Consider the following Python function:

```
def f (x, y):

    return math.sin (x + y) + (x - y) ** 2 - 1.5 * x + 2.5 * y + 1
```

This implements the so-called*McCormick function*(see https://en.wikipedia.org/wiki/File:McCormick_function.pdf) sometimes used to test optimization strategies.

Your goal is to implement two numerical optimization methods that find the minimum of this function. For your reference, thisminimum is found at:

```
>>> f(-0.54719755, -1.54719755)

-1.9132229549810367
```

Here are your tasks:

1. Implement a gradient descent algorithm to find the minimum of this function. Your algorithm should start with a learning rate of 1, and then implement the "bold driver" heuristic. That is, if the value of the objective function increases in an iteration, then you half the learning rate. If it decreases, multiply the learning rate by 1.1. The algorithm should be encapsulated within a procedure gd_optimize (a) where a is a NumPy array with two entries (the starting x and starting y). At the end of each iteration, your code should print out the value of the objective function. When the change in the objective function is less than 10e-20, the code should print out the current value for x and y and then exit (in the past, some people have had to change this threshold to 10e-10 in order to terminate, which is fine). Run your procedure using gd_optimize (np.array ([-0.2, -1.0])) and gd_optimize (np.array ([-0.5, -1.5])) and copy and paste your code and results here, in Canvas. For your reference, my results are below. Note that first test gets stuck at a local minimum, though due to small differences in your implementation, your first run may not get stuck like mine did.
2. Implement Newton's method to find the minimum of this function. This should beencapsulated within a procedurenm_optimize (a).Run your procedure usingnm_optimize (np.array ([-0.2, -1.0]))andnm_optimize (np.array ([-0.5, -1.5]))and copy and paste your code and results here, in OwlSpace. For your reference, my results are below.

Note that these codes are not long. My gradient descent code has a body that is 11 lines long (plus two additional one-line functions that compute the gradient). My Newton's method code is seven lines long (plus four additional one-line functions). Also note thatnp.linalg.inv ()takes the inverse of a matrix,np.transpose ()transposes a matrix, andnp.dot ()multiplies two matrices/vectors/combinations thereof.

My results:

```
>>> gd_optimize (np.array ([-0.2, -1.0]))

-1.87282722794

-1.48115354629

-1.80492099531

-1.7873866234

-1.78924994303

-1.79005851324

-1.79035784931

-1.79044716196

-1.79046668309

-1.79046923087

-1.79046931641

-1.79046931016

-1.7904693107

-1.79046931095

-1.79046931105

-1.79046931108

-1.79046931108
```

```
-1.79046931108

-1.79046931108

-1.79046931108

-1.79046931108

-1.79046931108

[-0.43548187 -1.23548187]


>>> gd_optimize (np.array ([-0.5, -1.5]))

-1.91092958058

-1.91146816749

-1.9110297007

-1.91322152817

-1.91322292147

-1.91322295258

-1.91322295461

-1.91322295487

-1.91322295493

-1.91322295494

-1.91322295494

-1.91322295498

-1.91322295498

-1.91322295498
```

```
[-0.54719755 -1.54719755]


>>> nm_optimize (np.array ([-0.2, -1.0]))

-1.91281352075

-1.91322291866

-1.91322295498

-1.91322295498

-1.91322295498

-1.91322295498

[-0.54719755 -1.54719755]


>>> nm_optimize (np.array ([-0.5, -1.5]))

-1.91322090085

-1.91322295498

-1.91322295498

-1.91322295498

-1.91322295498

[-0.54719755 -1.54719755]
```