简介： n维空间中，由n+1个顶点，可以组成“最简单”的图形，叫单纯形。  
NM法就是先构造一个初始的，包含给定点的单纯形。 在以上三种手段失效的时候，使用收缩。 （半径的定义可以有很多，比如两两点的距离， 只要当“半径”趋于0的时候，该单纯形趋于一个点即可）相关资料：NelderMeadProof.pdf 虽然只提到二维的情况，不过可以推广到n维。  
主要通过上面的几个图例，知道四种手段即可：  
反射(reflection)，扩展(expasion)，压缩(contraction)，收缩(shrink) 看懂了这四个基本手段后，就不用去搜其他资料了。 最好的方法就是直接找matlab里面的fminsearch函数。 这套代码里面，允许对以上四种操作的尺寸做定制。 这个距离在fminsearch里面可以手工定制：  
% Initialize parameters  
rho = 1; chi = 2; psi = 0.5; sigma = 0.5;  
同理还可以改变其他几个代码) 单纯形变换流程： 看流程之前，先确保一提到上面四个操作，就能反映出相关的点之间的关系。 最好点：best  
次差点：soso  
最差点：worst  
反射点：r  
扩展点：e  
内压缩点：c1(center和worst之间）  
外压缩点：c2(center和r之间） 1。如果反射点值小于best，那么考虑扩展点e，选r和e中小者去替换worst  
2。如果反射点值小于soso，用r如替换worst  
3。非以上两种情况，考虑压缩点  
3.1。worst比r小，那么考虑c1，如果c1比worst小，选c1替换，否则考虑收缩  
3.2。r比worst小，那么考虑c2，如果c2比r小，选c2替换，否则考虑收缩  
4。如果在第3步中确定需要收缩，那么将所有点向best方向按比例收缩 剩下的内容代码里面比较详细了。代码： 下面这个就是看了matlab实现的fminsearch之后，稍加改写的一个版本。  
去掉了一些通用性的定制变量，一些繁琐的参数判断，尽量写了详细的注释。 function [x , fval , flag] = nm\_min(fun , x0 , max\_time , eps)  
%realization of Nelder-Mead Simplex  
%[x fval flag] = nm\_min(f , x0 , max\_time , eps)  
%max\_time:最大迭代次数，默认10000  
%eps:精度，默认1e-5 %参数检查  
if nargin < 2,  
   error('请至少传入函数和初始点');  
end %默认值设置  
if nargin < 3  
    max\_time = 10000 ;  
end  
if nargin < 4  
    eps = 1e-5 ;     
end n = length(x0) ;%变量个数  
x0 = x0(:) ;%把x0变成列向量 %vx是单纯形矩阵，n行n+1列  
%即n维空间中的n+1个点  
vx = x0 ;  
%vf是函数值，对应每个列向量  
vf = feval\_r(fun , x0) ; %构造初始单纯形 for i = 1:n      
    x = x0 ;  
    if abs(x(i)) < eps  
        x(i) = x(i) + 0.005 ;%该维度上为0时，人工指定加上一定值  
    else  
        x(i) = x(i) \* 1.05 ;  
    end  
    vx = [vx , x] ;  
    vf = [vf , feval\_r(fun , x)] ;  
end %排序，做迭代准备  
[vf index] = sort(vf) ;  
vx = vx(:,index) ; %开始迭代  
while max\_time > 0     if max(max(abs(vx(:,2:n+1)-vx(:,1:n)))) < eps  
        break  
    end     best = vx(: , 1) ;  
    fbest = vf(1) ;  
    soso = vx(: , n) ;  
    fsoso = vf(n) ;  
    worst = vx(: , n+1) ;  
    fworst = vf(n+1) ;         
    center = sum(vx(: , 1:n) , 2) ./ n ;  
    r = 2 \* center - worst ;%反射点  
    fr = feval\_r(fun , r) ;  
    if fr < fbest         e = 2 \* r - center ; %扩展点  
        fe = feval\_r(fun , e) ;         if fe < fr  
            vx(: , n+1) = e ;  
            f(: , n+1) = fe ;  
        else  
            vx(: , n+1) = r ;  
            vf(: , n+1) = fr ;  
        end  
    else  
        if fr < fsoso             vx(: , n+1) = r ;  
            vf(: , n+1) = fr ;             
        else             %当压缩点无法得到更优值的时候，考虑收缩  
            shrink = 0 ;  
            if fr < fworst  
                %由于r点更优所以  
                %向r点的方向找压缩点  
                c = ( r + center ) ./ 2 ;  
                fc = feval\_r(fun , c) ;  
                if fc < fr  
                    %确定从r压缩向c可以改进  
                    vx(: , n+1) = c ;  
                    vf(: , n+1) = fc ;  
                else                     shrink = true ;  
                end  
            else  
                c = (worst + center) ./ 2 ;  
                fc = feval\_r(fun , c) ;  
                if fc < fr  
                    %确定从r压缩向c可以改进  
%（2009.7.22标注，这里应该是搞错了，回去再认真看一下）  
                    vx(: , n+1) = c ;  
                    vf(: , n+1) = fc ;  
                else                     shrink = 1 ;  
                end                 
            end%fr < fworst  
            if shrink  
                %压缩点并非更优，考虑所有点向best收缩  
                for i = 2:n+1  
                    vx(: , i) = ( vx(i) + best ) ./ 2 ;  
                    vf(: , i) = feval\_r(fun , vx(: , i)) ;  
                end  
            end%shrink  
        end%fr < fsoso  
    end%fr < fbest  
    [vf index] = sort(vf) ;  
    vx = vx(:,index) ;     
    max\_time = max\_time - 1 ;  
end%while max\_time > 0 x = vx(: , 1) ;  
fval = vf(: , 1); if max\_time > 0  
    flag = 1 ;  
else  
    flag = 0 ;  
end 示例： f = inline('x(1).^2+(x(2)-2).^2')  
[x fval flag] = nm\_min(f , [0;0]) x =  
         0  
    2.0000  
fval =  
8.0779e-028  
flag =  
     1 flag=1表示找到解，[0;2.0000]，相应的函数值为8.0779e-028(即0）