Problem 1

The answers to the stated questions:

1. **Q:** Exercise 2. a). Call the function *test_sine_toydata* with $\lambda = 1$ for three different Kernel widths, $\sigma = \{0.1, 1, 10\}$. How does the Kernel width affect the solution? Explain the observed behavior.

   A: The parameter σ tells us how far can two similar points be (when using a Gaussian Kernel). If the parameter is big, then the variance of the Gaussian distribution centered in each point will be also big. Therefore, as we decrease σ, we trust more the data and our prediction will be more sensible to outliers. When we increase σ the prediction will tend to look more like normal linear regression, because we will use less the advantage of similarities.

2. **Q:** Exercise 2. b). Call the function *test_sine_toydata* with $\sigma = 1$ for three different regularization parameters, $\lambda = \{10^{-10}, 1, 500\}$. How does the regularization parameter, affect the solution? Explain the observed behavior.

   A: the regularization term has the effect of penalizing large values of w_i unless supported by the training data. Larger values of lambda tend to drive more values of w_i towards 0, thus reducing the complexity of the solution. We can see this in our plots: for $\lambda = 10^{-10}$, the penalizing effect is very small, resulting in an over-fitted solution which closely models the Gaussian noise added to the sinusoidal data, and which as a result fails to generalize well. At the other extreme, choosing $\lambda = 500$ will lead to many w_i with very small values, leading to a nearly linear, under-fitted solution which also fails to generalize well. For $\lambda = 1$, we obtain a curve which most closely resembles the underlying sinusoidal data of the training set.

3. **Q:** Exercise 3. Explain briefly how λ and σ are chosen within the function *crossvalidate_krr*.

   A: The parameters σ and λ are chosen from the lists *kwidths* and *llambdas* respectively, which are given as optional parameters to the function *crossvalidate_krr*. The training is made on every possible combination of parameters from these two lists.

4. **Q:** Exercise 4. What does a boxplot show? Do we gain something from Kernel Ridge Regression as compared to simple linear regression?

   A: A boxplot shows what is the distribution of the results based on the position of the quartiles. Compared to simple linear regression, we see a considerable increase in the median coefficient of determination across the five folds, indicating that the predictions are more accurate.

5. **Q:** Exercise 5. For the last question, we have applied the function *test_handpositions* only to the first 1000 data points out of the 10255 available data points. Why did we do so in this exercise?

   A: It would've been very time and space consuming to cross validate over such an amount of data points in the *KRR* case. Because in the *KRR* algorithm we always compute the similarities between all the points in the training phase and then on the prediction also we are using all the training set. If we would've choose all the 10255 the Kernel matrix (for the default number of folds) would had over $6 * 10^7$ elements to store (big space) and use (bigger time).

Problem 2

The requested plots. The first 4 figures represent the output of the function *test_sine_toydata* for various parameters λ, σ. The fifth figure represents the comparison of the r^2 factor between *KRR* and *LR*:

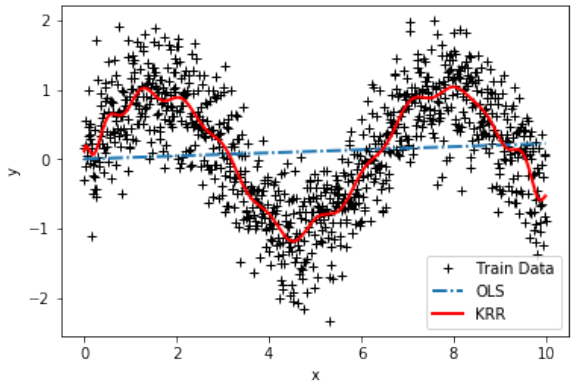

Figure 1: $\lambda = 10^{-10}$ and $\sigma = 1$

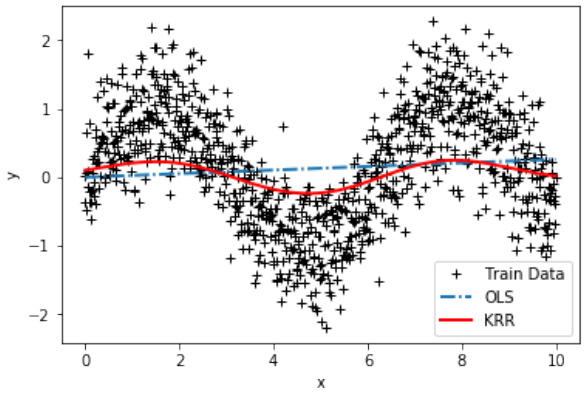

Figure 2: $\lambda = 500$ and $\sigma = 1$

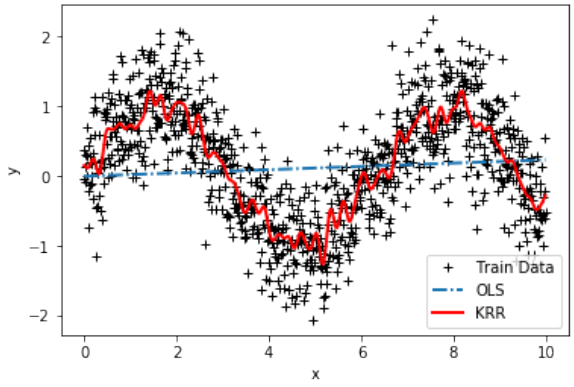

Figure 3: $\lambda = 1$ and $\sigma = 0.1$

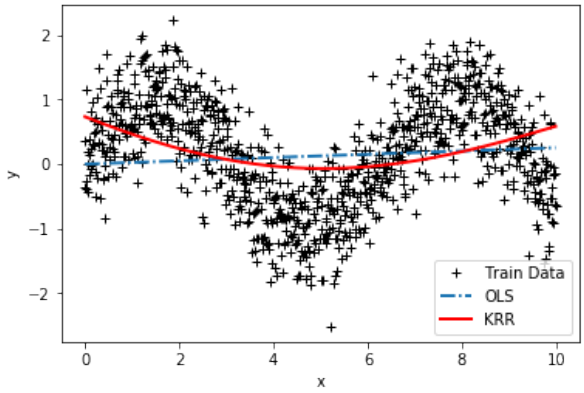

Figure 4: $\lambda = 1$ and $\sigma = 10$

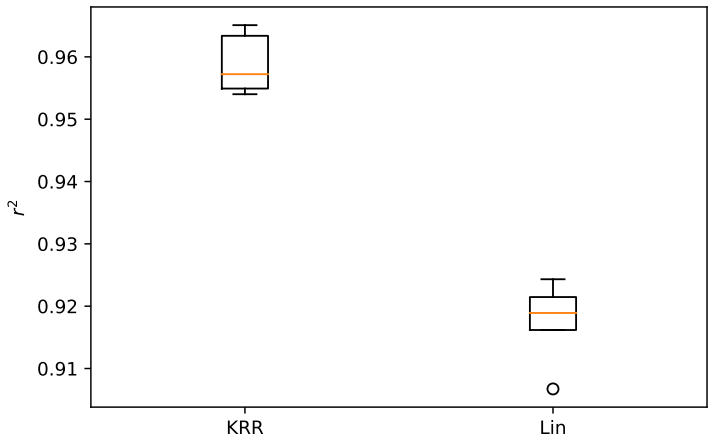

Figure 5: Comparison of the r^2 factor distribution between *Kernel Ridge Regression* (left) and *Linear Regression* (right)

Problem 2 continued on next page...

Problem 3

The code for the functions *train_krr* and *apply_krr* is the flowing:

```
def train_krr(X_train, Y_train,kwidth,llambda):
    ''' Trains kernel ridge regression (krr)
    Input:       X_train  -  DxN array of N data points with D features
                 Y        -  D2xN array of length N with D2 multiple labels
                 kwdith   -  kernel width
                 llambda    -  regularization parameter
    Output:      alphas   -  NxD2 array, weighting of training data used for apply_krr
    '''
    # your code here
    K = GaussianKernel(X_train, X_train, kwidth)
    KLI = (K + np.diag(llambda * np.ones(len(K))))
    alphas = np.linalg.inv(KLI).dot(Y_train.T)
    return alphas
```

```
def apply_krr(alphas, X_train, X_test, kwidth):
    ''' Applys kernel ridge regression (krr)
    Input:      alphas      -  NtrxD2 array trained in train_krr
                X_train     -  DxNtr array of Ntr train data points with D features
                X_test      -  DxNte array of Nte test data points with D features
                kwidht      -  Kernel width
    Output:     Y_test      -  D2xNte array
    '''
    # your code here
    k = GaussianKernel(X_test, X_train, kwidth)
    return k.dot(alphas).T
```