**Report for Mean-Shift Algorithm**

**2.1 Implement the distance Function**

We use 2-norm as the distance metric between points. Since radius is , we need to consider all the points. The code is as followed:



**2.2 Implement the gaussian Function**

We compute weight as a Gaussian function of distance. The code is as followed:



Note that the normalization term of gaussian distribution would have no effect on the final output, since in the update\_point or update\_point\_batch function, we will normalize the weights such that the sum of the weights is 1. So, it does not matter whether to normalize in gaussian function or not. To make the algorithm a bit faster, I don’t use normalizer in the gaussian function.

**2.3 Implement the update\_point Function**

To make algorithm faster, I don’t use for loops. Instead, I transpose the weight and then multiply it by X matrix. Then the sum would be the new update point. The code is following:



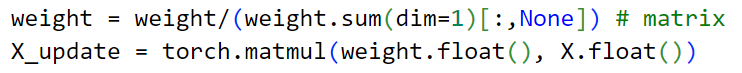
**2.4 Accelerating the Naïve Implementation**

To accelerate the whole pipeline, we vectorize the inputs to avoid looping over each single point. The details are as followed:

1. In distance\_batch function, first reshape X for broadcasting, then we can compute the distances between any 2 points. Note that in this case, the first argument x is useless. The code is as followed:



1. In update\_point\_batch function, the weight is a matrix instead of a vector. Thus, I use matrix multiplication to compute the new points X\_update:



1. In meanshift\_step\_batch function, call distance\_batch function and update\_point\_batch function instead of distance function and update\_point function.

Using both slow implementation (update for one point at a time) and fast implementation (update all points together using vectorization), we get following results:

1. Slow implementation:



1. Fast implementation:



As can be seen vectorization contributes to tremendous speed-up.