MATLAB - Comando `fminunc`

Universidade do Minho, Escola de Engenharia, Departamento de Produção e Sistemas

Ana Maria A. C. Rocha

FMINUNC - usa o método quasi-Newton ou o método 'trust-region' (regiões de confiança).

```
[x,fval,exitflag,output] = fminunc(fun,x0,options,...)
```

O comando `fminunc` tem como argumentos de entrada:

`fun` - é a função objetivo a minimizar

`x0` - é a aproximação inicial

`options` - opções para o controlo do processo de otimização.

O comando `fminunc` pode ter como argumentos de saída:

`x` - é a solução ótima do problema.

`fval` - é o valor da função objetivo em `x`.

`exitflag` - descreve valores de saída do `fminunc`	
1	a magnitude do vetor gradiente é menor que a tolerância de otimalidade `OptimalityTolerance`.
2	um alteração em x é menor que a tolerância `TolX`.
3	uma alteração na função objetivo é menor que a tolerância `TolFun`.
5	um previsto decréscimo na função objetivo é menor que a tolerância `TolFun`.
0	o número de iterações excedeu o `MaxIter` ou o `MaxFunEvals`.
-1	indica que a função não convergiu para uma solução.
-3	o problema parece ser não limitado.

`output` - estrutura que contém informação acerca do processo de otimização	
`output.iterations`	indica o número de iterações realizadas
`output.funcCount`	indica o número de cálculos da função
`output.firstorderopt`	indica o valor da medida de otimalidade de 1^a ordem (norma do gradiente na solução em `x`)
`output.algorithm`	indica o algoritmo usado
`output.cgiterations`	indica o número total de iterações PCG (apenas para o algoritmo 'trust-region')
`output.lssteplength`	indica o comprimento do passo da procura direta relativamente à direção de procura (apenas para o algoritmo 'quasi-newton')
`output.stepsize`	indica o comprimento do passo final em `x`
`output.message`	indica a mensagem de saída.

Para ver as opções disponíveis fazer:

- `optimoptions('fminunc')`
- `optimoptions('fminunc','algorithm','quasi-Newton')`
- `optimoptions('fminunc','algorithm','trust-region')`

`Options` - definição de parâmetros	
`Algorithm`	Escolhe o algoritmo a usar: 'quasi-newton' (default) ou 'trust-region' (requer que seja dado o gradiente).
`DerivativeCheck`	Compara as derivadas fornecidas pelo utilizador com a sua aproximação por diferenças finitas `on -` mostra o valor da discrepância entre valores calculados e fornecidos `off -` (default) não mostra
`Display`	Nível de apresentação `off -` não apresenta nada `iter -` resultados em cada iteração e uma mensagem de saída por defeito `iter-detailed -` resultados em cada iteração e uma mensagem técnica de saída `notify -` resultado se a função não convergir e uma mensagem de saída por defeito `notify-detailed -` resultado se a função não convergir e uma mensagem técnica de saída `final -` (default) apresenta apenas o resultado final e uma mensagem de saída por defeito `final-detailed -` apresenta apenas o resultado final e uma mensagem técnica de saída
`FunValCheck`	Verifica se os valores da função objetivo são válidos `on -` apresenta erro se complex, Inf ou NaN. `off -` (default) não apresenta erro
`MaxFunEvals`	Número máximo de cálculos da função - (default = 100*numberofvariables)
`MaxIter`	Número máximo de iterações - (default = 400)
`OptimalityTolerance`	Condição de paragem baseada na medida de otimalidade de 1^a ordem - (defaulta = 1e-6)
`OutputFcn`	Especifica uma ou mais funções que o processo de otimização pode invocar, em cada iteração.
`PlotFcn`	Representa graficamente várias medidas do progresso do algoritmo, ao longo das iterações. `@optimplotx -` desenha o ponto atual. `@optimplotfunccount -` desenha o número de cálculos da função objetivo. `@optimplotfval -` desenha o valor da função objetivo. `@optimplotstepsize -` desenha o comprimento do passo. `@optimplotfirstorderopt -` desenha a medida de otimalidade de 1^a ordem.
`GradObj`	Gradiente da função objetivo definido pelo utilizador. `on -` usa o gradiente definido pelo utilizador. `off -` (default) aproxima o gradiente por diferenças finitas
`TolFun`	Tolerância de paragem relativamente à função objetivo - (default = 1e-6)
`TolX`	Tolerância de paragem relativamente a x - (default = 1e-6)
`Hessian`	Modo de definição da Hessiana. `on -` usa a Hessiana definida pelo utilizador. `off -` (default) aproxima a Hessiana por diferenças finitas
`HessUpdate`	Método para escolher a direção de pesquisa no algoritmo 'quasi-Newton'. `bfgs -` (default) aproxima a Hessiana pela fórmula BFGS `lbfgs -` aproxima a Hessiana pela Low-memory BFGS (problemas de grandes dimensões) `dfp -` aproxima a Hessiana pela fórmula DFP `steepdesc -` descida máxima

Algoritmos de otimização

- **Quasi-Newton Algorithm**

  O algoritmo quasi-Newton usa a fórmula de atualização BFGS (por defeito) para aproximar a matriz Hessiana e um método de procura linear cúbica.

- **Trust Region Algorithm**

  O algoritmo 'trust-region' requer que seja fornecido o vetor gradiente da função objetivo. É baseado no método de Newton 'interior-reflective', cuja cada iteração envolve a solução aproximada de um sistema linear de grandes dimensões usando o método dos gradientes conjugados pré-condicionados (PCG).

Síntese

- Sem mudar qualquer opção ⇒ **Método quasi-Newton** (matriz Hessiana aproximada pela fórmula BFGS)
- Usando GradObj = 'on' ⇒ **Método 'trust-region'**

1. Considere o seguinte problema de otimização sem restrições

$$f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2.$$

Calcule o seu mínimo usando o algoritmo quasi-Newton. Inicie o processo iterativo com o ponto $(1, 0)$.

(a) Sem usar qualquer opção.

```
[x,fval,exitflag,output]=fminunc(@QN,[1,0])
function [f] = fun(x)
    f=@(x)x(1)^2+x(2)^2-x(1)*x(2);
end
```

Nota: por defeito usa o método quasi-Newton e a fórmula de atualização BFGS.

(b) Visualizando os resultados obtidos em cada iteração.

```
op=optimset('Display','iter');
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

(c) Representando graficamente os valores da função objetivo ao longo das iterações.

```
op=optimset('PlotFcns',@optimplotfval);
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

(d) Usando como tolerância de paragem $TolX = 10^{-10}$ e $TolFun = 10^{-12}$.

```
op=optimset('TolX',1e-10,'TolFun',1e-12);
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

(e) usando a fórmula de atualização DFP.

```
op=optimset('hessupdate','dfp')
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

(f) Usando como tolerância de paragem 20 iterações.

```
op=optimset('MaxIter',20);
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

(g) Usando como tolerâncias de paragem $TolX = 10^{-4}$ e 50 como máximo de iterações.

```
op=optimset('TolX',1e-3,'MaxIter',50);
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

(h) Usando como tolerâncias de paragem $TolFun = 10^{-5}$, 50 como máximo de iterações e a fórmula de atualização DFP.

```
op=optimset('TolFun',1e-5,'MaxIter',50,'HessUpdate','DFP');
[x,fval,exitflag,output]=fminunc(@QN,[1,0],op)
function [ f ] = QN( x )
    f=x(1)^2+x(2)^2-x(1)*x(2);
end
```

2. O lucro da colocação de um sistema elétrico é dado por

$$\mathcal{L}(x_1, x_2) = 20x_1 + 26x_2 + 4x_1x_2 - 4x_1^2 - 3x_2^2$$

em que x_1 e x_2 designam, respetivamente, o custo da mão de obra e do material.

$$\max L(x_1, x_2) = -\min(-\mathcal{L}(x_1, x_2))$$

$$\min -(20x_1 + 26x_2 + 4x_1x_2 - 4x_1^2 - 3x_2^2)$$

Calcule o lucro máximo usando o método quasi-Newton, iniciando o processo com $x^{(1)} = (0, 0)$ e:

(a) usando o método quasi-Newton e a fórmula de atualização da Hessiana BFGS

```
[x,fval,exitflag,output]=fminunc(@QN1,[0,0])
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

(b) usando a fórmula de atualização DFP (da matriz Hessiana)

```
op=optimset('hessupdate','dfp')
[x,fval,exitflag,output]=fminunc(@QN1,[0,0],op)
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

(c) usando a opção da representação gráfica dos valores da função objetivo, ao longo das iterações (`optimset('PlotFcns',@optimplotfval)`).

```
op=optimset('PlotFcn',@optimplotfval);
[x,fval,exitflag,output]=fminunc(@QN1,[0,0],op)
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

(d) usando $tolX = 10^{-2}$ e a fórmula de atualização BFGS

```
op=optimset('tolx',1e-2);
[x,fval,exitflag,output]=fminunc(@QN1,[0,0],op)
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

(e) usando $tolX = 10^{-2}$ e a fórmula de atualização DFP

```
op=optimset('tolx',1e-2,'hessupdate','dfp');
[x,fval,exitflag,output]=fminunc(@QN1,[0,0],op)
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

(f) usando $TolFun = 10^{-12}$ e a fórmula de atualização BFGS

```
op=optimset('TolFun',1e-12);
[x,fval,exitflag,output]=fminunc(@QN1,[0,0],op)
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

(g) usando no máximo 2 iterações, a fórmula de atualização BFGS e visualizando os resultados obtidos em cada iteração.

```
op=optimset('MaxIter',2,'Display','iter');
[x,fval,exitflag,output]=fminunc(@QN1,[0,0],op)
function [f] = QN1(x)
    f=-(20*x(1)+26*x(2)+4*x(1)*x(2)-4*x(1)^2-3*x(2)^2);
end
```

3. A soma de três números (x_1, x_2 e x_3) positivos é igual a 40. Determine esses números de modo que a soma dos seus quadrados seja mínima. Use a relação da soma para colocar x_3 em função das outras 2 variáveis.

   Sabemos que $x_1 + x_2 + x_3 = 40$ e queremos minimizar $x_1^2 + x_2^2 + x_3^2$, logo fazendo $x_3 = 40 - x_1 - x_2$

$$\min_{x \in \mathbb{R}^2} f(x_1, x_2) \equiv x_1^2 + x_2^2 + (40 - x_1 - x_2)^2$$

   Resolva o problema, iniciando o processo iterativo com a aproximação inicial $(x_1, x_2)^{(1)} = (10, 10)$ e:

   (a) usando o método quasi-Newton e a fórmula de atualização da Hessiana BFGS

```
[x,fval,exitflag,output]=fminunc(@QN1,[10,10])
function [f] = QN1(x)
    f=x(1)^2+(2)^2+(40-x(1)-x(2))^2;
end
```

   (b) usando o método quasi-Newton e a fórmula de atualização DFP

```
op=optimset('hessupdate','dfp')
[x,fval,exitflag,output]=fminunc(@QN1,[10,10],op)
function [f] = QN1(x)
    f=x(1)^2+(2)^2+(40-x(1)-x(2))^2;
end
```

   (c) usando o método quasi-Newton, fórmula BFGS e $tolX = 10^{-4}$

```
op=optimset('tolx',1e-4);
[x,fval,exitflag,output]=fminunc(@QN1,[10,10],op)
function [f] = QN1(x)
    f=x(1)^2+(2)^2+(40-x(1)-x(2))^2;
end
```

   (d) usando o método quasi-Newton, $tolfun = 10^{-8}$, $tolx = 10^{-12}$ e a fórmula de atualização DFP

```
op=optimset('tolfun',1e-8,'tolx',1e-12,'hessupdate','dfp');
[x,fval,exitflag,output]=fminunc(@QN1,[10,10],op)
function [f] = QN1(x)
    f=x(1)^2+(2)^2+(40-x(1)-x(2))^2;
end
```

4. Suponha que pretendia representar um número A positivo como uma soma de três números positivos x_1, x_2 e x_3. Para $A = 180$, determine esses números de tal forma que o seu produto seja o maior possível. Resolva o problema a partir da aproximação inicial $(x_1, x_2)^{(1)} = (55, 55)$.

   Sabemos que $A = 180 = x_1 + x_2 + x_3$ e queremos maximizar $x_1 x_2 x_3$, logo fazendo $x_3 = 180 - x_1 - x_2$

$$\max_{x \in \mathbb{R}^2} f(x_1 x_2) \equiv x_1 x_2 (180 - x_1 - x_2) \;\Leftrightarrow\; \min_{x \in \mathbb{R}^3} -(x_1 x_2 (180 - x_1 - x_2))$$

   (a) usando o método quasi-Newton, fórmula BFGS e $tolfun = 10^{-5}$

```
op=optimset('tolfun',1e-5);
[x,fval,exitflag,output]=fminunc(@QN,[55,55],op)
function [f] = QN(x)
    f=-(x(1)*x(2)*(180-x(1)-x(2)));
end
```

   (b) usando o método quasi-Newton, $tolfun = 10^{-8}$, $tolx = 10^{-12}$ e a fórmula de atualização DFP

```
op=optimset('tolfun',1e-8,'tolx',1e-12,'hessupdate','dfp');
[x,fval,exitflag,output]=fminunc(@QN,[55,55],op)
function [f] = QN(x)
    f=-(x(1)*x(2)*(180-x(1)-x(2)));
end
```