Report for Mean-Shift Algorithm

2.1 Implement the distance Function

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

```
dist = torch.norm((X.float() - x.float()), dim=1)
```

2.2 Implement the gaussian Function

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

```
weight = torch.exp(-(dist/bandwidth)**2/2)
```

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:

```
weight = weight/weight.sum()
x = torch.sum(weight[:, None] *X, dim = 0)
```

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:

a) 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:

```
d = X.reshape(-1, 1, 3) - X.reshape(1, -1, 3)
dist = torch.norm(d.float(), dim=2)
```

b) 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:

```
weight = weight/(weight.sum(dim=1)[:,None]) # matrix
X_update = torch.matmul(weight.float(), X.float())
```

c) 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:

a) Slow implementation:

```
Elapsed time for mean-shift: 12.28938364982605
```

b) Fast implementation:

```
Elapsed time for mean-shift: 2.279999256134033
```

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