KU LEUVEN

FACULTY OF SCIENCE

Exercise Session 2

Support Vector Machines

Daniel Gerardo GIL SANCHEZ
daniel.gilsanchez@student.kuleuven.be
MSc Statistics

Supervisor: Prof. Johan Suykens

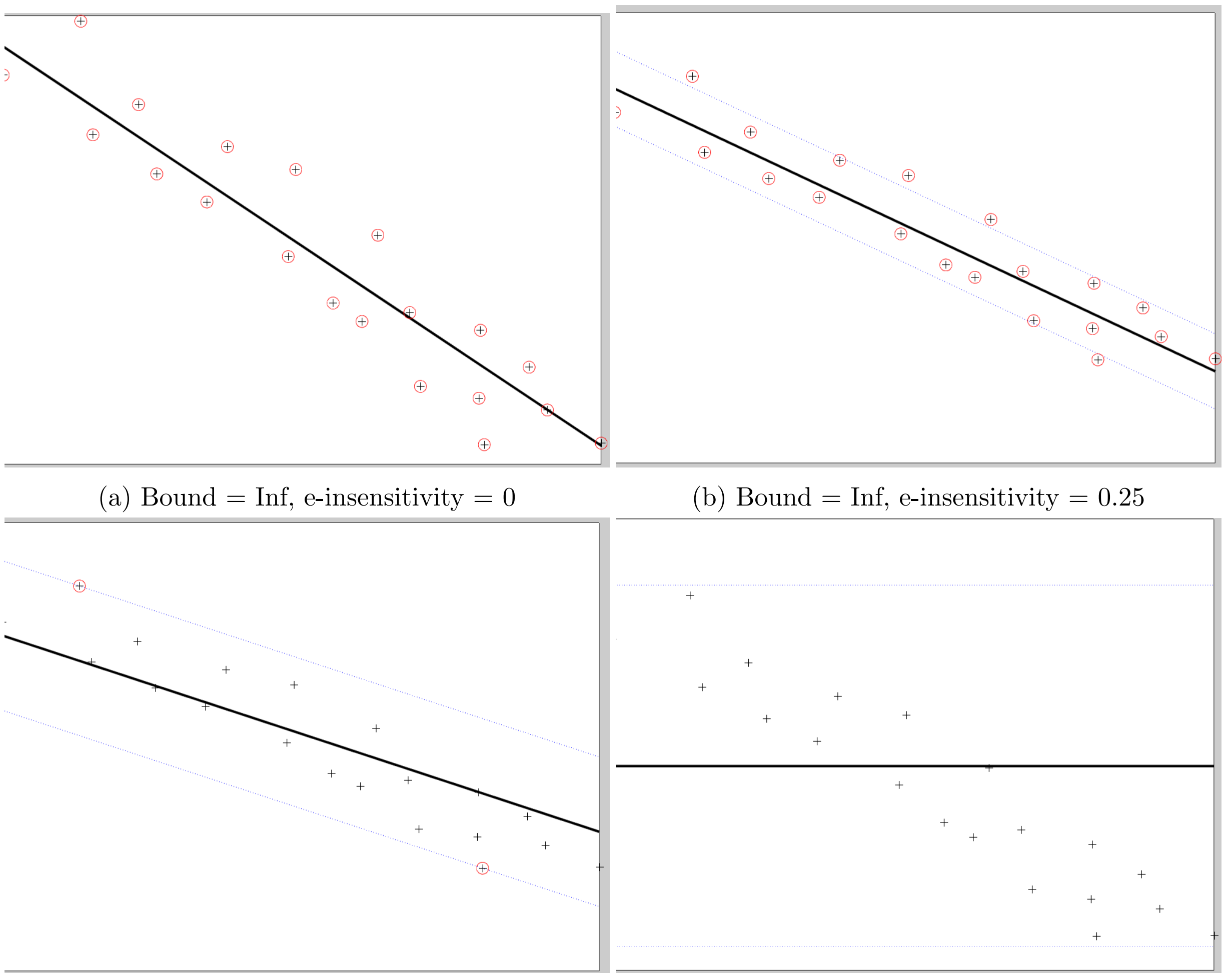
Academic year 2018-2019

1 Exercises

1.1 Support vector machine for function estimation

- **Construct a dataset where a linear kernel is better than any other kernel (around 20 data points). What is the influence of `e` (try small values such as $0.10,\ 0.25,\ 0.50, \ldots$) and of `Bound` (try larger increments such as $0.01,\ 0.10,\ 1,\ 10,\ 100$). Where does the sparsity property come in?**

  Figure 1 shows how the number of support vectors and the overall fit change when `e` is manipulated. In the first scenario `e` is equal to zero, resulting in a good fit and 20 support vectors. When `e` increases, the number of support vectors is smaller and the fit gets worse. For instance, when `e` is equal to 1, there are no support vectors in the solution and the fit is just a horizontal line.



(a) Bound = Inf, e-insensitivity = 0

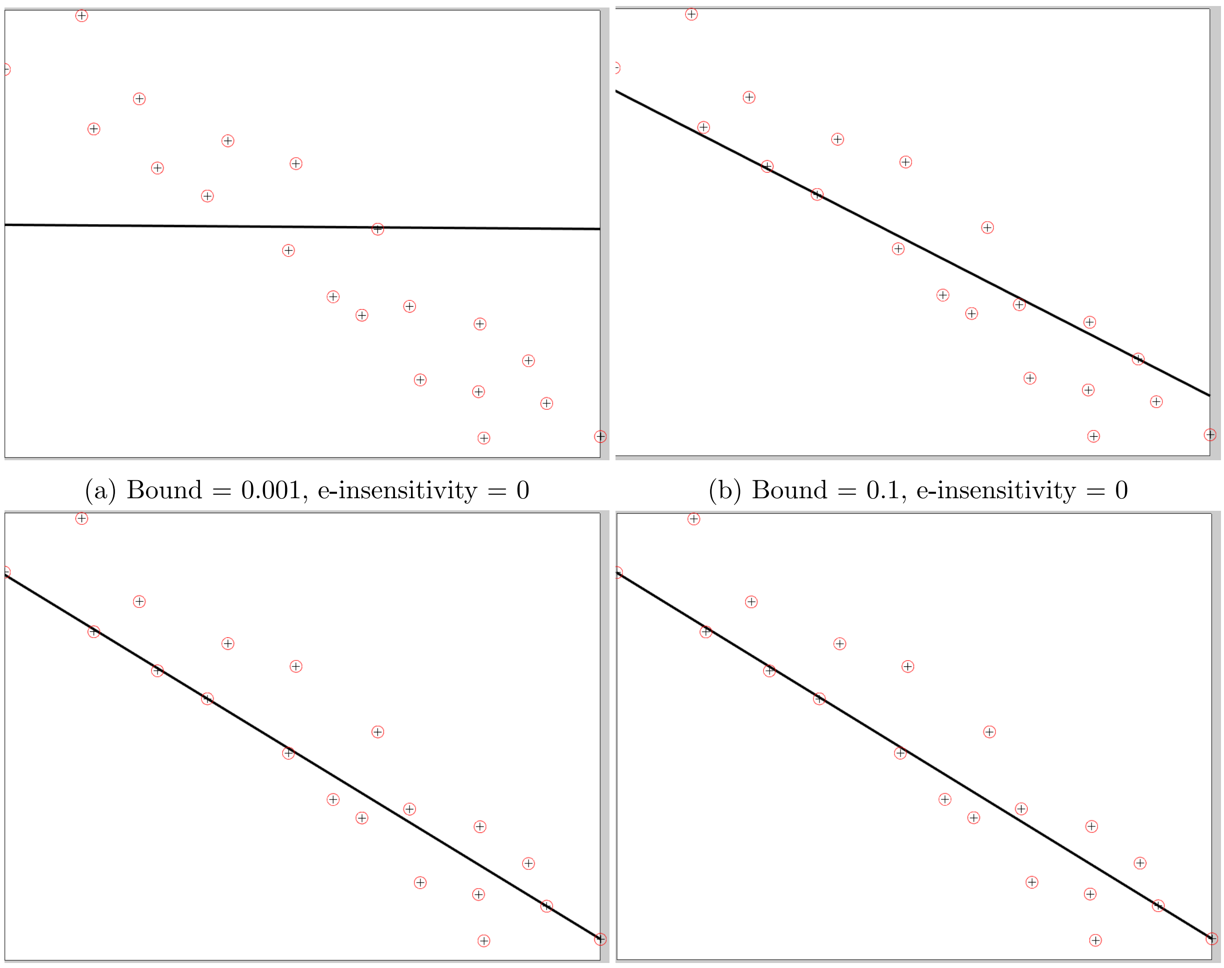
(b) Bound = Inf, e-insensitivity = 0.25

(c) Bound = Inf, e-insensitivity = 0.5

(d) Bound = Inf, e-insensitivity = 1

Figure 1: Influence of `e`

In a similar fashion, Figure 1 shows how the solution changes when `Bound` is manipulated. In the first plot `Bound` is close to zero, resulting in a poor fit by considering just a horizontal line. As `Bound` increases in little amounts, the fit improves up to a point that it does not change any more, see plots 2c and 2d. Other attempts were also tried by considering `Bound` equal to 10 and 100, but the fit did not change with respect to the last solution shown here.



(a) Bound $= 0.001$, e-insensitivity $= 0$
(b) Bound $= 0.1$, e-insensitivity $= 0$
(c) Bound $= 0.5$, e-insensitivity $= 0$
(d) Bound $= 1$, e-insensitivity $= 0$

Figure 2: Influence of `e`

Overall, it is important to notice that `Bound` does not control the sparsity of the solution. It does not matter what number is considered, the number of support vectors will be the same. On the other hand, `e` does control the sparsity of the solution. The larger `e`, the sparser the solution. This is related with the `e`-insensitive loss function, which is a modification of the L1 loss function, defined as:

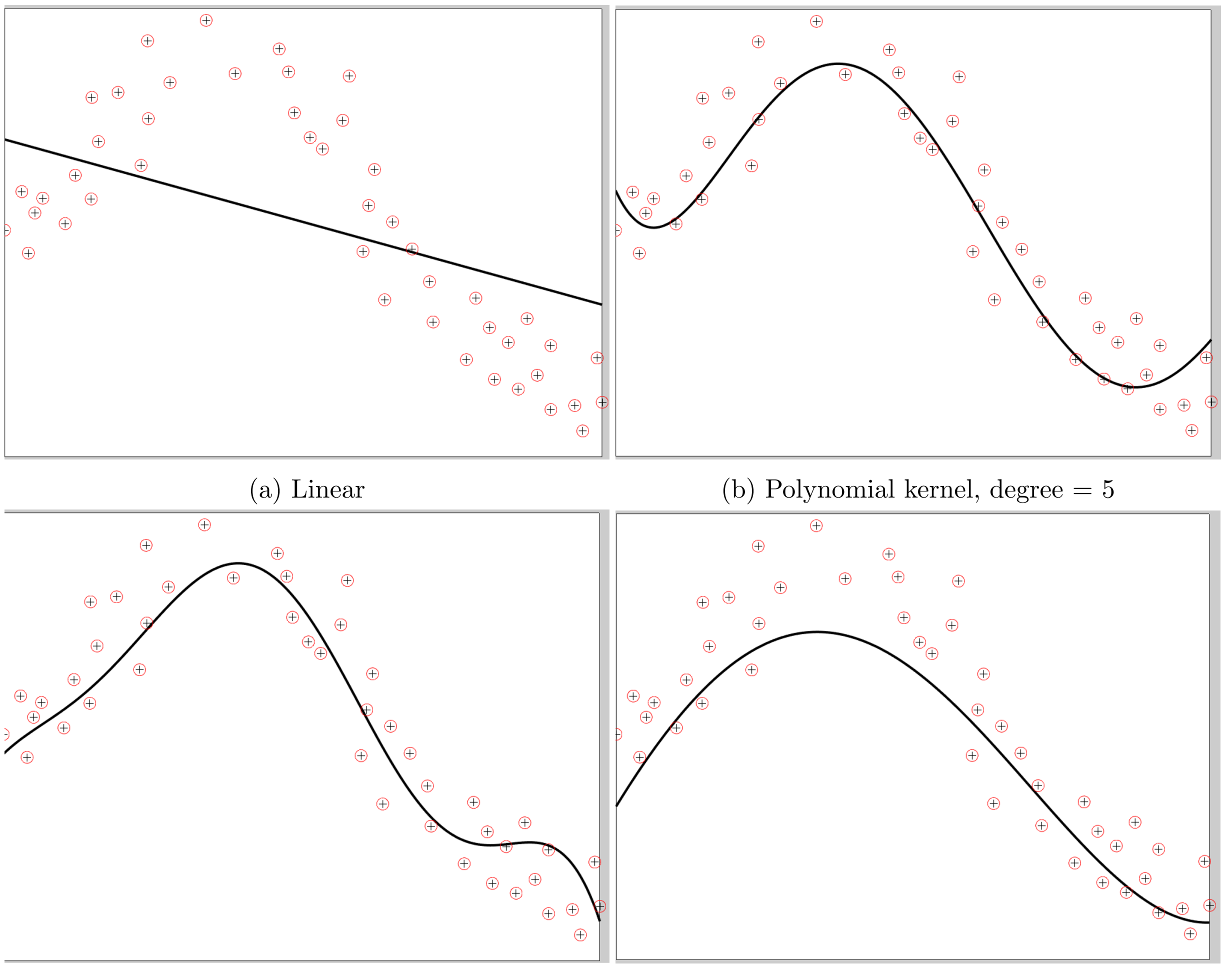
$$|y - f(x)|_\epsilon = \begin{cases} 0 & \text{, if } |y - f(x)| \leq \epsilon \\ |y - f(X)| - \epsilon & \text{, otherwise} \end{cases} \quad (1)$$

- ***Construct a more challenging dataset (around 20 data points). Which kernel is best suited for your dataset? Motivate why.***

  Figure 3 shows different solutions using different kernels in a new dataset with nonlinear structure. As it was expected, the linear kernel did not fit the data well. The polynomial kernel solution was close to a good fit, ignoring the tails of the function, where it started to increase. On the other hand, the RBF kernel with $\sigma^2 = 1$ led to an overall good solution when compared to the solution using $\sigma^2 = 3$. In general, polynomial and RBF kernels produce better solutions that a linear kernel when the underlying process that generates the data is nonlinear. This happens because of how they are defined:

  – Polynomial: $K(x, x_k) = (x_k^{'}x + \tau)^d$ with degree d and $\tau \geq 0$

– RBF: $K(x, x_k) = exp(-||x - x_k||_2^2/\sigma^2)$



(a) Linear

(b) Polynomial kernel, degree = 5

(c) RBF kernel, $\sigma^2 = 1$

(d) RBF kernel, $\sigma^2 = 3$

Figure 3: Influence of e

- ***In what respect is SVM regression different from a classical least squares fit?***

  In linear function approximation, in classical least squares, the purpose is to minimize the distance between each data point and the overall solution. This is conducted by minimizing the following expression:

$$R = \frac{1}{N}\sum_{n=1}^{N}(y_k - w'x_k - b)^2$$

  In SVM, the empirical risk is minimized, which is defined as:

$$R_emp = \frac{1}{N}\sum_{n=1}^{N}|y_k - w'x_k - b|_\epsilon$$

  or a modification of this expression can also be minimized, as the e–insensitive loss function (defined in equation 1).

  The idea behind function estimation in SVM, is that all data points are contained in a $\epsilon-$tube: $|y_i - \hat{y}_i| \leq \epsilon$. If this is not the case, some data points can be outside the $\epsilon-$tube by introducing the so-called slack variables in the minimization process. As a consequence, the SVM solution tends to be more robust than a classical least squares solution.

In nonlinear function approximation, SVM solutions tend to be more precise because its flexibility in the choice of a Kernel function, such as polynomial, RBF, MLP, among others. On the contrary, the only way that classical least squares can deal with nonlinear functions is by introducing polynomial terms in the solution.

1.2 A simple example: the sinc function

1.2.1 Regression of the sinc function

- ***Try out a range of different gam and $sig2$ parameter values (e.g., gam = 10, 10^3, 10^6 and $sig2$ = 0.01, 1, 100) and visualize the resulting function estimation on the test set data points. Discuss the resulting function estimation. Report the mean squared error for every combination (gam, $sig2$).***

  Figure 4 shows how the fit changes when `gam` and `sig2` are manipulated. In the first and the second column of plots the overall fit is really close to the underlying process that generated the data. The only difference between these plots is the maximum and minimum of approximated function: in some cases is close enough (plot 4a), whereas in others the fit is not that good (plot 4b). On the other hand, the third column of plots shows the impact of `sig2` in the final solution: in all of theses cases, `sig2` is 100 and the approximated function is not good at all. The best possible combination of parameters is `gam` = 10^6 and `sig2` = 1, where the corresponding mean squared error (MSE) is 0.0099 (plot 4h).

- ***Do you think there is one optimal pair of hyperparameters? Argument why (not).***

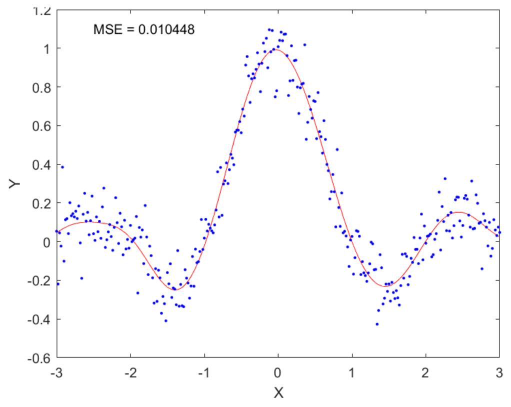
  As it has been shown in Figure 4, there is indeed an optimal pair of hyperparameters. The purpose of `gam` is to determine the trade-off between the fitting error minimization and the smoothness of the estimated function, this parameter is the regularization constant. On the other hand, `sig2` is the parameter present in the RBF kernel, which is defined as:

$$K(x, x_k) = exp\left(\frac{-||x - x_k||_2^2}{\sigma^2}\right)$$

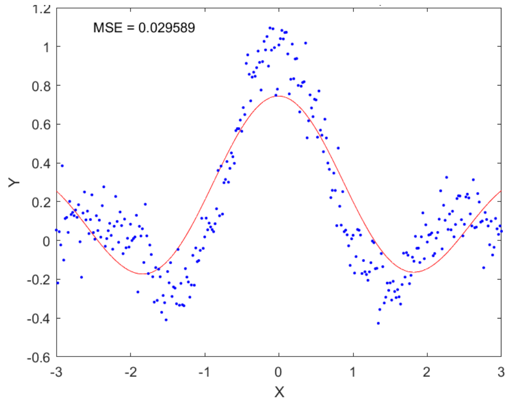
  Note that `sig2` is in the denominator, so when sigma is very large, the fraction will be small and the exponential will be close to one. For this reason, the importance of the numerator will always be diminished by the denominator, impacting the solution of the model. This can be easily seen in the last column of plots in Figure 4, where the overall fit is the worse of all possible scenarios.

- ***Tune the gam and $sig2$ parameters using the tunelssvm procedure. Use multiple runs: what can you say about the hyperparameters and the results? Use both the simplex and gridsearch algorithms and report differences.***

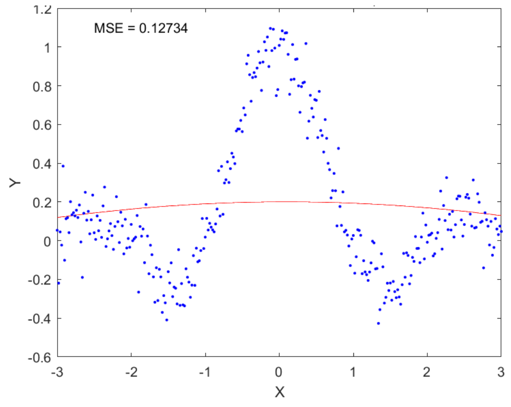
  Each algorithm (simplex or gridsearch) was run 10 times and a comparison of hyperparameters, MSE and processing times was conducted. Table 1 shows these results.



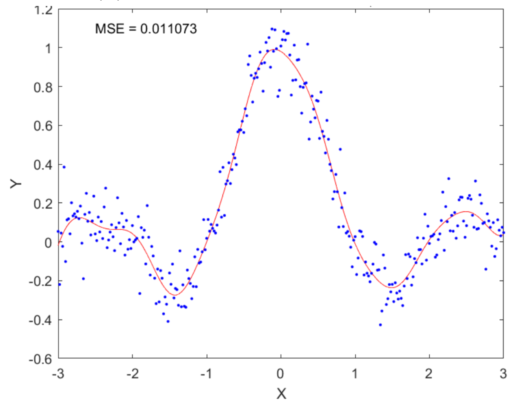
(a) `gam` $= 10$ and `sig2` $= 0.1$



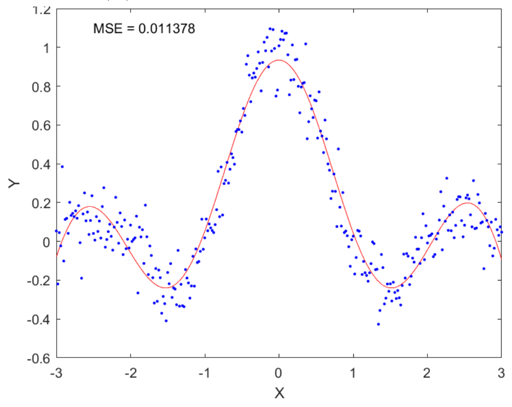
(b) `gam` $= 10$ and `sig2` $= 1$



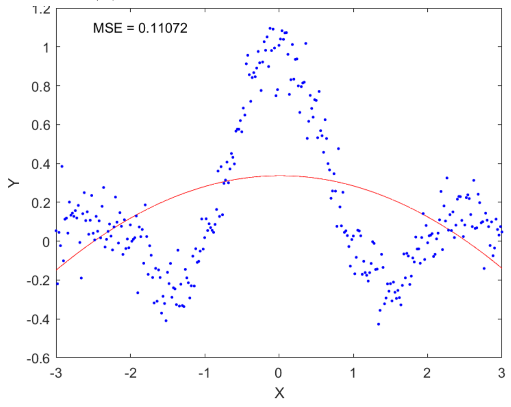
(c) `gam` $= 10$ and `sig2` $= 100$



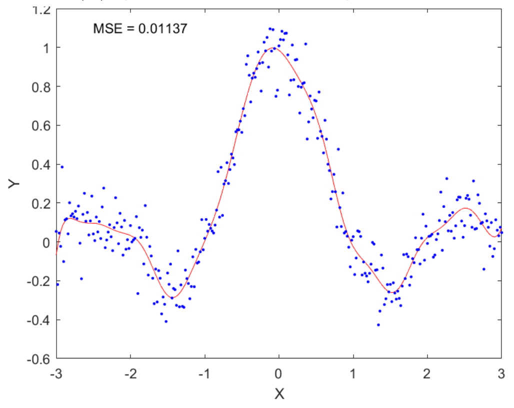
(d) `gam` $= 1000$ and `sig2` $= 0.1$



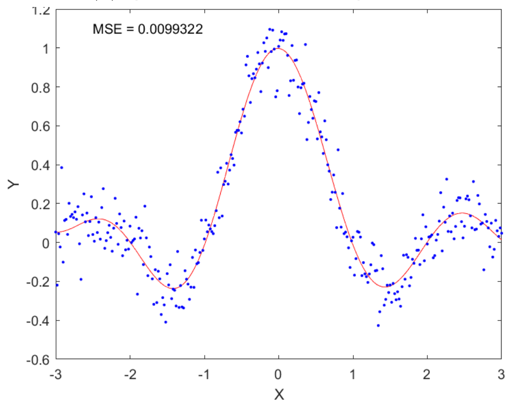
(e) `gam` $= 1000$ and `sig2` $= 1$



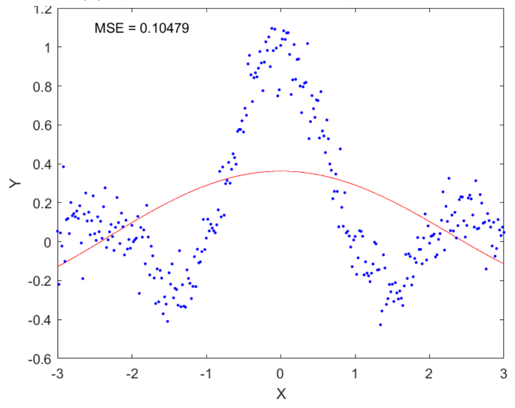
(f) `gam` $= 1000$ and `sig2` $= 100$



(g) `gam` $= 10^6$ and `sig2` $= 0.1$



(h) `gam` $= 10^6$ and `sig2` $= 1$



(i) `gam` $= 10^6$ and `sig2` $= 100$

Figure 4: Different values for `gam` and `sig2`

i	Simplex				Gridsearch			
	γ	σ^2	**SE**	**Time**	γ	σ^2	**MSE**	**Time**
1	10.62	0.236	0.011	1.865	9.685	0.183	0.010	2.797
2	3.87	0.038	0.010	1.098	4.634	0.039	0.011	2.068
3	5.337	0.049	0.010	1.287	23.271	0.064	0.011	2.439
4	26.279	0.057	0.011	1.324	7.298	0.213	0.011	1.643
5	5.885	0.21	0.010	1.337	2.461	0.035	0.010	1.856
6	532035.683	0.145	0.010	1.051	875.662	0.091	0.011	1.927
7	21.528	0.285	0.011	1.332	3.145	0.154	0.010	1.658
8	13.718	0.247	0.010	1.209	3.449	0.17	0.010	1.738
9	6.195	0.187	0.010	1.121	3.418	0.04	0.010	1.704
10	4.393	0.037	0.010	1.109	6.065	0.209	0.011	1.735

Table 1: Comparison of algorithm results over different runs (i)

The first thing that can be noticed is the difference between pairs of parameters from one run to the other under the same algorithm. For instance, in the `simplex` algorithm, the pair of parameters of the first run is (10.62, 0.236) and the sixth run is (532035.68, 0.145). This happens to both algorithms because the routine in Matlab does the following:

1. Use Coupled Simulated Annealing (CSA) to determine suitable starting points for every method.
2. These starting points are then given to one of the three optimization algorithms (`simplex`, `gridsearch` or `linesearch`).

According to the `LS-SVMlab Toolbox` user's guide, CSA is more effective than multistart gradient descent optimization and it uses the acceptance temperature to control the variance of the acceptance probabilities. This means that this algorithm is an improved optimization routine because it reduces the sensitivity of the algorithm to the initialization parameters. This can actually be confirmed by the MSE in each run. No matter what pair of parameters is chosen in each run, the MSE is more or less constant.

Now in terms of computational speed there is a notable difference between algorithms. The average speed of `simplex` algorithm is 1.2733 seconds, whereas for the `gridsearch` algorithm is 1.9565 seconds. This difference does not seem to be significant in this example, but if a more complex problem is analyzed, differences in processing time can be important.

In the end, the decision of which algorithm to use depends on the particular problem and the time at hand to do the analysis. `Gridseach` will almost always be slower than `simplex`, because it makes the search of the pair of parameters in an exhaustive manner.

1.2.2 Application of the Bayesian framework

- ***Discuss in a schematic way how parameter tuning works using the Bayesian framework. Illustrate this scheme by interpreting the function calls denoted above.***

  In Bayesian approach all parameters and hyperparameters involved in a model have a probabilistic distribution. As an example, the RBF kernel has five unknown parameters: w and b are the parameters of the model in the primal representation. The regularization constant, γ, is now defined in terms of μ and ζ as $\gamma = \zeta/\mu$. And σ^2 is the parameter associated to the kernel function. These parameters now belong to different levels of inference: w and b are inferred at Level 1. μ and ζ are inferred at Level 2 and σ^2 is inferred at Level 3. These levels are defined using the Bayes rule as follows:

– Level 1: (inference of parameters w and b)

$$p(w, b|\mathcal{D}, \mu, \zeta, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|w, b, \mu, \zeta, \mathcal{H}_\sigma)}{p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\sigma)} p(w, b|\mu, \zeta, \mathcal{H}_\sigma)$$

The idea here is to maximize the posterior distribution of w and b, which is defined in terms of the likelihood (numerator), the regularization constant (denominator), also called evidence, and the prior distribution of w and b. This is conducted with the function `bay_optimize` in Matlab and setting the level argument to 1.

```
[~, alpha ,b] = bay_optimize ({ Xtrain , Ytrain , 'f', gam , sig2 }, 1)
```

It is important to note that in Matlab, the dual representation is used in the optimization of the parameters. The Lagrange multipliers, α, are used instead of w.

– Level 2: (inference of hyperparameters μ and ζ)

$$p(\mu, \zeta|\mathcal{D}, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\sigma)}{p(\mathcal{D}|\mathcal{H}_\sigma)} p(\mu, \zeta|\mathcal{H}_\sigma)$$

Here the posterior of the hyperparameters is again defined in terms of likelihood, evidence and prior distribution. What is important to note is the fact that the evidence in Level 1 is the same as the likelihood at this level. To maximize this expression in Matlab, `bay_optimize` is used, setting the level argument to 2.

```
[~, gam] = bay_optimize ({ Xtrain , Ytrain , 'f', gam , sig2 }, 2)
```

– Level 3: (inference of kernel parameters, σ^2 in this case)

$$p(\mathcal{H}_\sigma|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H}_\sigma)}{p(\mathcal{D})} p(\mathcal{H}_\sigma)$$

Similar to levels before, the likelihood in this level is the same as the evidence of Level 2. To maximize this expression in Matlab, `bay_optimize` is used, setting the level argument to 3.

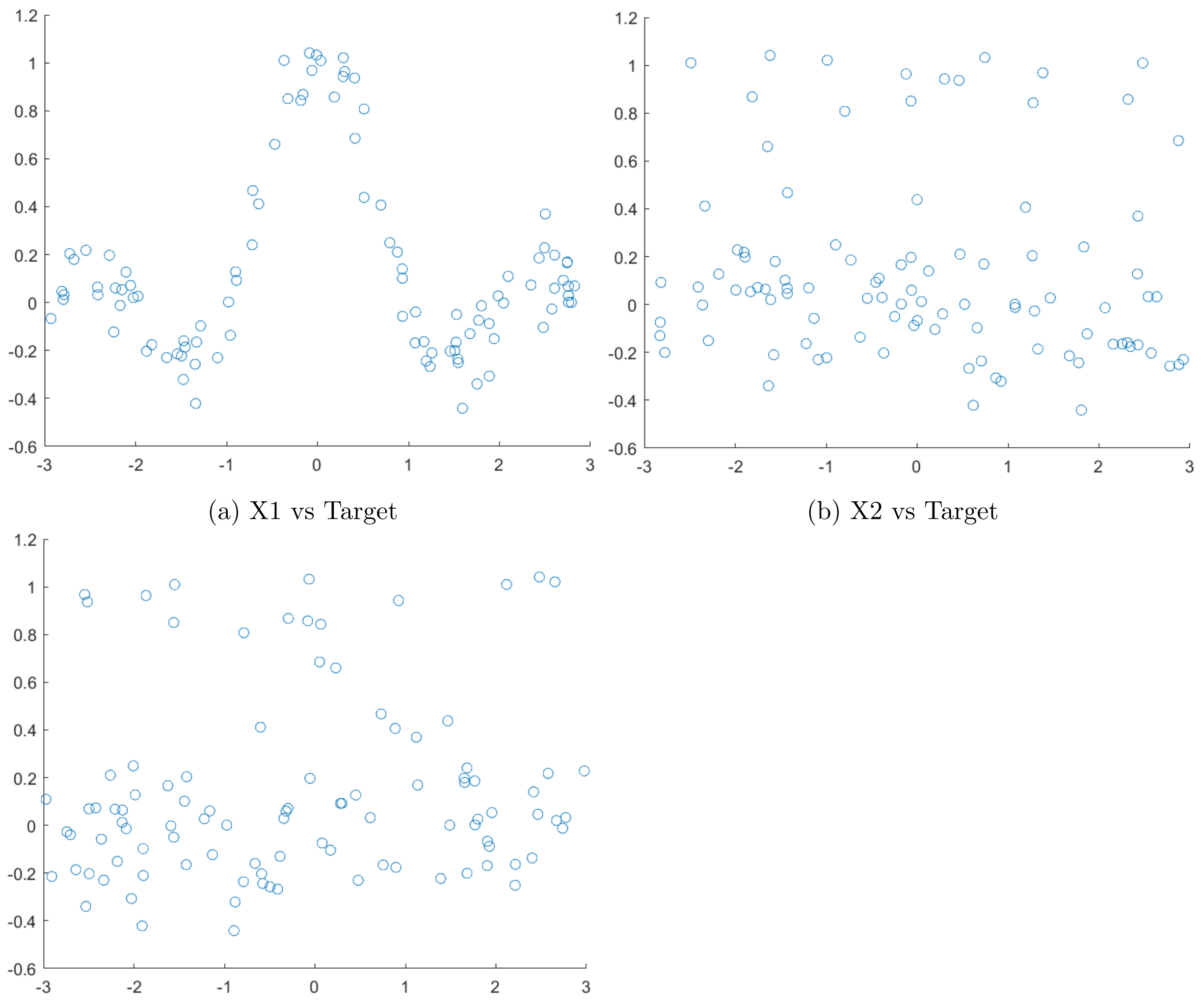
```
[~, sig2 ] = bay_optimize ({ Xtrain , Ytrain , 'f', gam , sig2 }, 3)
```

To summarize, The likelihood at a certain level equals the evidence at the previous level. In this way, by gradually integrating out the parameters at different levels, the subsequent levels are linked to each other.

1.3 Automatic Relevance Determination

• ***Visualize the results in a simple figure.***

Automatic Relevance Determination (ARD) is used to get the most relevant inputs for the model within the Bayesian framework. In this case, only the first variable seems to be important to get a good representation of the target variable. Figure 5 shows the scatter plot of each input variable with the response variable. It is very clear that the first variable is the one that has most of the information to get an approximation of the underlying function that generated the data, so the results are consistent with the graphical representation.



(a) X1 vs Target

(b) X2 vs Target

(c) X3 vs Target

Figure 5: Scatter plot of each input variable vs target variable

- ***How can you do input selection in a similar way using the crossvalidate function instead of the Bayesian framework?***

  According to the toolbox user's guide, `crossvalidate` function estimates the model performance with k-fold crossvalidation. This function can be used to get the model performance of each possible model that can be fitted with these variables and then select those variables that led to the best solution. In Matlab this look like:

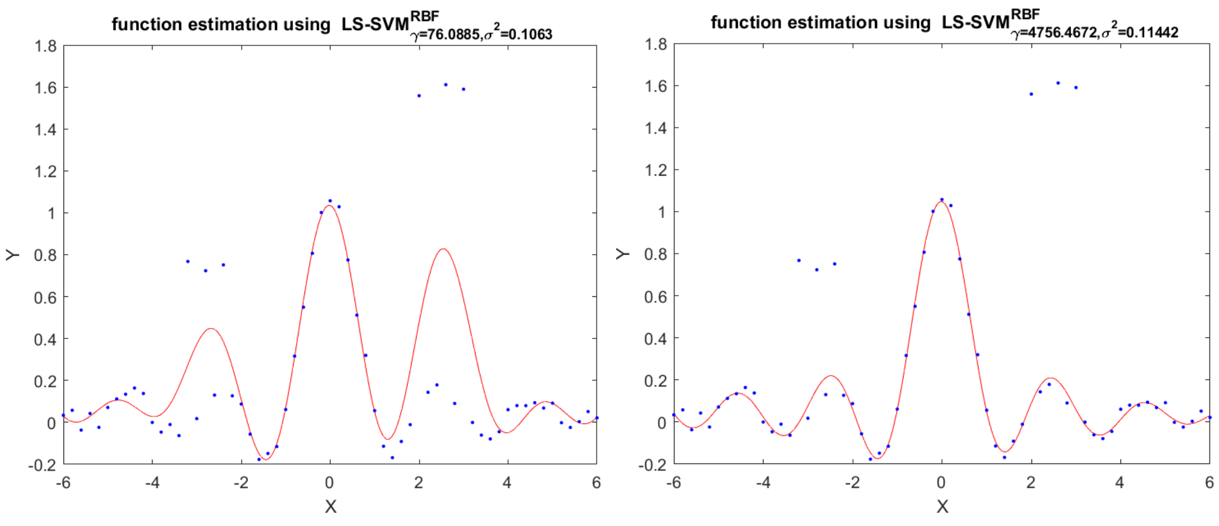
```
cost1 = crossvalidate({X(: ,1),Y,'f',gam,sig2}) % cost1 = 0.0113
cost2 = crossvalidate({X(: ,2),Y,'f',gam,sig2}) % cost2 = 0.1647
cost3 = crossvalidate({X(: ,3),Y,'f',gam,sig2}) % cost3 = 0.1769
cost4 = crossvalidate({X(: ,[1 2]),Y,'f',gam,sig2}) % cost4 = 0.0219
cost5 = crossvalidate({X(: ,[1 3]),Y,'f',gam,sig2}) % cost5 = 0.0200
cost6 = crossvalidate({X(: ,[2 3]),Y,'f',gam,sig2}) % cost6 = 0.2371
cost7 = crossvalidate({X(: ,[1 2 3]),Y,'f',gam,sig2}) % cost7 = 0.0600
```

  As a result, the best performance is given by the model that only considers the first input variable. Hence, these results are consistent with ARD where only the first variable is chosen.

1.4 Robust regression

- ***Visualize and discuss the results. Compare the non-robust version with the robust version. Do you spot any differences?***

  Figure 6 shows the difference in the approximated function when special attention is given to outliers and when this does not happen. The first plot gives the same weight to all observations in the dataset, even when outliers are present. As a result, the approximated function is highly influenced by those points located in the top of the plot, losing the underlying process that generated the data. In the second plot, Huber weights are used in the training process with the purpose of downweight the influence of these outliers. As a consequence, the approximated function is close to the real function in the whole range of the input variable.



(a) No special attention to outliers (b) Special attention to outliers

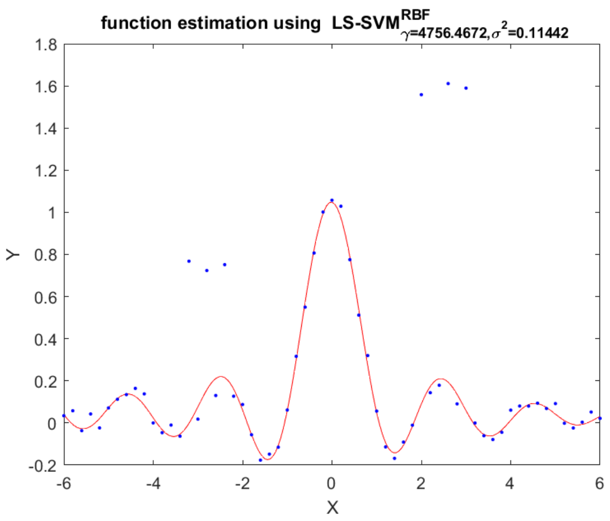
Figure 6: Approximated function with outliers

- ***Why in this case is the mean absolute error ('mae') preferred over the classical mean squared error ('mse')?***

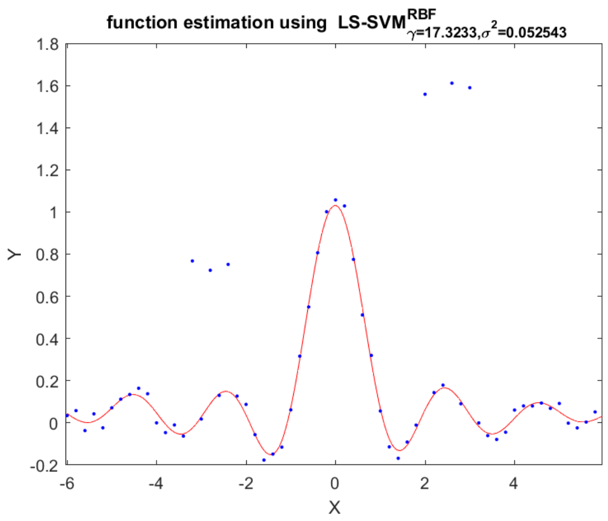
  By definition, MSE penalizes more those points that have large deviations from the estimated function. Square a big number, and it becomes much larger with respect to the others. This means that the training process gives more emphasis to those outlier values, impacting the overall approximated function. On the other hand, the MAE just remove the sign of the deviation of a specific point with respect to the estimated function. In this way, the impact of the outliers in the approximated function is not significant.

- ***Try alternatives to the weighting function wFun (e.g., 'whampel', 'wlogistic' and 'wmyriad'). Report on differences. Check the user's guide of LS-SVMlab for more information.***

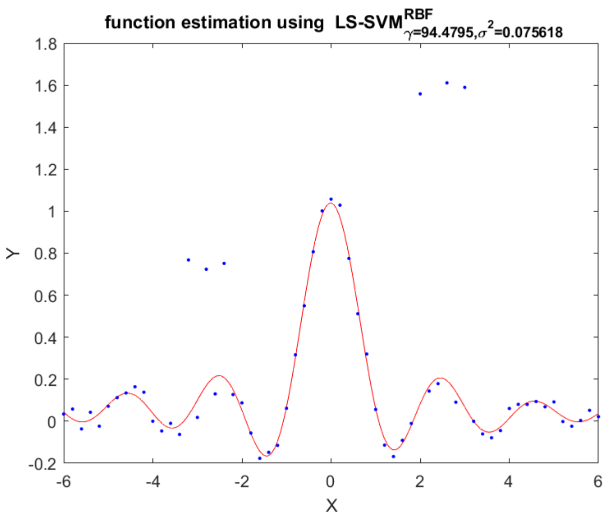
  Figure 7 shows the approximated function using different weighting schemes. It seems that there are not large differences between choosing one kind of weights or the other. In some cases the approximated function is higher when the outliers are located, in the ranges [-4,-2] and [1,3], but these differences are barely noticeable.



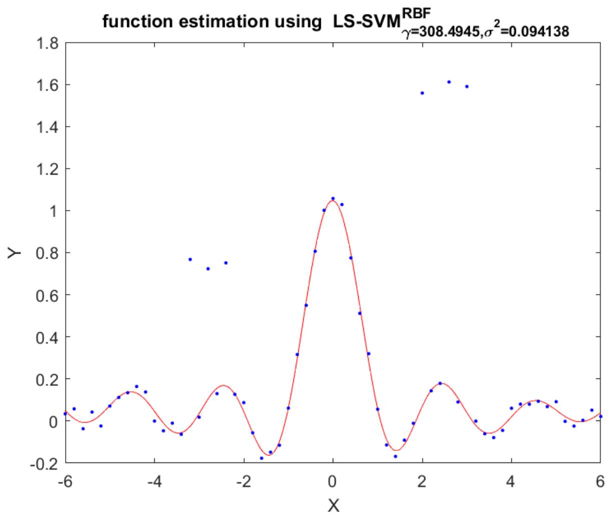
(a) Huber



(b) Hampel



(c) Logistic



(d) Myriad

Figure 7: Approximated function with outliers

2 Homework problems

2.1 Logmap dataset

- ***In Figure 8, the data points of the test set, that is, the actual data points that we want to predict are depicted in black, while the prediction is presented in red. Try to assess the performance of the prediction: do you think it's good? Is there still some room for improvements?***

  To measure the performance of the prediction, the mean squared error was computed by comparing the prediction with the true values. This value is 6.6111, but it is difficult to judge by this number alone whether the prediction is good or not. For this reason, it is better to compare the prediction visually. The prediction was specified to be 50-steps-ahead; in some time-periods the predicted value was really close to the true value, see range between 20 and 32. However, outside this window the prediction is not good enough because when the prediction is high, the true value is low and vice versa. Since the parameters of model were taken arbitrarily, the overall performance can be improved by tuning these parameters.

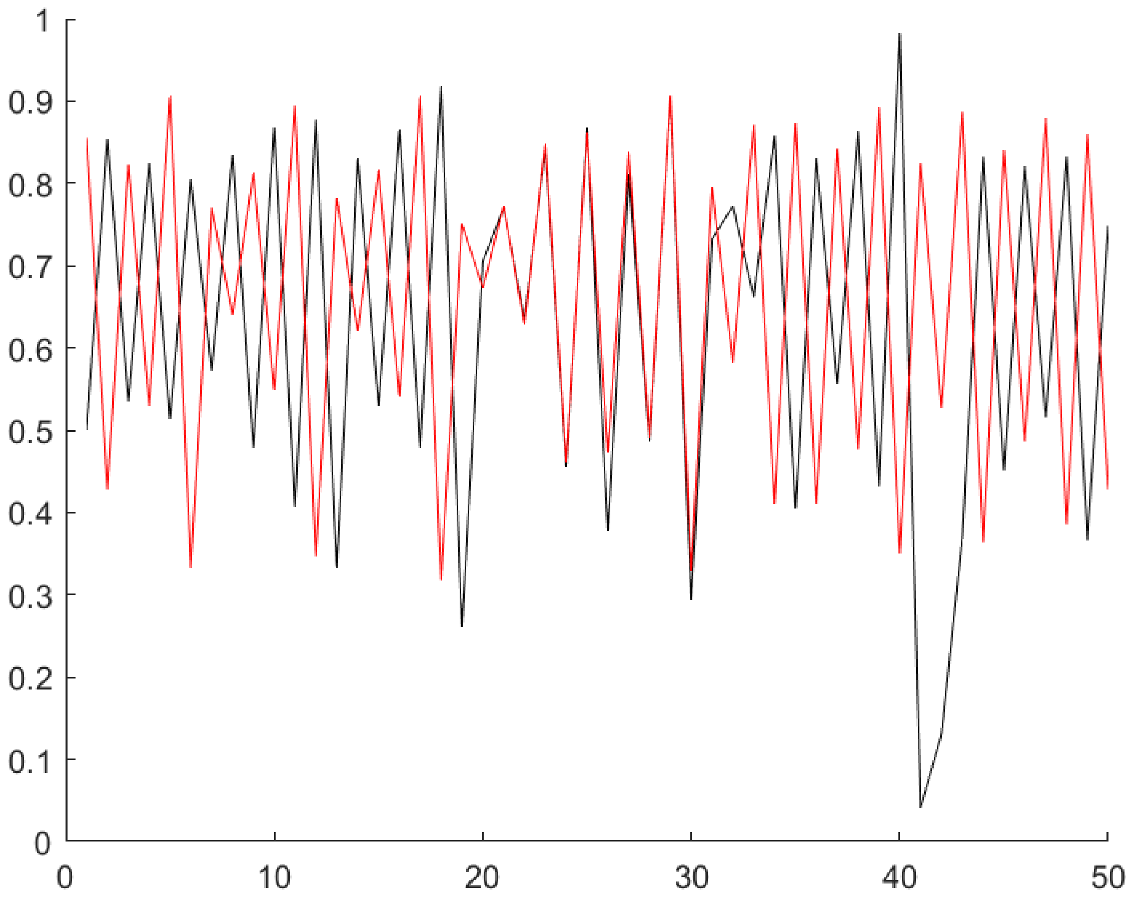


Figure 8: Prediction with arbitrary `gam` and `sig2` values

- ***As indicated numerous times before, the parameters `gam` and `sig2` can be optimized using crossvalidation. In the same way, one can optimize `order` as a parameter. Define a strategy to tune these 3 parameters.***

  As it was mentioned before, to tune parameters `gam` and `sig2`, a combination of Coupled Simulated Annealing and simplex algorithm is used. Since `order` cannot be optimized directly by any function in Matlab, different values should be tried and then choose the one that presents the best performance. In this way, 50 different `orders` were used in the search of the best model for prediction. This strategy works as follows:

  1. Define the order (between 1 and 50)
  2. Shape the data accordingly (using windowize and order)
  3. Tune `gam` and `sig2` with simplex algorithm
  4. Fit the model and predict 50-steps-ahead
  5. Compute the MSE
  6. Choose the order that has the smallest MSE

- ***Do time series prediction using the optimized parameter settings. Visualize your results. Discuss.***

Using the strategy mentioned above, the parameters that produced the best predictions are: `order` = 9, `gam` = 3826.5 and `sig2` = 94.3663. Figure 9 shows the true values in black and the predicted values in red. Even though visually it does not seem to outperform the previous model, where arbitrary parameters were used, the MSE of this model is smaller (3.58). As a consequence, this model is better to make predictions in time window of 50 steps.

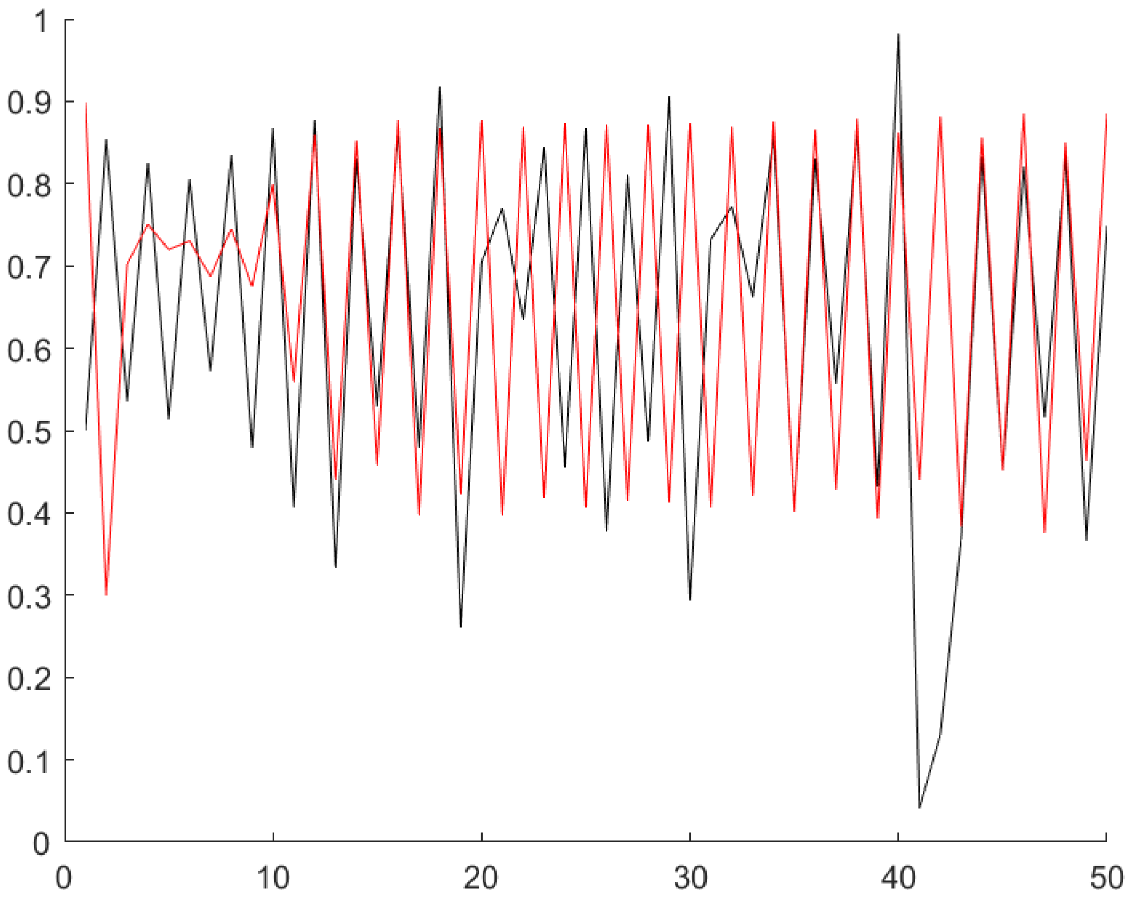
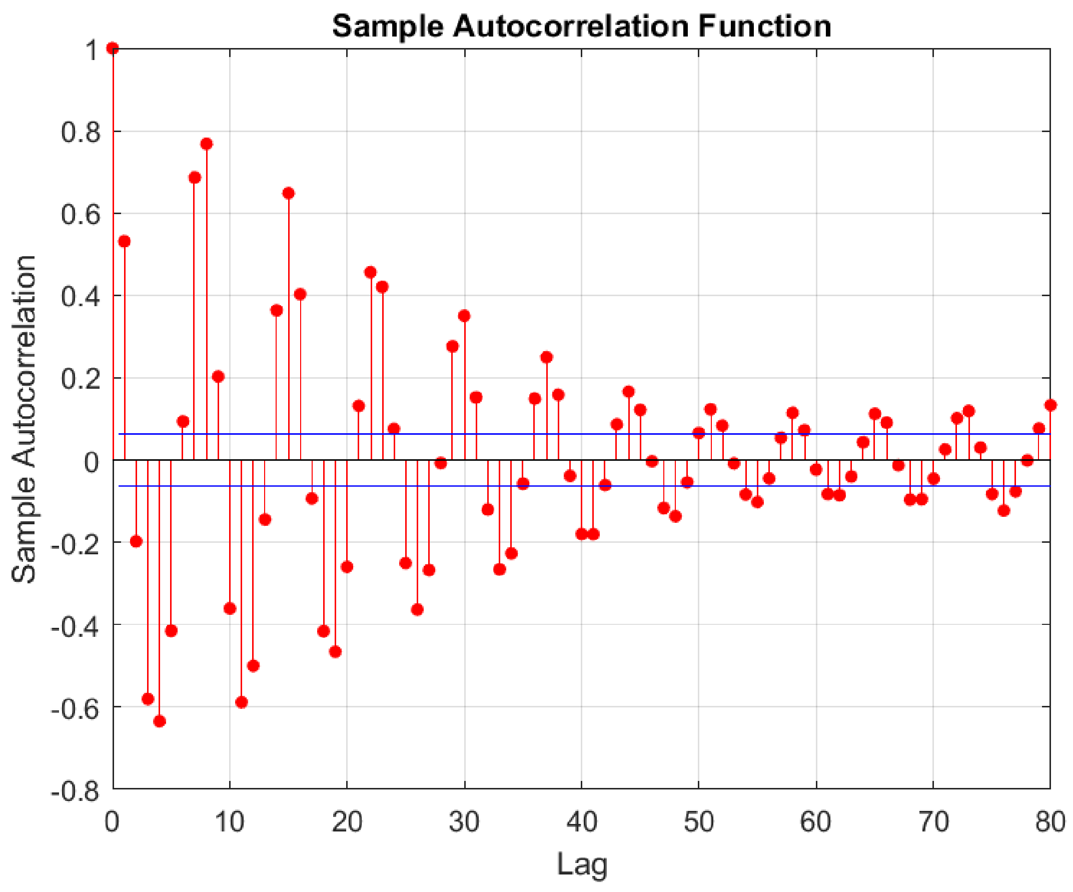


Figure 9: Prediction with tuned parameters

2.2 Santa Fe dataset

- ***Does `order` = 50 for the utilized auto-regressive model sounds like a good choice?***

  In time series problems, the order is specified by looking the correlogram and the partial correlogram (Figure 10). In the correlogram there are still significant autocorrelations at lag 80, indicating that the underlying process can be a moving average of order 80 or more. In the partial correlogram the last significant partial correlation is at lag 29, indicating that an autoregressive process of order 29 can be accurate. Since there are a lot of significant correlations and partial correlations, it can be concluded that 50 could be a good initial choice.



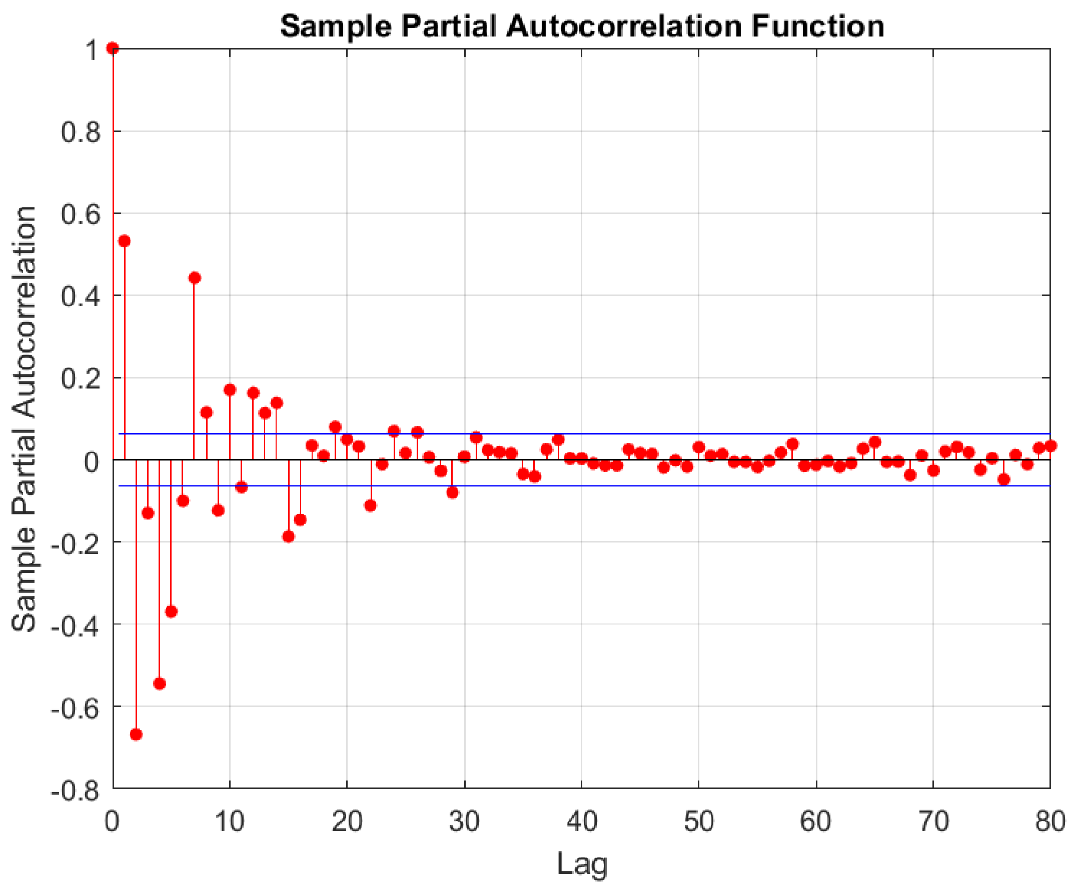


Figure 10: Correlogram and partial correlogram of Santa Fe dataset

- ***Would it be sensible to use the performance of this recurrent prediction on the validation set to optimize hyperparameters and the model order?***

  It depends on how the validation set is taken inside the cross-validation procedure. If the time-correlation is ignored and the validation set is built by randomly selecting observations from the training set, then there are problems in the optimization of the parameters. The first problem is related to the fact that future observations (which in theory are still unobserved) are used to train a model that predicts past observations, if that makes any sense. Second, ignoring the correlation over time may lead to unreliable results.

  In this sense, to optimize the hyperparameters, the cross-validation procedure has to make sure that the validation set comes chronologically after the training subset.

- ***Tune the parameters (`order`, `gam` and `sig2`) and do time series prediction. Visualize your results. Discuss.***

  Using the strategy mentioned before, `order`, `gam` and `sig2` were tuned. As a result, the best model in terms of predictive performance is the one that considers `order` = 55, `gam` = 114.5261 and `sig2` = 37.4373. Figure 11 shows the prediction in red and the true values in black. It can be seen that the prediction in the first 50 periods is almost the same as the true value. From time periods 50 onwards, the prediction is close to the true value, but the model underestimates or overestimates it in a small amount.

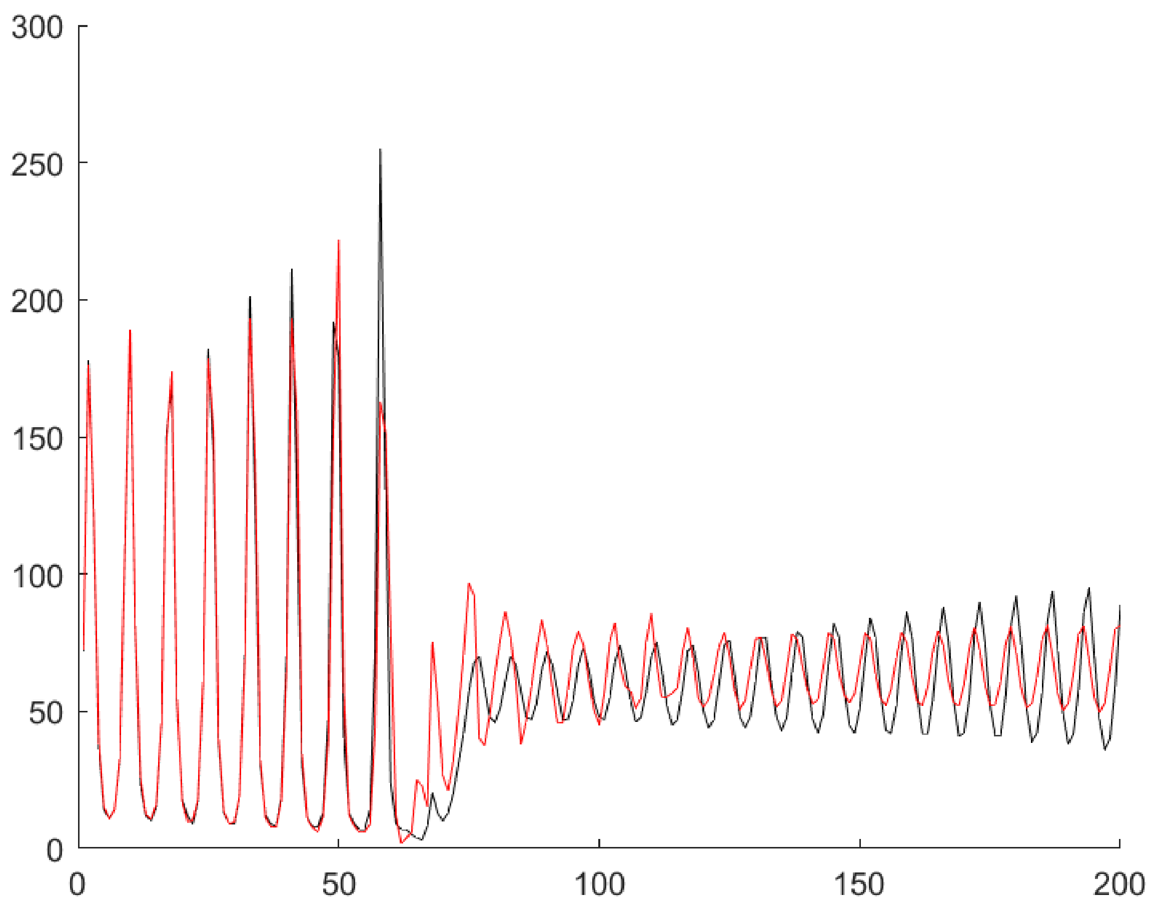


Figure 11: Prediction with tuned parameters