UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

MINIMIZAÇÃO DE UMA FUNÇÃO RESTRITA A PARTIR DA APLICAÇÃO DO MÉTODO DA PENALIDADE EXTERNA

ARTHUR HENRIQUE IASBECK

UBERLÂNDIA

17 DE OUTUBRO DE 2019

1. OBJETIVO

Foi proposta no presenta trabalho a minimização de $f(x)$, Eq. 1,

$$f(x) = \frac{1}{3}(x_1+1)^3 + x_2 \tag{1}$$

sujeita às seguintes restrições

$$-x_1 + 1 \leq 0 \tag{2}$$
$$-x_2 \leq 0 \tag{3}$$

2. METODOLOGIA

Uma vez que o problema proposto contém restrições de desigualdade, é necessário realizar o tratamento das mesmas para que seja possível transformá-lo num problema irrestrito. Uma vez realizado este procedimento, é possível adotar métodos como o das Variáveis Métricas, o de Newton ou o da Máxima Descida, por exemplo, para obtenção da solução do problema irrestrito.

No presente trabalho adotou-se o Método da Penalidade Externa (MPE) para tratamento das restrições. Este mesmo consiste na definição de uma nova função objetivo $\phi(x)$, Eq. 4, que inclui uma parcela de penalização que garante que $f(x) \to \infty$ caso x não atenda as restrições.

$$\phi(x) = f(x) + P(x) \tag{4}$$

No MPE defini-se a função $P(x)$ da seguinte forma,

$$P(x) = r_p(\sum_{i=1}^{m}(h_i(x))^2 + \sum_{j=1}^{k}(max(0, g_j(x)))^2) \tag{5}$$

em que m e k representam respectivamente o número de restrições de igualdade e desigualdade, e r_p é um valor que tende a infinito com o passar das iterações. Observa-se que $P(x)$ será maior que zero caso $h_i(x) \neq 0$ ou $g_j(x) > 0$ (o que faria, neste último caso, com que $max(0, g_j(x)) = g_j(x)$). Para evitar problemas de ordem numérica, é necessário que o valor de r_p seja inicialmente pequeno e cresça à medida que evolui o processo de otimização, de forma a garantir que $\phi(x^*) = f(x^*)$ sendo x^* uma solução que atenda às restrições de igualdade e desigualdade.

Aplicando a definição de $P(x)$ ao problema proposto obtém-se

$$\phi(x) = \frac{1}{3}(x_1+1)^3 + x_2 + r_p(max(0, -x_1+1)^2 + max(0, -x_2)^2) \tag{6}$$

A determinação do mínimo de $\phi(x)$ consiste num problema multi-dimensional irrestrito, ao qual foi aplicado o Método das Variáveis Métricas, que possibilita a redução da ordem do problema, que foi resolvido, em última instância, aplicando-se o Método da Seção Áurea.

A solução x^* foi obtida por meio do emprego de um código escrito no Matlab®, que foi condensado na função

```
[xOpt, fOpt, nVal, k] = varMet(x0, theta, rp0, rpInc, tol, h)
```

em que

- `xOpt`: Solução ótima obtida a partir da execução da função `varMet`.
- `fOpt`: Valor de $f(x^*)$.
- `nVal`: Número de avaliações da função objetivo.
- `k`: Número de iterações.
- `x0`: Palpite inicial para a solução.
- `theta`: Valor de θ a ser utilizado na computação do Método das Variáveis Métricas.
- `rp0`: Valor inicial de r_p.
- `rpInc`: Incremento de r_p, que é redefinido a cada iteração como `rp = rp*rpInc`.
- `tol`: Tolerância adotada para parada do algoritmo, que é executado até que a norma euclidiana entre x_{k+1} e x_k seja menor do que `tol`.
- `h`: Incremento adotado na computação numérica do gradiente da função objetivo.

Juntamente à função `varMet`, foram definidas outras três funções denominadas `funcFile.m`, `gradFile.m` e `funcFile.m`, nas quais o usuário deveria inserir, respectivamente, a função objetivo, o seu gradiente (caso o mesmo possa ser obtido analiticamente) e as restrições de igualdade e desigualdade atribuídas ao problema. Se o gradiente da função objetivo não for informado pelo usuário, será adotada a computação numérica do mesmo. Cabe ressaltar que as restrições devem ser informadas em sua forma matricial, como mostrado a seguir.

```
function [g, h] = constFile(x)

% Cada linha da matriz 'g' representa uma restricao de desigualdade
g = [-x(1) + 1
     -x(2)];

% Cada linha da matriz 'h' representa uma restricao de igualdade
h = [];
```

Executando a função `varMet` considerando as entradas apresentadas na Tab. 1, foi obtido o resultado apresentado abaixo, condizente com a solução analítica esperada.

```
x1* = 1.0000
x2* = 0.0000
f(x*) = 2.6667
Numero de avaliacoes da funcao objetivo: 253
Numero de iteracoes: 11
```

Tabela 1: Entradas atribuídas à função `varMet` para solução do problema de otimização proposto.

x0	[0 0 0]
theta	1
rp0	1
rpInc	10
tol	10^{-5}
h	10^{-10}

Os códigos empregados na obtenção da solução apresentada são introduzidos a seguir e podem ser acessados no link `https://github.com/ArthurIasbeck/OTMC4`.

1. Definição da função objetivo (`funcFile.m`).

```
function fObj = funcFile(x)

fObj = 1/3*(x(1) + 1)^3 + x(2);
```

2. Definição do gradiente da função objetivo (`gradFile.m`).

```
%#ok<*INUSD>
function df = gradFile(x)

% O gradiente da funcao objetivo, quando conhecido, deve ser inserido ...
    aqui.
% Caso o mesmo nao possa ser determinado, basta inserir 'df = []'.
df = [];
```

3. Definição das restrições do problemas (`constFile.m`).

```
function [g, h] = constFile(x)

% Cada linha da matriz 'g' representa uma restricao de desigualdade
g = [-x(1) + 1;
     -x(2)];

% Cada linha da matriz 'h' representa uma restricao de igualdade
h = [];
```

4. Código principal (`main.m`)

```
% Configuracoes previas
% Antes de executar a funcao principal 'main.m', eh preciso definir a ...
    funcao objetivo, seu gradiente (caso haja), e as restricoes (caso ...
    haja). Para tanto basta editar os arquivos 'funcFile.m', ...
    'gradFile.m', e 'constFile.m'
```

```
clc; clear; close all;

% Resolvendo o problema de otimizacao
[xOpt, fOpt, nVal, k, alfaValues] = varMet();

% Apresentando os resultados
for i = 1:length(xOpt)
    fprintf(['x',num2str(i),'* = %.4f\n'], xOpt(i))
end

fprintf('f(x*) = %.4f\n', fOpt)
fprintf('Numero de avaliacoes da funcao objetivo: %d\n', nVal)
fprintf('Numero de iteracoes: %d\n', k)

% Notas
% A seguir eh introduzida a forma completa da funcao 'varMet'
% [xOpt, fOpt, nVal, k, alfaValues] = varMet(x0, theta, rp0, rpInc, ...
    tol, h)

% Para os parametros nao informados pelo usuario serao adotados os ...
    valores padrao apresentados a seguir
% x0 = [0 0 0 ... 0]
% theta = 1
% rp0 = 1
% rpInc = 10
% tol = 1e-5
% h = 1e-10
```

5. Implementação do Método das Variáveis Métricas (`varMet.m`).

```
% Implementacao do Metodo da Variavel Metrica
function [xOpt, fOpt, nVal, k, alfaValues] = varMet(x0, theta, rp0, ...
rpInc, tol, h)

% Verificacao do palpite inicial fornecido pelo usuario
n = getOrder;
if nargin < 1 || isempty(x0)
    x0 = zeros(n,1);
else
    % Verificando a consistencia entre x0 e a funcao objetivo
    if n ≠ length(x0)
        error('O tamanho de x0 deve ser igual a ordem do problema.');
    end
end

% Verificando entradas da funcao e definindo valores padrao
if nargin < 2 || isempty(theta)
    theta = 1;
end
if nargin < 3 || isempty(rp0)
    rp0 = 1;
end
```

```
if nargin < 4 || isempty(rpInc)
    rpInc = 10;
end
if nargin < 5 || isempty(tol)
    tol = 1e-5;
end
if nargin < 6 || isempty(h)
    h = 1e-10;
end

% Verificando se o gradiente analitico foi definido
f = @(x) fObjConst(x);
dfTest = gradFile(x0);
if isempty(dfTest)
    % Gradiente numerico
    if nargin < 4 || isempty(h)
        df = @(x) grad(f,x);
    else
        df = @(x) grad(f,x,h);
    end
    numGrad = 1;
else
    % Gradiente analitico
    df = @(x) gradFile(x);
    numGrad = 0;
end

% Definicao das entradas padrao
if nargin < 2 || isempty(tol)
    tol = 1e-5;
end

if nargin < 3 || isempty(theta)
    theta = 1;
end

% Variaveis para controle de execucao
n = length(x0);
alfaValues = zeros(1,1);
k = 1;
nVal = 0;
H = eye(n);
global rp
rp = rp0;

% Implementacao do processo de otimizacao
while 1
    % Reduzir a dimensao do problema de otimizacao
    g = @(alfa) f(x0 - alfa*H*df(x0));

    % Resolver o problema de otimizacao uni-dimensional
    [alfaOpt, gOpt, nVal1] = aureaSec(g,-1,1,1e-4);

    % Atualizar a solucao otima
```

```
    x = x0 - alfaOpt*H*df(x0);

    % Armazenar dados de execucao
    alfaValues(k) = alfaOpt;
    if numGrad
        nVal = nVal + nVal1 + 1;
    else
        nVal = nVal + nVal1 + 2*n;
    end

    % OBS : Lembre-se que eh necessaria a computacao do gradiente ...
       para atualizacao de x. No entanto, se estivermos utilizando a ...
       aproximacao numerica para o gradiente, a computacao do mesmo ...
       levara, neste caso a mais avaliacoes da funcao objetivo.

    % Verificar a condicao de parada
    cp = norm(x - x0);
    if cp < tol
        break;
    end

    % Atualizacao de H (aproximacao para a inversa da Matriz Hessiana)
    p = x - x0;
    y = df(x) - df(x0);
    sigma = p'*y;
    tal = y'*H*y;
    D = ((sigma + theta*tal)/sigma^2)*(p*p') ...
    + ((theta - 1)/tal)*(H*y)*(H*y)' ...
    - (theta/sigma)*(H*y*p' + p*(H*y)');

    H = H + D;

    % Atualizar variaveis para a proxima iteracao
    x0 = x;
    k = k + 1;
    rp = rp*rpInc;
end

xOpt = x;
fOpt = f(xOpt);

% Obtencao numerica do gradiente
function df = grad(f, x, h)

% Definindo valores padrao
if nargin < 3
    h = 1e-10;
end

n = length(x);

df = zeros(n,1);
for i = 1:n
    dx = x;
```

```
    dx(i) = dx(i) + h;
    df(i) = (f(dx) - f(x))/h;
end

% Implementacao do Metodo da Secao Aurea
function [xOpt, fOpt, k] = aureaSec(f,a,b,tol)
    tal = 0.618;

    if nargin < 4
        tol = 1e-8;
    end

    alfa = a + (1 - tal)*(b - a);
    beta = a + tal*(b - a);
    fAlfa = f(alfa);
    fBeta = f(beta);

    k = 1;

    while abs(a-b) > tol
        if fBeta < fAlfa
            a = alfa;
            alfa = beta;
            fAlfa = fBeta;
            beta = a + tal*(b - a);
            fBeta = f(beta);
        elseif fAlfa ≤ fBeta
            b = beta;
            beta = alfa;
            fBeta = fAlfa;
            alfa = a + (1 - tal)*(b - a);
            fAlfa = f(alfa);
        end
        k = k + 1;
    end

    xOpt = (alfa+beta)/2;
    fOpt = f(xOpt);

% Funcao para calculo das penalidades
function P = penalty(x)

% Determinacao dos valores atribuidos a cada uma das restricoes
[g, h] = constFile(x);

% Determinacao das penalidades
Pg = sum(max(0,g).^2);
Ph = sum(h.^2);

% Calculo das penalidades
P = Pg + Ph;

% Funcao objetivo penalizada
function F = fObjConst(x)
```

```
global rp

% Computacao da funcao objetivo original
fObj = funcFile(x);

% Computacao das penalidades
P = penalty(x);

% Computacao da funcao objetivo penalizada
F = fObj + rp*P;

% Funcao para verificacao da ordem do problema com base na funcao ...
   objetivo
function n = getOrder

x = zeros(1,100);
f0 = funcFile(x);
for i = 1:100
    x(i) = 1;
    f = funcFile(x);
    if f0 == f
        break
    end
    x(i) = 0;
end
n = i - 1;
```