double precision x(n)
      double precision f, ng
      double precision g(n)
      integer exitflag

C     This subroutine can be used to do some extra job after the solver
C     has found the solution.
C
C     Parameters of the subroutine:
C
C     The paraemters of this subroutine are the same parameters of
C     subroutine inip. But in this subroutine there are not output
C     parameter. All the parameters are input parameters.

C     LOCAL SCALAR
      integer i, st

      character ( len = 10 ) :: pname
      character ( len = 10 ), dimension(n) :: vnames
      character ( len = 25 ) :: filename
      character ( len = 10 ) :: charflag
```

```
    double precision calls(4)
    double precision time(2)

    ng = 0.0d0
    do i = 1,n
      ng = max(ng, abs(g(i)))
    end do

    if (exitflag.eq.0) then
      write(*,*) 'Converged'
    elseif (exitflag.eq.1) then
      write(*,*) 'Maximum iteration reached'
    endif

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

    write(*,*) 'f = ', f
    write(*,*) '|g| = ', ng

    call cutest_unames(st, n, pname, vnames)

    filename = trim(pname)//'.tableline'
    open(unit=2, file=filename)

    call cutest_ureport(st, calls, time)

    if (exitflag.eq.0) then
      charflag = 'Converged'
    elseif (exitflag.eq.1) then
      charflag = 'fail'
    endif

    write(2,*) 'Problem |EXITFLAG | TIME   |    FVAL   |  GRADN
   CORM'
    write(2,1000) pname, charflag, time(1)+time(2), f, ng
1000 format(A9,1X,A9,1X,ES12.6,1X,SP,ES13.6,1X,ES13.6)

    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 CUTEst, e depois adapta o CUTEst para o problema. A segunda já leva em conta o formato das funções do CUTEst e faz pouca adaptação posteriormente. A terceira usa exatamente as funções do CUTEst, 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.

```
#include "steepest_descent.h"

#define EPSILON 1e-6

typedef unsigned int uint;

/*
 * These functions are necessary for the software. They must be
 * implemented in a separate file by the user.
 */
double objfun (double * x, uint n);
void   gradfun (double * x, uint n, double * g);

double Norm (double * x, uint n) {
  uint i;
  double s = 0.0;

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

  return sqrt(s);
}

double NormSqr (double * x, uint n) {
```

```
  uint i;
  double s = 0.0;

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

  return s;
}

void SteepestDescent (double * x, uint n, Status *status) {
  double * g, f, fp;
  double * xp, lambda, ng_sqr;
  double alpha = 1e-4;
  uint maxiter = 1e4;
  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 - alpha * 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++;
    if (status->iter >= maxiter)
```

```
    break;
  }

  status->f = f;

  free(xp);
  free(g);
}

void SD_Print (double * x, uint n, Status * status) {
  uint i;

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

  printf("x = (%lf", x[0]);
  for (i = 1; i < n; i++)
    printf(",%lf", x[i]);
  printf(")\n");

  printf("Iter        = %d\n", status->iter);
  printf("f(x)        = %lf\n", status->f);
  printf("|g(x)|      = %lf\n", status->ng);
  printf("objfun calls = %d\n", status->n_objfun);
  printf("gradfun calls = %d\n", status->n_gradfun);
  printf("\n");
}
```

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 x0 = (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 CUTEst é o arquivo `c_example1main.c`:

```
#include "cutest.h"
#include "steepest_descent.h"

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

void gradfun (double * x, unsigned int n, double * g) {
  int N = n;
  int st = 0;
  CUTEST_ugr(&st, &N, x, g);
}

int MAINENTRY () {
  double *x, *bl, *bu;
  char fname[10] = "OUTSDIF.d";
  int nvar = 0, ncon = 0;
  // Don't know where these numbers came from
  int funit = 42, ierr = 0, fout = 6, io_buffer = 11;
  int i;
  Status status; //Nothing to do with CUTEST status
  int st;

  FORTRAN_open(&funit, fname, &ierr);
  CUTEST_cdimen(&st, &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);

  CUTEST_usetup(&st, &funit, &fout, &io_buffer, &nvar, x, bl, bu);

  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);

  FORTRAN_close(&funit, &ierr);

  return 0;
}
```

Note que também é necessário definir as funções `objfun` e `gradfun`. Essas funções, por sua vez, chamam as funções correspondentes em CUTEst 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 CUTEst 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 CUTEst 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 CUTEst.

```
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 CUTEst, 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 CUTEst. 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 CUTEst. 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 CUTEst 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 `typedef`s 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 `typedef`s.

Para compilar a interface CUTEst 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]
       bl: [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 carcacterísticas, 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;
           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
```

5 Contribua

Contribua

A Licença

Referências

[1] http://cuter.rl.ac.uk/cuter-www

[2] https://magi-trac-svn.mathappl.polymtl.ca/Trac/cuter