```
import math
import numpy as np


def f(x, y):
  return math.sin(x + y) + (x - y) ** 2 - 1.5 * x + 2.5 * y + 1


def grad_fx(x, y):
  return math.cos(x + y) + 2 * (x - y) - 1.5


def grad_fy(x, y):
  return math.cos(x + y) - 2 * (x - y) + 2.5
```

Method 1: Gradient descent

```
def gd_optimize(arr):
  thresh = 10e-20
  lr = 1
  out = f(arr[0], arr[1])
  x = arr[0]
  y = arr[1]
  print(out)

  #The iteration
  while True:
    new_x = x - lr * grad_fx(x, y)
    new_y = y - lr * grad_fy(x, y)
    new_out = f(new_x, new_y)
    print(new_out)

    #heuristic:
    if new_out>out:
      lr = lr / 2
    else:
      lr = lr * 1.1

    if abs(new_out - out) < thresh:
      print(new_x, new_y)
      break

    x = new_x
    y = new_y
    out = new_out
```

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

```
-1.4920390859672263
-1.3175387318156826
-1.503265873161276
-1.3933929562543743
-1.9076321773193428
-1.912900015321147
-1.9131807504289906
-1.9132152450977031
-1.91322073144749
-1.913221746385557
-1.9132218772859027
-1.9132215459436885
-1.9132229477178324
-1.913222954748297
-1.913222954960514
-1.9132229549773028
-1.9132229549798279
-1.913222954980399
-1.9132229549805215
-1.913222954980426
-1.91322295498103
-1.9132229549810367
-1.9132229549810367
-0.5471975518820887 -1.547197550524929
```

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

```
-1.909297426825682
-1.9109295805761808
-1.9114681674883558
-1.9110297007042236
-1.9132215281704674
-1.9132229214706045
-1.913222952576786
```

```
-1.9132229546063524
-1.9132229548741102
-1.9132229549304762
-1.9132229549439543
-1.9132229549407707
-1.9132229549810185
-1.9132229549810362
-1.9132229549810358
-1.9132229549810367
-1.9132229549810362
-1.9132229549810367
-1.9132229549810367
-0.5471975510477202 -1.5471975510478024
```

Method 2: Newtons

```
def J_inv(x, y):
  out_mat = np.zeros((2,2))
  out_mat[0, 0] = -math.sin(x + y) + 2 #df/dx2
  out_mat[0, 1] = -math.sin(x + y) - 2
  out_mat[1, 0] = -math.sin(x + y) - 2
  out_mat[1, 1] = -math.sin(x + y) + 2

  out_inv = np.linalg.inv(out_mat)
  return out_inv

def grad_mat(x, y):
  out = np.zeros(2)
  out[0] = grad_fx(x, y)
  out[1] = grad_fy(x, y)
  return out


def nm_optimize(arr):
  thresh = 10e-10
  out = f(arr[0], arr[1])
  x = arr[0]
  y = arr[1]
  print(out)

  #The iteration
  while True:
    J_in = J_inv(x, y)
    F_grad = grad_mat(x, y)
    new_arr = arr - np.dot(J_in, np.transpose(F_grad))
    new_x = new_arr[0]
    new_y = new_arr[1]
    new_out = f(new_x, new_y)
    print(new_out)

    if abs(new_out - out) < thresh:
      print(new_x, new_y)
      break

    x = new_x
    y = new_y
    arr = new_arr
    out = new_out
```

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

```
-1.4920390859672263
-1.9128135207487111
-1.9132229186591214
-1.9132229549810362
-1.9132229549810362
-0.5471975511965976 -1.5471975511965976
```

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

```
-1.909297426825682
-1.9132209008539096
-1.913222954980231
-1.9132229549810362
-0.5471975511963294 -1.5471975511963294
```