1. b) Values my function returns together with interval used are presented below. One can see that the solution is not exactly same for every interval. I think this can be explained by the use of sqrt(epsilon) in determining whether limits of the interval are close enough to each other that one can say that the minimum has been found. For doubles, machine epsilon is about 1e-16, which means its square root is about 1e-8. This leaves plenty of room for wiggle in last shown decimal places.

[5.0, 10.0] 7.1553782834
[5.1, 10.1] 7.1553782788
[4.9, 9.9] 7.1553782543

2. b) Search range (I always used squares, program allows other shapes too) given to fork_2d are shown below together with the location of the minimum points. Results are fairly close to the expected value of (1, 1), though not quite as good as in exercise 1, probably due to banana function being generally difficult.

[0.000000, 1.500000]: (0.9999967479, 0.9999934967)
[0.400000, 1.400000]: (0.9999962410, 0.9999924801)
[-0.600000, 1.600000]: (0.9999963142, 0.9999926265)
[0.000000, 10.000000]: (1.0000069223, 1.0000138513)
[0.000000, 20.000000]: (1.0000074285, 1.0000148644)

3. My output for amolin_banana when the triangle used is defined by points (0,0), (2,0) and (1,2):
rtol=   0.00000
Number of function calls = 169
   1.00000048   1.00000095   2.27373675E-13
   1.00000048   1.00000095   2.27373675E-13
   1.00000048   1.00000095   2.27373675E-13

Using (0,0), (6,0) and (6, 12):
nfunk= 1000
   1.00000000   1.00000000   0.00000000
   1.00000000   1.00000000   0.00000000
   1.00000000   1.00000000   0.00000000

Using (-5, 0), (7,0) and (1, 12):
nfunk= 1001
   1.00000000   1.00000000   0.00000000
   1.00000000   1.00000000   0.00000000
   1.00000000   1.00000000   0.00000000

Simplex method produces better results than my implementation from exercise 2 and seems to be more tolerant to growing the search area.

4.   amolin_fit gives
rtol=   0.00008
Number of function calls =   89
   1.00009346   -5.09707606E-04   3.00055218

```
  0.999960780      -5.55145089E-05   3.00030684
   1.00007725      -2.75181548E-04   3.00045395
```
when run with initial simplex (-5, 0), (5, 0), (0, 5).

This represents line y=x. It goes through some of the points, but pretty much ignores some of them. Of course there is some deviation from values a=1 and b=0 but the differences are so small that they are probably due to errors. To get a fit that would look better to my eye, one would probably have to use something other than L1-norm.