The Two-dimensional Nelder-Mead Method

In this exercise, you will implement the two-dimensional Nelder-Mead method to minimize a user-defined function f within a user-defined tolerance t and

Write the function `NM2.r` that approximates the minimum of a function of two variables. The goal of the Nelder-Mead method is to remove the worst point in a simplex of three points in the *xy*-plane and replace it with a better point.

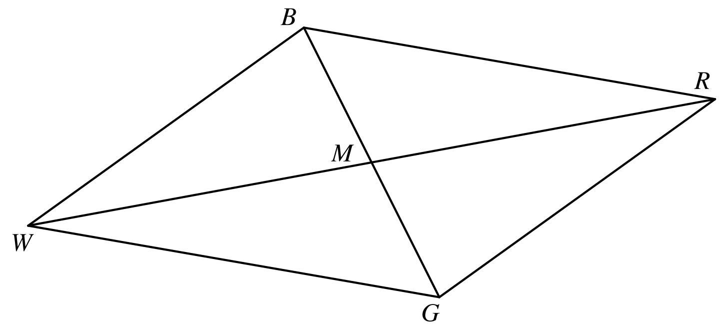
Inputs:	f,	the function you are trying to minimize.
	P,	one approximation of the minimum.
	Q,	a second approximation of the minimum.
	R,	a third approximation of the minimum.
	t,	the tolerance for the relative error.
Outputs:	X,	the approximation for the minimizer of the function
	Z,	the approximation of the minimum of the function.

Note that f is now a function of two variables, x and y. Also, we now have three different initial points: P, Q and R, which are vectors in **R**.

First of all, we have to initialize the algorithm labeling the initial points P, Q and R as B, for "Best," G, for "Good," and W, for "Worst." These labels satisfy the inequalities $f(B) \leq f(G) \leq f(W)$.

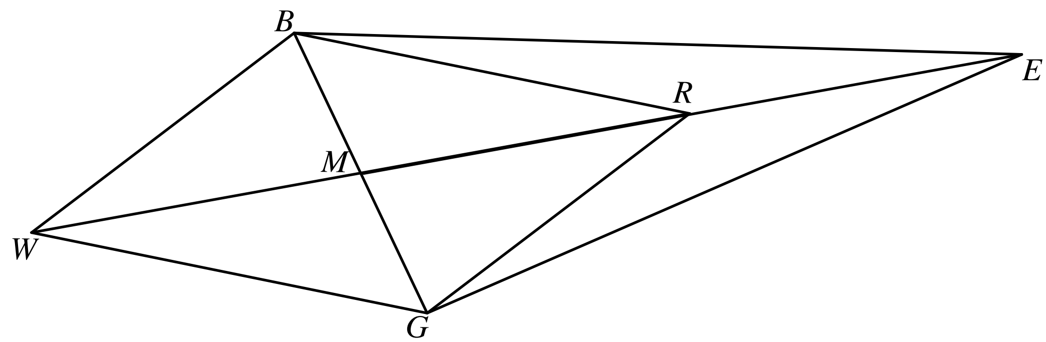
Since the initial points have been labeled, we now describe one iteration of the algorithm. Each iteration can be described as follows:

1. Midpoint
   Calculate the midpoint M of the line segment BG: $M = (B+G)/2$.

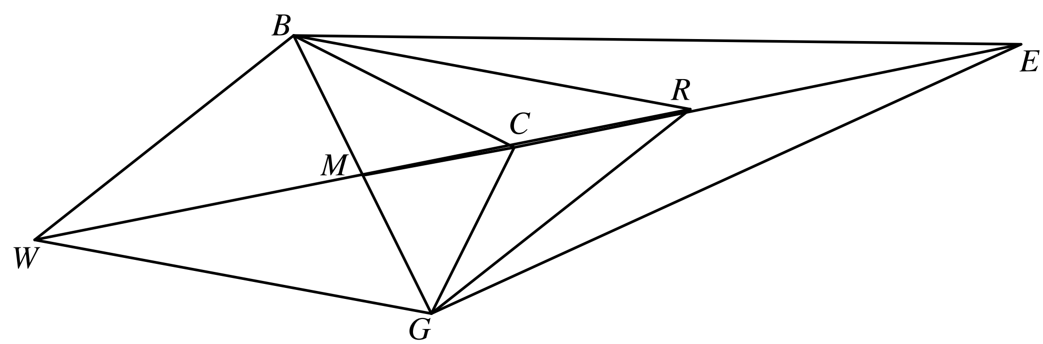


2. Reflection
   Calculate the *reflection point* $R = 2M - W$.
3. Expansion
   If $f(R) < f(B)$ (that is, if R is better than the best point), calculate the *expansion point* $E = 2R - M$.

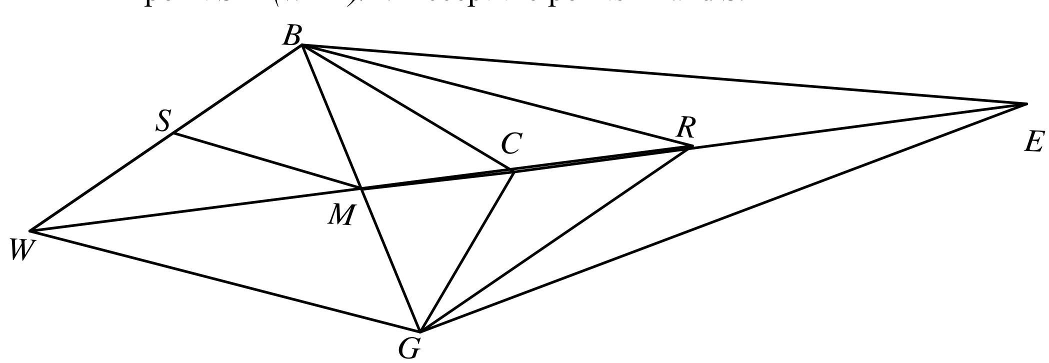
a. If $f(E) < f(R)$ (that is, if E is even better than R), accept Points B, G, and E. Relabel.
b. If $f(E) \geq f(R)$ (that is, E is no better than R), then accept R.



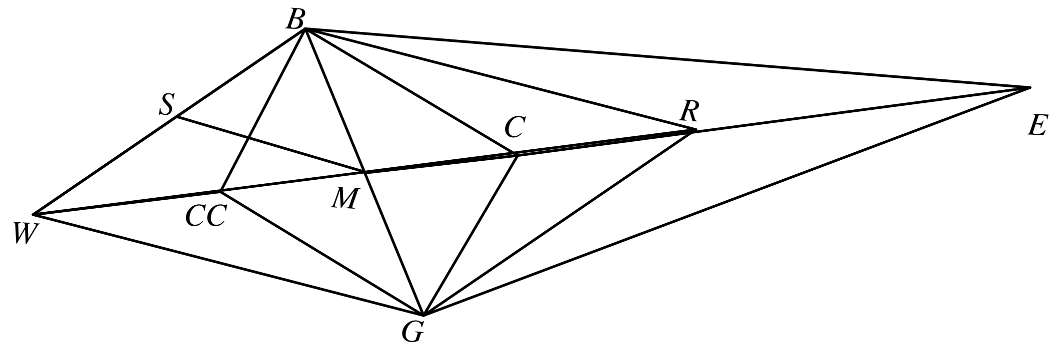
4. Acceptance
   If $f(B) \leq f(R) < f(G)$ (that is, R is better than G but is not as good as B), accept R.
5. Outside Contraction
   If $f(G) \leq f(R) < f(W)$ (that is, R is better than W but is not as good as G), calculate the *outside contraction point* $C = (M+R)/2$.
   a. If $f(C) \leq f(R)$, accept C.



b. On the other hand, if $f(R) < f(C)$, then *shrink* the triangle by calculating the point $S = (W+B)/2$. Accept the points M and S.



6. Inside Contraction
   If $f(W) \leq f(R)$ (that is, R is worse than the worst point W), calculate the *inside contraction point* $CC = (W+M)/2$.



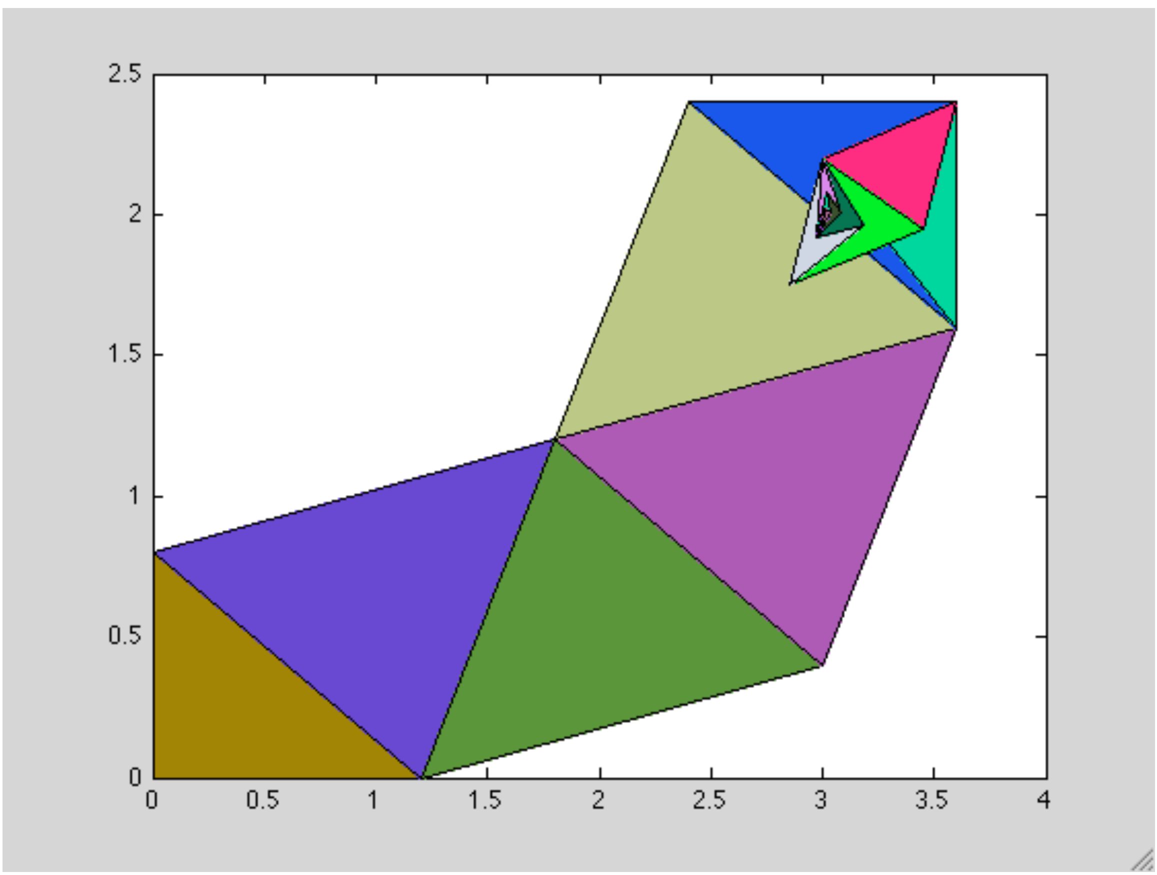
   a. If $f(CC) < f(W)$, accept CC.
   b. Otherwise, *shrink* the triangle by calculating the point $S = (W+B)/2$. Accept the points M and S.

When should we quit? We will stop the algorithm when the length of the longest side of the triangle is less than the user-defined tolerance. That is, steps 1 through 6 above are inside a *while* loop. The function repeats this process until the exiting criterion is satisfied.

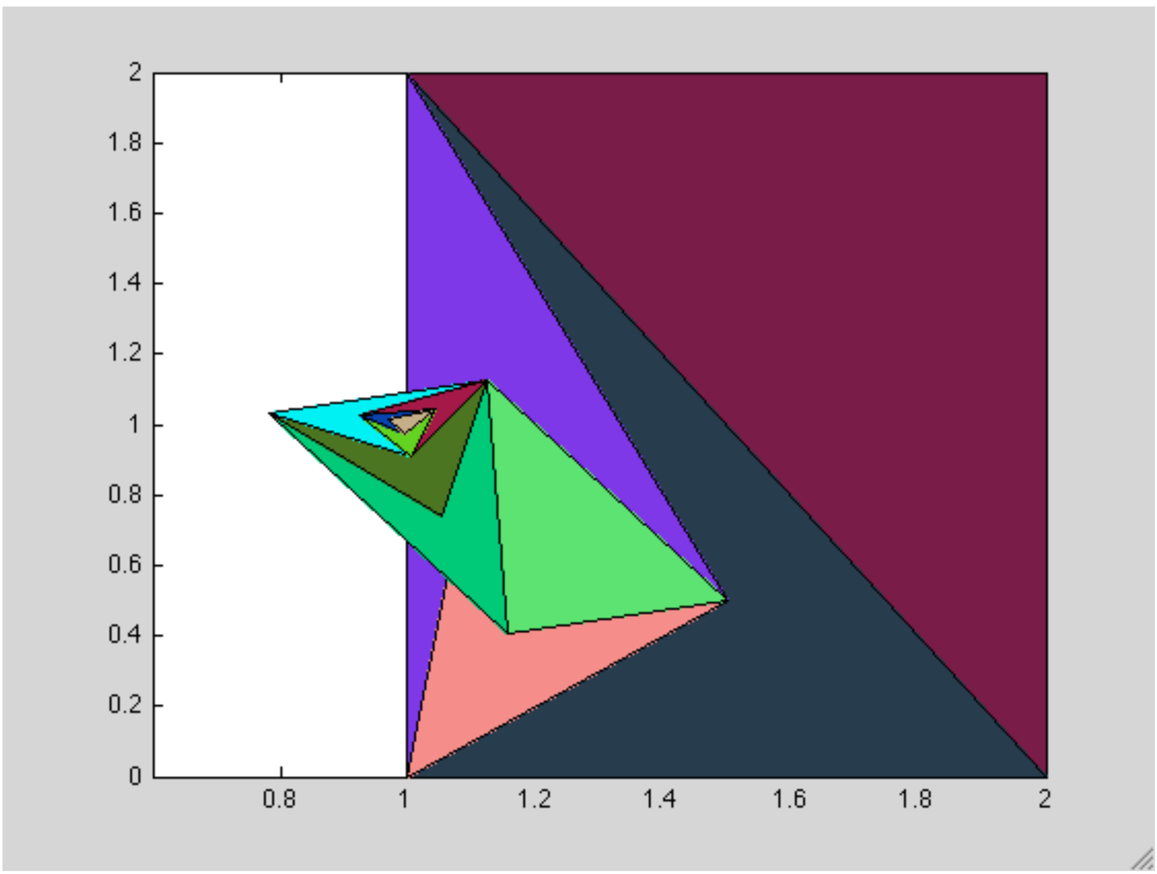
Here are several examples.

Good luck.

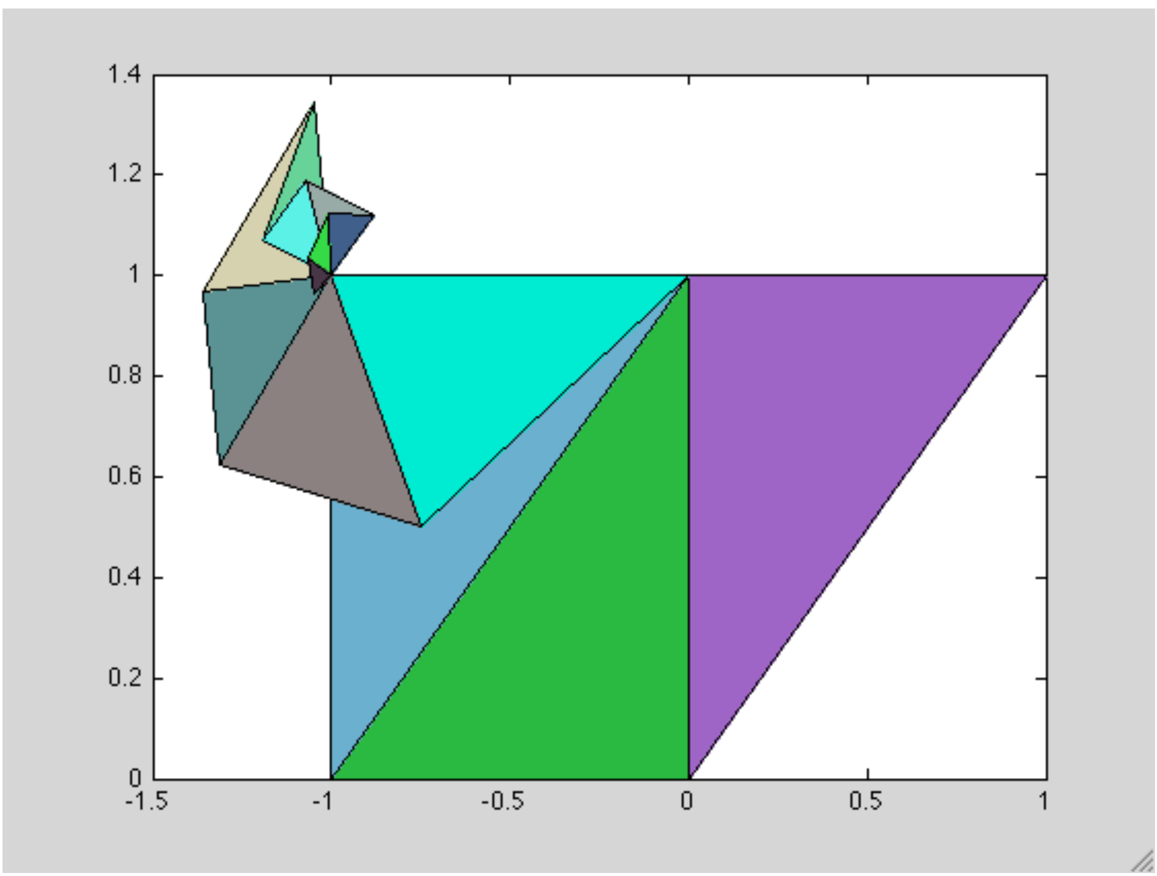
```
>> global fcount
>> format long
>> fcount = 0;
>> [X,Z] = NM2('f1',[0 0],[1.2 0],[0 0.8],0.1)

X =

   3.010290527343750   2.020715332031250


Z =

  -6.999678151756527

>> fcount

fcount =

    30
```



```
>> fcount = 0;
>> hold off
>> [X,Z] = NM2('f2',[1,2],[2,0],[2,2],0.1)

X =

   0.993713378906250   0.972717285156250


Z =

   1.002331048150609

>> fcount

fcount =

    24
```



```
>> fcount = 0;
>> hold off
>> [X,Z] = NM2('f3',[0,0],[0,1],[1,1],0.1)

X =

    -1     1


Z =

  -0.500000000000000

>> fcount

fcount =

    24
```



Here are the functions used in the three examples above.

```
function z = f1(x)

global fcount

fcount = fcount + 1;

z = x(1)^2 - 4*x(1) + x(2)^2 - x(2) - x(1)*x(2);

end
```

```
function z = f2(x)

global fcount

fcount = fcount + 1;

z = x(1)^3 + x(2)^3 - 3*x(1) - 3*x(2) + 5;

end
```

```
function z = f3(x)

global fcount

fcount = fcount + 1;

z = (x(1)-x(2))/(2+x(1)^2+x(2)^2);

end
```