3. Run SuperLearn.py. The last line of the file calls the test1 function, which calls your learning algorithm on the four example points and prints out results. As a result of learning, your approximate function value should move 10% of the way from its original value towards the target value. The before value of the fourth point should be nonzero (why?). Make sure this is working correctly before going on to the next step.

Because we call example (4,2,-1)first, and it is very similar as (4,2.1,-1). Therefore, the function already stores some value in the weight. Because the index is similar, it can get the value from weight. Unlike others are call the first time of these index.

4. Change the last line of SuperLearn.py to call test2 instead of test1 and execute it again. The test2 function constructs 20 examples using the target function:
target = sin(x - 3) cos(y) + N(0, 0.1);
where N(0; 0:1) is a normally distributed random number with mean 0 and standard deviation 0.1. The examples are sent to learn, and then the resultant function f (after 20 examples) is printed out to a _le (f20) in a form suitable for plotting by Excel or similar. A Monte Carlo estimate of the Mean-Squared- Error in the function is also printed to standard output. The program the continues for 10,000 more examples, printing out the MSE after each 1000, and then _nally prints out the function to a _le (f10000) for plotting again. You should see the MSE coming down smoothly from about 0.25 to almost 0.01 and staying there (why does it not decrease further towards zero?).

Because the function's limit, there will always be some error there. The function use valuable alpha to learn. However, the alpha is not small enough to make the value farther more accurate. When it really close to 0.01, it will start to increase, because the error is there.