Capítulo 22
Otimização

Resumo do capítulo

- Neste capítulo, otimização refere-se ao processo de determinar os pontos onde uma função assume determinado valor, ou assume valores extremos.
- Abordaremos aqui funções disponíveis no **MATLAB** básico, mas há uma grande variedade de funções disponíveis na *optimization toolbox*. Veja *Help –> MATLAB –> Mathematics –> Function Functions –> Minimizing Functions and Finding Zeros*.
- Seja $f(x)$ uma função. Dado y, determinar x tal que $y = f(x)$ pode ser complicado, principalmente quando não sabemos calcular $f^{-1}(x)$. Nestes casos precisamos estimar o valor de x por um procedimento iterativo, chamado de *zero finding* (porque buscamos o zero da função $h(x) = y - f(x)$).
- Abordaremos somente o caso unidimensional; dimensões maiores requerem o uso de *toolboxes*.

A função *fzero*

- O máximo de uma função $f(x)$ é dado pelo mínimo de $-f(x)$. Porisso, podemos nos ater aos métodos para determinação de valores extremos mínimos; isto é com *algoritmos de minimização*.

- No **MATLAB** podemos usar a função $fzero$ para encontrar o zero de uma função unidimensional.

- O algoritmo usado é uma combinação do método da bisseção e da interpolação quadrática inversa.

Exemplo

- Vamos exemplificar o uso da função $fzero$ para minimizar a a função $humps$:

```
>> x = linspace(-.5,1.5);
>> y = humps(x);
>> plot(x,y)
>> grid on
>> title('Humps Function')
```

- O arquivo **humps.m** calcula a função:

$$humps(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6,$$

cujos zeros estão próximos de $x = -0.2$ e $x = 1.3$.

Usando a função $fzero$

```
>> format long % para exibir mais dígitos
>> x = fzero('humps', 1.3) % calculando o zero próximo a 1.3
x =
   1.29954968258482

>> disp(humps(x)) % verificando o valor de humps(x)
     0

>> [x, value] = fzero('humps',-0.2) % calculando o zero próximo
% a -0.2 e já verificando o valor de humps(x)
x =
  -0.13161801809961
value =
     8.88178419700125e-16
```

- A função $fzero$ fornece apenas um zero, aquele que é mais próximo da aproximação inicial. Assim, se houver mais de um zero é preciso fazer uma chamada a $fzero$ para cada um deles, com diferentes aproximações iniciais.

Observações a respeito de $fzero$

- A primeira ação da função $fzero$ é procurar por uma mudança de sinal na função em cada lado da aproximação inicial. Quando encontra, é formado um intervalo onde a função efetuará a busca pelo zero. Note que se a função é contínua esta troca de sinal indica que houve cruzamento com o eixo X. O valor retornado é o mais próximo do ponto de cruzamento.
- Podemos chamar a função $fzero$ com um intervalo, mas devemos garantir que os extremos do intervalo possuem sinais diferentes:

```
>> [x,valor] = fzero('humps',[-2 0])
x =
  -0.13161801809961
value =
   0
>> [x,valor] = fzero('humps',[0 1.2])
??? Error using ==> fzero
The function values at the interval endpoints must differ in sign
```

A função a ser pesquisada

- A função para a qual queremos encontrar um ou mais zeros pode ser fornecida como:
  - string que identifica o arquivo .m que a contém (como no exemplo anterior);
  - function handle;

```
>> disp(fzero(@humps,[-2 0]))
 -0.13161801809961
```

  - expressão em string.

```
>> humpsstr ='1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6';
>> disp(fzero(humpsstr,[-2 0]))
 -0.13161801809961
```

  - function *in line*;

```
>> hinline = inline(humpsstr);
>> disp(fzero(hinline,[-2 0]))
 -0.13161801809961
```

A função a ser pesquisada

- As quatro formas anteriores de definição da função se aplicam a todas as funções de otimização que discutiremos;
- Todas estas formas são equivalentes em termos de resultados;
- Estas formas **não** são equivalentes em termos de velocidade de execução:
  - **Mais rápido:** escrever o arquivo M e usar uma function handle apontando para este arquivo;
  - **Intermediário:** usar o arquivo M diretamente;
  - **Mais lento:** usar a função in line ou expressão em string.
    - No caso de uma expressão em string o dado é convertido em uma função in line antes da execução. Logo, os dois casos tornam-se equivalentes, em um deles a conversão para função in line é feita pela função $fzero$, no outro pelo usuário.

Parâmetros para $fzero$

- A função $fzero$, bem como todas as funções de otimização, possuem diversos parâmetros definíveis.
- As funções $optimset$ e $optimget$ são usadas para estabelecer e recuperar parâmetros para todas as funções de otimização. Isto porque todas estas funções, mesmo as da *optimization toobox*, compartilham o mesmo formato de controle de parâmetros.
- No caso da função $fzero$ são usados dois parâmetros: $Display$ e $TolX$. O primeiro controla a quantidade de detalhes fornecidos durante a execução e o segundo estabelece uma limite de tolerância para a resposta final. Vamos ver alguns exemplos de uso.

Exemplos de uso de parâmetros

```
>> options = optimset('Display','iter'); % retorno de optimset é uma estrutura
>> [x,value] = fzero(@humps, [-2 0], options)
 Func-count      x          f(x)         Procedure
    1            -2      -5.69298        initial
    2             0       5.17647        initial
    3       -0.952481      -5.07853        interpolation
    4       -0.480789      -3.87242        interpolation
    5       -0.240394      -1.94304        bisection
    6       -0.120197       0.28528        bisection
    7       -0.135585     -0.0944316         interpolation
    8       -0.131759    -0.00338409         interpolation
    9       -0.131618    1.63632e-06         interpolation
   10       -0.131618   -7.14819e-10         interpolation
   11       -0.131618              0       interpolation
Zero found in the interval: [-2, 0].
x =  -0.13161801809961
value =  0
>> options = optimset('Display','none');
>> [x,value] = fzero(@humps, [-2 0], options)
x = -0.13161801809961
value = 0
```

Exemplos de uso de parâmetros

```
>> options = optimset('Display','final','TolX',0.1);
>> [x,value] = fzero(@humps, [-2 0], options)
Zero found in the interval: [-2, 0].
x =
  -0.24039447250762
value =
  -1.94303825972565

>> options = optimset('Display','iter','TolX',0.1);
>> [x,value] = fzero(@humps, [-2 0], options)
 Func-count      x          f(x)         Procedure
    1            -2      -5.69298        initial
    2             0       5.17647        initial
    3      -0.952481      -5.07853        interpolation
    4      -0.480789      -3.87242        interpolation
    5      -0.240394      -1.94304        bisection
Zero found in the interval: [-2, 0].
x =
  -0.24039447250762
value =
  -1.94303825972565
```

Minimização

- Usamos minimização quando queremos determinar os máximos e os mínimos de uma função.
- Matematicamente, isto é feito analiticamente, determinado-se os pontos onde a derivada da função é zero (veja o gráfico da função `humps`).
- Em muitos casos não é possível, ou não é viável, determinar os pontos em que a derivada é nula. Nestes casos precisamos recorrer a métodos numéricos para determinar os extremos da função.
- O **MATLAB** fornece, na sua parte básica, duas funções para determinação de mínimos de funções:
  - $fminbnd$: minimização em uma dimensão;
  - $fminsearch$: minimização em dimensão n.
- Para determinação do máximo, fazemos a busca pelo mínimo de $-f(x)$. O valor retornado é o negativo do valor real.

Minimização em uma dimensão

- Usamos a função $fminbnd$ para encontrar o mínimo de funções unidimensionais. Esta utiliza uma combinação de busca por seções áureas e interpolação parabólica.
- Vamos exemplificar o uso da função $fminbnd$ usando a função **humps**. Note que há um máximo próximo de 0.3 e um mínimo próximo de 0.6.

```
>> [xmin, valor] = fminbnd('humps',0.5,0.8)
xmin =                    valor =
   0.63700821196362          11.25275412587769
>> [xmin, valor] = fminbnd('-humps(x)',0.2, 0.4)
xmin =                    valor =
   0.30036413790024         -96.50140724387050
```

- No último exemplo **'-humps(x)'** é uma expressão que identifica o que deve ser minimizado. Poderíamos também modificar o arquivo .m.
- Podemos modificar e recuperar as opções com $optimset$ e $optimget$. Para detalhes veja *help on line*.

Minimização em dimensão n

- É feita pela função $fminsearch$, que tenta encontrar o mínimo de uma função escalar $f(x)$ de argumento vetorial x.
- A função $fminsearch$ implementa o algoritmo de busca simplex, de Nelder-Mead que modifica os componentes de x para atingir o mínimo de $f(x)$.
  - para funções suaves: não tão eficiente;
  - Não requer informações sobre o gradiente (cálculo pode ser caro).
  - Tende a ser mais robusto para funções que não são suaves.

Minimização em dimensão n

- Nosso exemplo ilustrativo será a *função de Rosenbrock*, também conhecida como função banana:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- Veja o gráfico desta função:

Arquivo: mm2202.m

```
x = [-1.5:0.125:1.5];
y = [-.6:0.125:2.8];
[X,Y] = meshgrid(x,y) ;
Z = 100.*(Y-X.*X).^2 + (1-X).^2;
mesh(X,Y,Z)
hidden off
xlabel('x(1)'), ylabel('x(2)')
title('Figure 22.2: Banana Function')
hold on
plot3(1,1,1,'k.','markersize',20)
hold off
%(mm2202.m plot)
```

Usando $fminsearch$

- "Vetorizando" a entrada: $x(1) \leftarrow x_1$ e $x(2) \leftarrow x_2$. Isto pode ser feito com a seguinte função M: ($banana.m$)

```
function f=banana(x)
% Função banana de Rosenbrock
f=100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
```

- Vamos agora usar a função $fminsearch$:

```
>> format long
>> [xmin, fxmin,conv,saida] = fminsearch('banana',[-1.9,2])
xmin =
   1.00001666889480   1.00003447386277
fxmin =                                  conv =
    4.06855135063419e-10                    1
saida =
    iterations: 114
     funcCount: 210
     algorithm: 'Nelder-Mead simplex direct search'
```

Função $fminsearch$ - adicionais

- Os quatro parãmetros de saída do exemplo anterior são:
  - $xmin$: o ponto de mínimo;
  - $fxmin$: o valor da função no ponto encontrado;
  - $conv$: indica o sucesso da busca;
  - $sada$: estrutura com as estatísticas do algoritmo.
- A função $fminsearch$ também aceita uma estrutura de opções como as funções de otimização que já vimos.
- As opções que podem ser estabelecidas para esta função são: $Display$, $MaxFunEvals$, $MaxIter$, $TolFun$ e $TolX$ (estão especificadas na última tabela da seção 22.3).
- Reexecute a função usando as opções abaixo e observe o resultado:

```
>> opcoes = optimset('Display','none','TolFun',1e-8,'TolX',1e-8);
>> [xmin, fxmin,conv,saida] = fminsearch('banana',[-1.9,2], opcoes)
```

Considerações finais

- Resoluções numéricas trabalham com suposições a respeito da função que passará pelo processo iterativo. Queremos garantir convergência e em um tempo viável.
- No **MATLAB** havendo algum erro, a busca pode se encerrar sem que nada seja retornado, ou quando retornam dados, os resultados não são os desejados.
  - Comece com uma boa aproximação inicial;
  - Se componentes da solução possuem ordens de magnitude muito diferentes considere a possibilidade de escaloná-los;
  - Se necessário, quebre o problema em problemas menores;
  - Estude sua função para ter certeza de que ela não retorna números complexos, $NaNs$ ou Inf;
  - Evite funções descontínuas;
  - Acrescente restrições ao intervalo ao qual x pertence adicionando alguma penalidade à função iterada.