Fall 2021: Advanced Topics in Numerical Analysis:

Numerical Optimization

Final Project: Performance Comparision of L-BFGS, Nonlinear Conjugate Gradient, Stochastic Gradient Descent, and Adam on Multilayer Perceptron

Ly Cao (Netid: lc3940)

1. Linear Conjugate Gradient Descent

1.1. Convergence in the Krylov Subspace

Consider the problem of minimizing a quadratic function of the form:

$$\min_x q(x) = c^T x + x^T H x$$

whose gradient is $g(x) = c + Hx$ and Hessian is H which contains constant entries

Below is a reformulation of theorem 5.3 [5] for the above problem

Theorem 1.1 [5]

Suppose that the kth iterate generated by the conjugate gradient method is not the solution point x^*. The following four properties hold:

- $g_{k+1}^T g_i = 0, i = 0, \ldots, k-1$
- $\text{span}\{g_0, g_1, \ldots, g_k\} = \text{span}\{g_0, Hg_0, \ldots, H^k g_0\}$
- $\text{span}\{p_0, p_1, \ldots, p_k\} = \text{span}\{g_0, Hg_0, \ldots, H^k g_0\}$
- $p_k^T H p_i = 0, i = 0, \ldots, k-1$

There

We say that a set of n linearly independent vectors $\{p_0, p_1, \ldots, p_n\}$ is a conjugate set of the vectors follow these two properties

$$\begin{cases} p_j^T H p_i = 0, i \neq j \\ p_i^T H p_i \neq 0 \end{cases}$$

If H is positive definite, then $p_i^T H p_i > 0$

According to theorem 1.1, every gradient iterate $g_k \in \text{span}\{g_0, Hg_0, \ldots, H^k g_0\}$ and since $g_{k+1}^T g_k = 0$ we must have that either g_{k+1} is linearly independent of $\text{span}\{g_0, Hg_0, \ldots, H^k g_0\}$ or $g_{k+1} = 0$. To see how g_0 and the eigenvalues of H affect the convergence of linear conjugate gradient, consider the case where H is diagonal so that the eigenvalues of H, $\lambda_1, \lambda_2, \ldots, \lambda_n$ are entries in its diagonal and construct the following matrix where we stack up the vectors in Krylov subpspace where $x, g \in R^n, H \in R^{n \times n}$ and let $g_0^1, g_0^2, \ldots, g_0^n$ be the entries of g_0

$$\begin{bmatrix} g_0^1 & g_0^2 & \cdots & g_0^n \\ g_0^1\lambda_1 & g_0^2\lambda_2 & \cdots & g_0^n\lambda_n \\ g_0^1\lambda_1^2 & g_0^2\lambda_2^2 & \cdots & g_0^n\lambda_n^2 \\ \cdots & \cdots & \cdots & \cdots \\ g_0^1\lambda_1^n & g_0^2\lambda_2^n & \cdots & g_0^n\lambda_n^n \end{bmatrix}$$

Algorithm 1 Linear Conjugate Gradient for minimizing q(x)[5]

$k \leftarrow 0; g_0 \leftarrow c + Hx_0$
while $g_k \neq 0$ **do**
  if k=0 **then**
    $p_k \leftarrow g_k$
  else
    $\beta_{k-1} \leftarrow \frac{p_{k-1}^T H g_k}{p_{k-1}^T H p_{k-1}}$
    $p_k \leftarrow -g_k + \beta_{k-1} p_{k-1}$
  end if
  $\alpha_k \leftarrow \frac{-g_k^T p_k}{p_k^T H p_k}$; $x_{k+1} = x_k + \alpha_k p_k$
  $g_{k+1} \leftarrow g_k + \alpha_k H p_k; k \leftarrow k + 1$
end while

Consider row i and j where $i < j$ while pivoting on row j during the Gaussian-Elimination algorithm and let H' be the matrix after pivoting on row j. We want $H'_{ij} = g_0^i \lambda_i^j - \sum_{k,k<j} \alpha_k g_0^i \lambda_i^k = g_0^i(\lambda_i^j - \sum_{k,k<j} \alpha_k \lambda_i^k) = 0$ where α_k are some constants such that we can eliminate column entries $k < j$ in row j using the combinations of kth rows where $k < j$. The pivot position $H_{jj'} = g_0^j \lambda_j^j - \sum_{k,k<j} \alpha_k g_0^j \lambda_j^k = g_0^j(\lambda_j^j - \sum_{k,k<j} \alpha_k \lambda_j^k)$ where the coefficients α_k are exactly the same as those in the equation for H'_{ij} since we use row reduction.

If $H^j g_0$ is linearly independent of $\text{span}\{g_0, Hg_0, \ldots, H^{j-1}g_0\}$, then we can find g_j that satisfies the first result of theorem 1.1 since $g_j \in \text{span}\{g_0, Hg_0, \ldots, H^{j-1}g_0, H^j g_0\}$ and $g_{i,i<j} \in \text{span}\{g_0, Hg_0, \ldots, H^{j-1}g_0\}$ and there is a component of vector $H^j g_0$ that is orthogonal to all vectors $g_{i,i<j}$. Otherwise, $\text{span}\{g_0, Hg_0, \ldots, H^{j-1}g_0, H^j g_0\} = \text{span}\{g_0, Hg_0, \ldots, H^{j-1}g_0\}$ and to satisfy $g_j g_i = 0, i < j$ we must have that $g_j = 0$ and linear conjugate gradient converges at iteration j.

Coming back to our constructed matrix, we notice that either when $g_0^j = 0$ or $\lambda_i = \lambda_j$ or $\lambda_j = 0$, $H'_{ij} = H'_{jj} = 0$, so $H^j g_0 \in \text{span}\{g_0, Hg_0, \ldots, H^{j-1}g_0\}$ and $g_j = 0$. Therefore, the convergence of linear conjugate gradient not only depends on how many distict eigenvalues of H, the ordering of these eigenvalues on the diagonal entries of H (whether similar eigenvalues occur on the earlier rows of H), and whether there is any zero entry in the starting gradient g_0. Since the dimension of any Krylov subspace in iterations $1, 2, \ldots, k$ is at most n, and g_{k+1} must be orthogonal to all the previously generated Krylov subspace, $g_{k+1} = 0$ in at most n iterations.

1.2. Extension to Nonlinear Case

Another way to look at linear conjugate gradient is to consider the gradient equation itself $g(x) = c + Hx$. Consider again the situation when H is diagonal, then we can find the solution in n iterations where each solve the equation $c_i = -\lambda_i x_i$ for the optimal point x. When H is not diagonal, the problem is harder since we have to find the right combinations of the columns of H that result in -c where the cofficients for column ith in H is given by x_i. When H is a constant matrix, we have n linear equations to solve for the root of g(x) which stays the same as x changes, which is trivial to solve (but coming up with n linearly independent models is dificult, which we now have a solution such as using linear conjugate gradient). When H is not a constant matrix, or H is nonlinear, the set of n ideal linear models change as x changes, so finding an iterative algorithm in this case is challenging.

Notice that in Linear conjugate gradient, when $g_{k+1} = 0$, then $\alpha_k H p_k + g_k = 0$ or $\alpha_k H p_k = -g_k$ which has the form of Newton method $\nabla^2 f_k p_k^N = -\nabla f_k$ [5] so linear conjugate gradient

is an iterative algorithm that makes use of the constant Hessian of function q(x) to solve for the approximate solution to Newton step. However, deriving a step using Hessian is not always possible when the function we're considering is nonlinear. For example, the gradient Hessian for function $x^{\frac{1}{3}}$ is undefined at 0 ($\lim_{x\to 0} = \infty$) but the gradient is $\frac{1}{3}x^{\frac{-2}{3}}$ and the Hessian $\frac{-2}{9}x^{\frac{-5}{3}}$ are continuous.

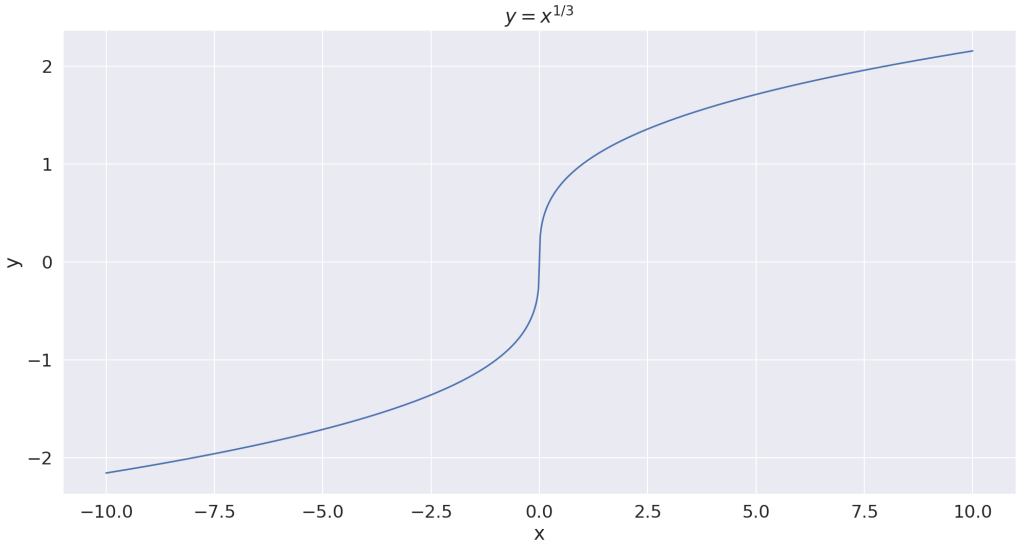


Figure (1) $y = x^{\frac{1}{3}}$

2. **Nonlinear Conjugate Gradient Descent**

As another example, consider the function $f(x) = e^{-x_1x_2}$ whose gradient is $g(x) = \begin{bmatrix} -x_2e^{-x_1x_2} \\ -x_1e^{-x_1x_2} \end{bmatrix}$ and Hessian is $\begin{bmatrix} x_2^2e^{-x_1x_2} & -e^{-x_1x_2} + x_1x_2e^{-x_1x_2} \\ e^{-x_1x_2} + x_1x_2e^{-x_1x_2} & x_1^2e^{-x_1x_2} \end{bmatrix}$ and we notice that both coordinates of g(x) are convoluted in the sense that $g_1(x)$ and $g_2(x)$ contains both x_1 and x_2 and so are the entries of H. There isn't a simple way to separate the two coordinates so that we can find solution by solving in each coordinate direction independently of the other like how we could by developing linear models using the Hessian as in 1.1. If we can find a transformation of variables (including increasing the number of variables) such that each coordinate of g(x) contains only one variable, then we can hope to develop a method to solve for g(x) in each coordinate independently and terminate in at most the number of steps equal to the new number of variables. But linearizing a nonlinear problem may raise the number of variables significantly, so transforming and solving for each coordinate may not be a feasible solution.

Notice that in algorithm 1, the steps p_k are pre-computed and used to develop the approximations of the next gradients using linear models to solve $g(x) = 0$, what if instead we develop the iterates by replacing H with $H(x_k)$ so that $g_{k+1} = g_k + \alpha H_k p_k$ and deriving p_k such that $p_{k+1}^T H_k p_k = 0$? This results in the nonlinear conjugate gradient method as described in [5]

Algorithm 2 produces search directions that are conjugate with respect to the average Hessian

Algorithm 2 The Hestenes-Stiefel formula for Nonlinear Conjugate Gradient

Given x_0;
Evaluate $f_0 = f(x_0), \nabla f_0 = \nabla f(x_0)$
Set $p_0 \leftarrow -\nabla f_0, k \leftarrow 0$
while $\nabla f_k \neq 0$ **do**
Compute α_k and set $x_{k+1} = x_k + \alpha_k p_k$;
Evaluate ∇f_{k+1};

$$\beta_{k+1}^{HS} \leftarrow \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{(\nabla f_{k+1} - \nabla f_k) p_k}; \tag{1}$$

$$p_{k+1} \leftarrow -\nabla f_{k+1} + \beta_{k+1}^{HS} p_k; \tag{2}$$

$$k \leftarrow k + 1; \tag{3}$$

end while

over the segment $[x_k, x_{k+1}]$ defined as

$$\bar{G}_k \equiv \int_0^1 |\nabla^2 f(x_k + \tau \alpha_k p_k)| d_\tau$$

so that $p_{k+1} \bar{G}_k p_k = 0$. From Taylor's theorem, we also have that $\nabla f_{k+1} = \nabla f_k + \alpha_k \bar{G}_k p_k$ or $\bar{G}_k p_k = \nabla f_{k+1} - \nabla f_k$ (*)

$$p_{k+1} \bar{G}_k p_k = 0 \tag{4}$$

$$\Leftrightarrow -\nabla f_{k+1}^T \bar{G}_k p_k + \beta p_k^T \bar{G}_k p_k = 0 \tag{5}$$

$$\Leftrightarrow \beta = \frac{f_{k+1}^T \bar{G}_k p_k}{p_k^T \bar{G}_k p_k} \tag{6}$$

$$\Leftrightarrow \beta = \frac{f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{p_k^T (\nabla f_{k+1} - \nabla f_k)} \text{ from (*)} \tag{7}$$

Here is my attempt to solve for

$$x^* = \min_x f(x) = e^{-x_1 x_2}$$

using NonlinearCG with β^{HS} whose true solution approaches 0

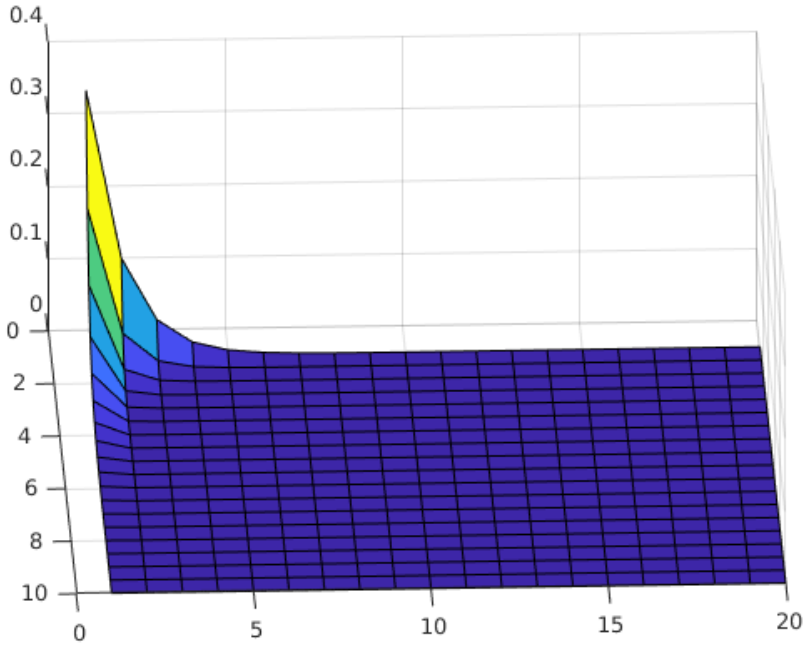


Figure (2) The function $e^{-x_1 x_2}$

The solution x^* is colored green in the following figure

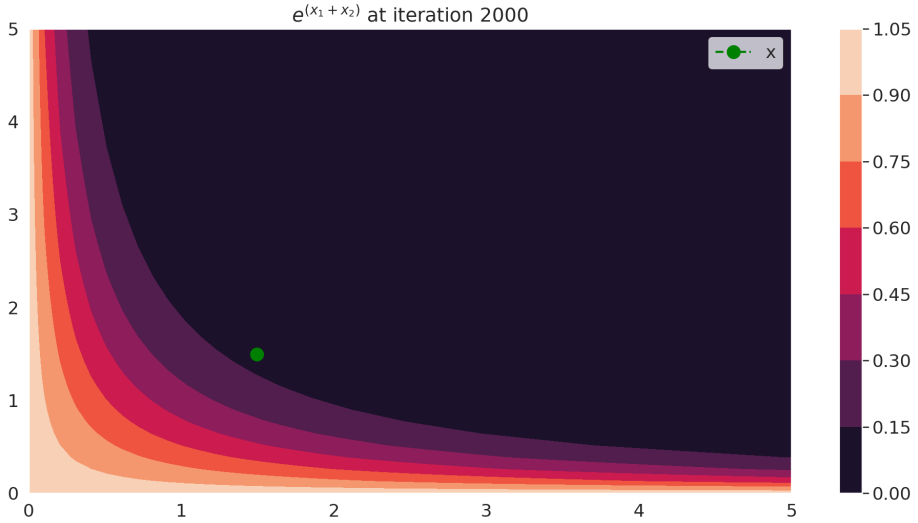


Figure (3) Contour graph of the solution for $e^{-x_1 x_2}$

We notice that algorithm 2 did not solve the problem well enough since our solution is in the dark region where the value approaches 0 using ftol $= 1e - 14$ as a termination condition for the change in loss functions. The number of iterations 2000 displayed means that I reran algorithm 2 2000 times.

Here is my attempt to construct the matrix consists of the gradients computed during iterations

of algorithm 2

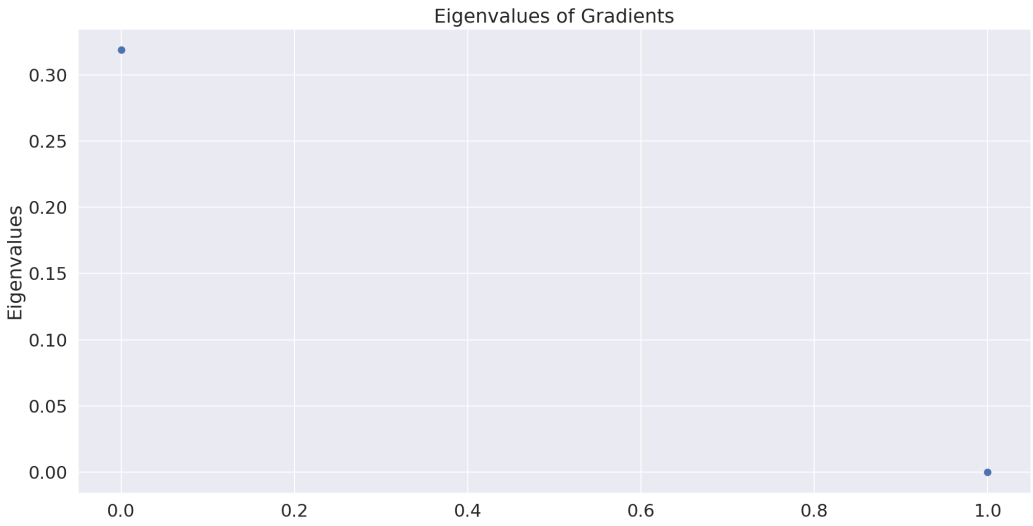


Figure (4) Contour graph of the solution for $e^{-x_1 x_2}$

So there are only 2 eigenvalues, as the dimension of g(x) is 2, but the true linearized version of g(x) may contain a much larger number of variables, which is probably why we need to rerun the algorithm up to maxit = 2000 times and to see a good approximation to the minimal value of f(x), but yet to solve to problem exactly. The final value at the solution is 0.10652362657283636, which is not close to ftol as we would expect for the real solution.

Two other variations of β are Fletcher-Reeves method which sets $\beta_{k+1}^{FR} = \frac{\nabla f_{k+1}^T f_{k+1}}{f_k f_k^T}$ and Polak-Ribiere method which sets $\beta_{k+1}^{PR} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{||\nabla f_k||^2}$.

A line search for step α_k in algorithm 2 satisfies strong Wolfe conditions if it satisfies two conditions

$$\begin{cases} f(x_k + \alpha_k p_k) & \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k \\ |\nabla f(x_k + \alpha_k p_k)^T p_k| & \leq c_2 \nabla f_k^T p_k \end{cases}$$

Lemma 5.6 in [5] states that if α_k is chosen so that strong Wolfe conditions are satisfied and we set $0 < c_2 < \frac{1}{2}$, then Fletcher-Reeves method produces a descent direction at every iteration. The step defined by Fletcher-Reeves method is $p_k = -\nabla f_k + \beta_k^{FR} \nabla f_k^T p_{k-1}$ and multiplying the expression by ∇f_k^T, we have: $\nabla f_k^T p_k = -||\nabla f_k||^2 + \beta_k^{FR} \nabla f_k^T p_{k-1}$ so if $\beta_k^{FR} f_k^T p_{k-1}$ is positive and dominates the first term, then Fletcher-Reeves method may produce an ascent direction. Even though lemma 5.6 showed a safeguarded way to use FR algorithm, the author of [5] has suggested to use the following variation for β

$$\begin{cases} -\beta_k^{FR} \text{ if } \beta_k^{PR} & < -\beta_k^{FR} \\ \beta_k^{PR} \text{ if } |\beta_k^{PR}| & < \beta_k^{FR} \\ \beta_k^{FR} \text{ if } v\beta_k^{PR} & > \beta_k^{FR} \end{cases}$$

following the argument that global convergence is guaranteed for $\beta_k \leq \beta_k^{FR}$ and β^{PR} tends to perform better than β^{FR} in practice

3. **L-BFGS**

Algorithm 3 L-BFGS [5]

$q \leftarrow \nabla f_k$;
for **do**$i = k-1, k-2, \ldots, k-m$
  $\alpha_i \leftarrow \rho_i s_i^T q$;
  $q \leftarrow q - \alpha_i y_i$;
end for
$r \leftarrow H_k^0 q$;
for **do**$i = k-m, k-m-1, \ldots, k-1$
  $\beta \leftarrow \rho_i y_i^T r$;
  $r \leftarrow r + s_i(\alpha_i - \beta)$
end for
stop with result $H_k \nabla f_k = r$

L-BFGS was derived from the form of BFGS which produces B as an approximation to the true Hessian from which its inverse which will be referred to below as H can be derived [7] and satisfy:

$$x_{k+1} = x_k - \alpha_k H_k \nabla f_k \tag{8}$$

where

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T$$

and

$$\rho_k = \frac{1}{y_k^T s_k}, V_k = I - \rho_k y_k s_k^T$$

and

$$s_k = x_{k+1} - x_k, y_k = \nabla f_{k+1} - \nabla f_k$$

where the full approximation

$$H_k = (V_{k-1}^T \ldots V_1^T) H_k^0 (V_1 \ldots V_{k-1})$$
$$+\rho_1 (V_{k-1}^T \ldots V_2^T) s_1 s_1^T (V_2 \ldots V_{k-1})$$
$$+\rho_2 (V_{k-1}^T \ldots V_3^T) s_2 s_2^T (V_3 \ldots V_{k-1})$$
$$+ \ldots$$
$$+\rho_{k-1} s_{k-1} s_{k-1}^T$$

is restricted to m last pairs of $(s_{k-1}, y_{k-1}), \ldots (s_{k-m}, y_{k-m})$ to update H_k or

$$H_k = (V_{k-1}^T \ldots V_{k-m}^T) H_k^0 (V_{k-m} \ldots V_{k-1})$$
$$+\rho_{k-m} (V_{k-1}^T \ldots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \ldots V_{k-1})$$

$$+\rho_{k-m+1}(V_{k-1}^T \dots V_{k-m+2}^T)s_{k-m+1}s_{k-m+1}^T(V_{k-m+2} \dots V_{k-1})$$
$$+\dots$$
$$+\rho_{k-1}s_{k-1}s_{k-1}^T$$

Notice that since we only need the matrix-vector product $H_k \nabla f_k$ to compute the next iterate x_{k+1}, algorithm 3 provides a way to do so. To see why it works, first divide the terms in two parts, $(V_{k-1}^T \dots V_{k-m}^T)$ and $H_k^0(V_{k-m} \dots V_{k-1})$, $(V_{k-1}^T \dots V_{k-m+1}^T)\rho_{k-m}s_{k-m}s_{k-m}^T$ and $(V_{k-m+1} \dots V_{k-1})$, $(V_{k-1}^T \dots V_{k-m+2}^T)\rho_{k-m+1}s_{k-m+1}s_{k-m+1}^T$ and $(V_{k-m+2} \dots V_{k-1}), \dots, \rho_{k-1}s_{k-1}s_{k-1}^T$ and empty. Now the first loop and the middle part between 2 loops are used to compute the right component of each part multiply by ∇f_k and the second loop is used to compute the left component of each part. We have

$V_{k-1}\nabla f_k = \nabla f_k - \rho_{k-1}y_{k-1}s_{k-1}^T \nabla f_k$ which is what computed in Operation 4, so we update the new residual as q and continue multiplying q on the left by V_{k-2} if it exists in the right component we're considering

Consider the backwards loop by looking at an example

$V_{k-m+1}^T \rho_{k-m}s_{k-m}\rho_{k-m}s_{k-m}^T q = \rho_{k-m}s_{k-m}s_{k-m}^T q - \rho_{k-m+1}s_{k-m+1}y_{k-m+1}^T \rho_{k-m}s_{k-m}s_{k-m}^T q$ and let $r = \rho_{k-m}s_{k-m}s_{k-m}^T q$ then $V_{k-m+1}^T s_{k-m}\rho_{k-m}s_{k-m}^T q = s_{k-m}\alpha_{k-m} - \rho_{k-m+1}s_{k-m+1}y_{k-m+1}^T r$ which we cannot compute since indices $k-m$ and $k-m+1$ are different and should be in different iterations of the loop. However, we can compute $s_{k-m}\alpha_{k-m}$ from the current term and $-\rho_{k-m+1}s_{k-m+1}y_{k-m+1}^T r$ version of the previous term which is $-\rho_{\rho k-m}s_{\rho k-m}y_{\rho k-m}^T r$. An example of this would be to compute $I@s_{k-m}s_{k-m}^T q$ of the second term and $(-\rho_{k-m}s_{k-m}y_{k-m}^T)@H_k^0 q$ of the first term and let the subtraction in the second term be computed along with the addition of the third term. Notice how the last term $\rho_{k-1}s_{k-1}s_{k-1}^T$ doesn't have a subtraction term since it never multiply by a matrix V, the algorithm will add up all terms properly at the end of the second loop.

Since $s_k = x_{k+1} - x_k = -\alpha_k H_k \nabla f_k$, $s^T \nabla f_k = -\nabla f_k H \nabla f_k$ and H_k is guaranteed to be positive definite over the iterations so that $\nabla f_k^T s < 0$ and the steps produced by L-BFGS are guaranteed to be a descent step.

4. **Stochastic Gradient Descent**

In machine learning, each update to the parameters are computed using backpropagation on a loss function over the data in the training dataset that has n samples. Define the loss function as

$$f(x; w) = \frac{1}{n}\sum_i f_i(x_i; w) \tag{9}$$

where x_i is a sample in the dataset and w is the parameter, then the update for w using backpropagation in the kth iteration is

$w = \sum_i w - \eta_k \frac{1}{n}\nabla_w f(x; w)$ where η_k is the learning rate determined from from scheduler algorithm

If we instead just choose a random sample and the associated loss $f_i(x; w)$ to update w, and let X_i be the indicator that sample x_i is picked for the update, Prob$(X_i) = \frac{1}{n}$ so that the new loss function becomes $E(f(x; w)) = \sum_i Prob(X_i)f_i(x_i; w) = \frac{1}{n}\sum_i f_i(x_i; w)$ which is the same as the original loss function defined in (9). Therefore, if the number of iterations is large enough, our update in expectation will be equivalent to when we use the whole dataset for every update. However, if the sample we chose is a perturbed point, then we may choose an ascent step at an iteration, so stochastic gradient descent doesn't guarantee a descent step [7].

5. **Adam**

   Adam is a variation of stochastic gradient descent and so is also not a descent algorithm. The algorithm can be found at [3].

6. **Experiments**

   6.1. **Dataset**

   The dataset consists of Date, Low, High, Close price, Volume, and Stock Trading of Uniqlo stock in 2016 and was obtained via Kaggle [4]. The dataset contains 1226 samples which is splitted into 70% training and 30% test which is equivalent to 858 and 368 samples in each set. The dataset is explored using XGBoost as in [1] and Lasso Regression to determine the relationship between features and extract important ones for training.

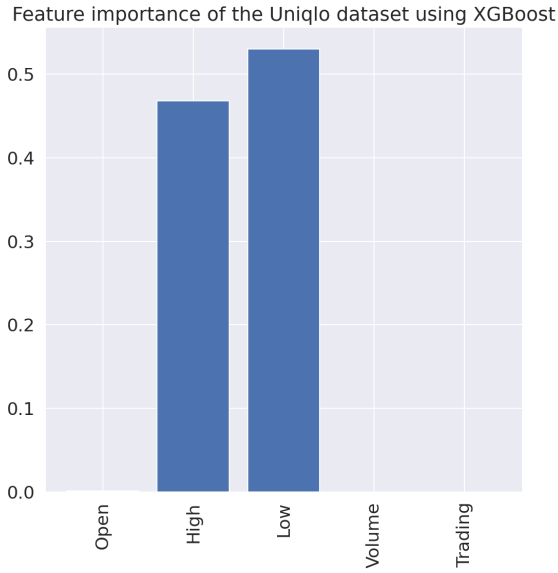


Figure (5) Feature Importance with XGBoost

From Figure (5), we can see that Low and High are the most important features to predict Close while Open, Volume, and Stock Trading carry almost no weight in determining the Closing price.

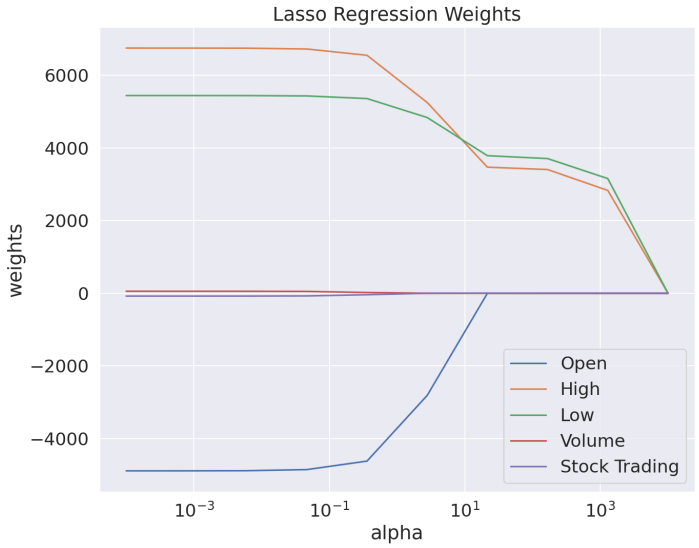


Figure (6) Feature Importance with Lasso

Figure (6) confirms that Low and High are the most important features, as regularized parameter α increases, Open decreases to 0 quicklier than Low and High. On the other hand, Volume and Stock Trading remains close to 0 even when α is small, indicating that there is no relationship between Volume, Stock Trading and CLose.

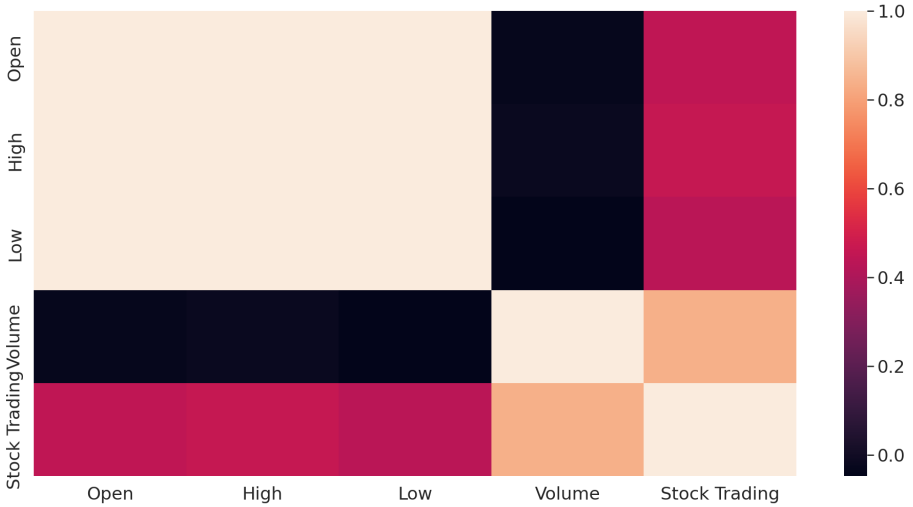


Figure (7) Feature Correlation

Figure (7) shows that Low, High, Open are highly correlated while Volume and Stock Trading are highly correlated with each other, but barely correlated with Low, High, and Open. With this exploratory procedure, I have chosen Low and High as features for my models MLP and MLPLarge as will be described in section 6.2

6.2. **Model**

The first model is a Multilayer Perceptron that consists of 2 hidden layers where the first one has 100 neurons and the second one has 500 neurons. Each fully-connected layer is followed by a ReLU activation $f(x) = \max(0, x)$. The input of this model will be dataset where low and high of the stock are chosen as features and the output is the closing price of the stock. This model will be referred to as MLP and has **101001** parameters. The original dataset is used which has 858 and 368 samples in training and test set respectively used in the experiments with different learning rates for each optimizer.

The second model is also a Multilayer Perceptron that also contains 2 hidden layers where the first layer contains 200 neurons and the second layer contains 1000 neurons and ReLU was also used as an activation function. The input and output of this model are the same as those of MLP's. This model will be referred to as MLPLarge and has **402001** parameters. The dataset used for this model is the same as that in MLP.

The third model uses input as a n_{in} time steps of the close price and output the next 2 time step of the close price. The dataset is transformed properly and the n_{in} time steps as features. The model also consists of 2 hidden layers where the first layer has 200 neurons followed and the second layer has 500 neurons. Both are followed by ReLu activation. This model will be referred as MLPMultistep along with n_{in} time steps. The experiments used time steps 3,5,10,20 in which each is considered a separate model. MPMultistep with 3,5,10,20 steps has **201902, 202302, 203302, and 205302** parameters respectively. The number of samples in (training set, test set) are (171, 73), (122,52), (71,30), and (39,16) respectively. Notice how as the number of time step increases, there are less samples in the transformed training and test set. We will see in section 6.3 that the decreased number of dataset affects these models' performance significantly.

The model will optimize the corresponding datasets using the four optimizers LBFGS, NonlinearCG (HS and `FR_PR`), SGD, and Adam over mean squared error loss function defined as $f(y) = \frac{1}{2}(y - y_{true})^2$

6.3. Software

The software I used to run experiments was [6] where LBFGS, SGD, and Adam are built in, and MLP, MLPLarge, MLPMultistep, and Nonlinear Conjugate Gradient algorithms (HS and `FR_PR`) are coded using the PyTorch library.

6.4. Analysis

The tables below will use shortcuts maxit, `func_calls`, `n_iter`, lr no. params $<$ 1e-6, MLPMultistep3, MLPMultistep5, MLPMultistep10, MLPMultistep20 to denote the maximum number of iterations set for optimization steps, the number of function calls used in each optimization step over the iterations (only applicable to LBFGS and Nonlinear Conjugate Gradient), the number of iterations in each optimization step over the iterations, the number of parameters that are essentially 0 (with a ftol of 1e-6), MLPMultistep model with 3,5,10, and 20 time steps as input features. In this section, I will comment on the behaviour of the four optimization algorithms as shown in the graphs and tables produced during my experiments.

optimizer	model	lr	total time	func_calls	n_iter	maxit	no. params < 1e-6
LBFGS	MLP	0.0001	19.33	3009	598	3000	49.555%
LBFGS	MLP	0.001	21.194	3011	772	3000	50.020%
LBFGS	MLP	0.01	27.951	3022	1110	3000	49.433%
LBFGS	MLP	0.1	38.121	3023	2095	3000	49.969%
LBFGS	MLPLarge	0.0001	60.556	3005	588	3000	49.872%
LBFGS	MLPLarge	0.001	67.297	3009	768	3000	49.965%
LBFGS	MLPLarge	0.01	80.399	3018	1103	3000	49.917%
LBFGS	MLPLarge	0.1	121.01	3015	2143	3000	49.672%
LBFGS	MLPMultistep10	0.0001	3.095	714	144	3000	50.050%
LBFGS	MLPMultistep10	0.001	2.621	512	134	3000	50.001%
LBFGS	MLPMultistep10	0.01	2.56	390	140	3000	49.972%
LBFGS	MLPMultistep10	0.1	2.985	271	168	3000	50.044%
LBFGS	MLPMultistep20	0.0001	0.333	140	29	3000	49.955%
LBFGS	MLPMultistep20	0.001	0.34	118	34	3000	49.941%
LBFGS	MLPMultistep20	0.01	0.346	91	39	3000	49.734%
LBFGS	MLPMultistep20	0.1	0.746	82	72	3000	49.915%
LBFGS	MLPMultistep3	0.0001	19.638	3019	626	3000	50.226%
LBFGS	MLPMultistep3	0.001	23.64	3019	831	3000	51.369%
LBFGS	MLPMultistep3	0.01	30.207	3006	1147	3000	51.311%
LBFGS	MLPMultistep3	0.1	30.817	1976	1308	3000	51.328%
LBFGS	MLPMultistep5	0.0001	17.105	3020	616	3000	50.003%
LBFGS	MLPMultistep5	0.001	20.78	3003	790	3000	50.012%
LBFGS	MLPMultistep5	0.01	27.025	3017	1101	3000	50.376%
LBFGS	MLPMultistep5	0.1	10.816	844	496	3000	50.032%

Table (1) LBFGS Performance Profiler

From table 1, we can see that the total time LBFGS takes is proportional to the number of parameters, as the number of parameters for MLP, MLPMultistep3, MLPMultistep5, MLPMultistep10, MLPMultistep20, MLPLarge are 101001, 201902, 202302, 203302, 205302, and 402001 so the time that MLP, MLPMultistep3, MLPMultistep5, MLPMultistep10, MLPMultistep20 takes are much less than MLPLarge takes since each optimization step of LBFGS requires multiple iterations to calculate a step that is sufficiently descent where each also has to go through more iterations to determine the step length α that satisfies the strong Wolfe conditions. Notice how the family of MLPMultistep models require significantly less time as the number of steps increases. This is due to the significantly reduced number of training datasets as noted in section 6.2, so LBFGS does not have enough information to improve on and has to stop early. As a result, figures 10,11,12,13 reflect that MLPMultistep models perform poorly. In contrast, MLP (figure 8) predicts the test set well while requires much less number of parameters compared to MLPLarge (figure 9), which performs best out of the 6 models on the test set. We also notice that as we set the learning rate smaller, the algorithm runs faster. To understand this behavior, we need to look at the source code [6] and notice that each line search computed α_k for a descent direction p_k in (8) starts with a step length of the learning rate. The smaller the starting point is, the more likely the step is close to a desired step that satisfy strong Wolfe condition in the neighborhood of the current parameter x_k since p_k is a descent step. Thus, each optimization step takes less time. This is reflected in the total number of function calls and total number of iterations of LBFGS in each optimization step over the max number of iteration steps. Looking at figure 8, the smallest learning rate of 0.0001 performs as well as the largest learning rate of 0.1, but we cannot conclude a trend here since the learning rate in between 0.001, 0.01 result in worse performances. In figure 9, however, smaller learning rate seems to help the performance of LBFGS on MLPLarge. We also notice that in comparision to all 3 other algorithms, LBFGS takes a significantly more time, as theoretical result from [5] has noted that LBFGS has a linear convergence rate. Moreover, while SGD and Adam takes a random sample to update per iterate, LBFGS uses all parameters to calculate an update per step.

Lastly, I want to talk about how LBFGS performs so well while only keep m pairs in the history on MLP and MLPLarge. Notice from 1 that most of the parameters are essentially 0, in fact, when setting ftol=1e-9, I got a similar percentage but due to space constraint I have provided external log files that contain more information on the iterates. The activation function used for MLP and MLPLarge are ReLU, which could makes the Hessian very sparse and contain many zeros, which may explain why we don't need to remember too much information from previous iterates to capture the Hessian well enough. We refer to the Hessian expression derived from [2]

$$\frac{\partial^2 E_p}{\partial w_{ij} w_{nl}} = z_j \sigma_n f'(a_l) g_{li} + z_j z_l b_{ni} \tag{10}$$

where

$$\begin{cases} z_j & = f(a_j) \\ a_j & = \sum_i w_{ji} z_j + \theta_j \\ E & = \sum_p E_p \\ \sigma_n & = \frac{\partial E_p}{\partial a_n} \\ g_{li} & = \frac{\partial a_l}{\partial a_i} \\ \beta_{ni} & = \frac{\partial \sigma_n}{\partial a_i} \end{cases}$$

and w_{ji} is the parameter weight that connects layer j and layer i where j $< i$, f(x) is the activation function which in our case is ReLU, E is the error term which in our case is mean squared error. From (10), we see that the Hessian entry is 0 if z_j becomes 0 after being activated by ReLU, confirming the sparse structure that can result from our choice of the activation function.

optimizer	model	lr	total time	func_calls	n_iter	maxit	no. params < 1e-6
NonlinearCG_FR_PR	MLP	0.0001	13.468	3047	582	3000	49.855%
NonlinearCG_FR_PR	MLP	0.001	13.707	3032	774	3000	50.154%
NonlinearCG_FR_PR	MLP	0.01	13.257	3040	1012	3000	49.405%
NonlinearCG_FR_PR	MLP	0.1	9.699	2055	1172	3000	49.944%
NonlinearCG_FR_PR	MLPLarge	0.0001	41.722	3048	591	3000	50.000%
NonlinearCG_FR_PR	MLPLarge	0.001	42.505	3050	733	3000	49.874%
NonlinearCG_FR_PR	MLPLarge	0.01	43.111	3040	1082	3000	49.973%
NonlinearCG_FR_PR	MLPLarge	0.1	45.816	3047	1807	3000	49.951%
NonlinearCG_FR_PR	MLPMultistep10	0.0001	2.071	1238	282	3000	50.342%
NonlinearCG_FR_PR	MLPMultistep10	0.001	1.272	709	217	3000	50.266%
NonlinearCG_FR_PR	MLPMultistep10	0.01	1.99	1010	494	3000	49.834%
NonlinearCG_FR_PR	MLPMultistep10	0.1	5.822	3026	1148	3000	49.920%
NonlinearCG_FR_PR	MLPMultistep20	0.0001	0.044	55	11	3000	49.966%
NonlinearCG_FR_PR	MLPMultistep20	0.001	0.035	46	12	3000	49.728%
NonlinearCG_FR_PR	MLPMultistep20	0.01	0.015	33	12	3000	49.968%
NonlinearCG_FR_PR	MLPMultistep20	0.1	0.004	24	12	3000	49.844%
NonlinearCG_FR_PR	MLPMultistep3	0.0001	8.873	3029	1214	3000	49.931%
NonlinearCG_FR_PR	MLPMultistep3	0.001	9.823	3042	1920	3000	50.175%
NonlinearCG_FR_PR	MLPMultistep3	0.01	5.152	1767	607	3000	50.159%
NonlinearCG_FR_PR	MLPMultistep3	0.1	8.649	3038	802	3000	49.897%
NonlinearCG_FR_PR	MLPMultistep5	0.0001	4.482	2148	607	3000	49.843%
NonlinearCG_FR_PR	MLPMultistep5	0.001	4.815	2053	975	3000	50.006%
NonlinearCG_FR_PR	MLPMultistep5	0.01	1.096	485	237	3000	49.918%
NonlinearCG_FR_PR	MLPMultistep5	0.1	1.233	559	213	3000	49.820%
NonlinearCG_HS	MLP	0.0001	12.387	3028	580	3000	50.169%
NonlinearCG_HS	MLP	0.001	13.937	3048	750	3000	49.693%
NonlinearCG_HS	MLP	0.01	4.158	978	326	3000	49.896%
NonlinearCG_HS	MLP	0.1	15.02	3037	1698	3000	50.146%
NonlinearCG_HS	MLPLarge	0.0001	41.746	3050	598	3000	50.105%
NonlinearCG_HS	MLPLarge	0.001	41.769	3041	757	3000	49.932%
NonlinearCG_HS	MLPLarge	0.01	42.777	3025	1043	3000	49.994%
NonlinearCG_HS	MLPLarge	0.1	46.293	3050	1752	3000	49.898%
NonlinearCG_HS	MLPMultistep10	0.0001	1.515	883	201	3000	50.144%
NonlinearCG_HS	MLPMultistep10	0.001	2.568	1438	416	3000	50.213%
NonlinearCG_HS	MLPMultistep10	0.01	0.79	425	189	3000	50.234%
NonlinearCG_HS	MLPMultistep10	0.1	5.831	3026	1091	3000	50.249%
NonlinearCG_HS	MLPMultistep20	0.0001	0.059	63	13	3000	49.992%
NonlinearCG_HS	MLPMultistep20	0.001	0.039	47	12	3000	49.817%
NonlinearCG_HS	MLPMultistep20	0.01	0.017	35	12	3000	50.126%
NonlinearCG_HS	MLPMultistep20	0.1	0.004	24	11	3000	50.319%
NonlinearCG_HS	MLPMultistep3	0.0001	8.659	3028	935	3000	49.898%
NonlinearCG_HS	MLPMultistep3	0.001	9.542	3052	1612	3000	49.914%
NonlinearCG_HS	MLPMultistep3	0.01	8.485	3031	694	3000	50.288%
NonlinearCG_HS	MLPMultistep3	0.1	7.071	2468	694	3000	49.978%
NonlinearCG_HS	MLPMultistep5	0.0001	5.883	2738	938	3000	49.892%
NonlinearCG_HS	MLPMultistep5	0.001	4.408	1786	840	3000	50.017%
NonlinearCG_HS	MLPMultistep5	0.01	0.562	266	121	3000	49.814%
NonlinearCG_HS	MLPMultistep5	0.1	1.246	563	255	3000	50.105%

Table (2) Nonlinear Conjugate Gradient Performance Profiler

Similar to LBFGS, in MLPLarge, the smaller the step length is, the less time the algorithm takes and the pattern is not so clear in other parameters. In MLP and MLPLarge, LBFGS and both Nonlinear Conjugate Gradient algorithms (HS and FR_PR) have similar number of function calls and optimization iterations, but the total time of Nonlinear Conjugate Gradient is significantly less since computation of each descent step involves only 2 steps in comparision to the 2m recursion loop in LBFGS. From figure (32) to (43), we also notice that both LBFGS and Nonlinear Conjugate Gradient only move in descent steps or we don't see any spike in the loss graph in any model. Figures (16) - (19) also show that Nonlinear Conjugate Gradient algorithms also perform poorly on MLPMultistep model family. The figures in section 6.5.2 also show that β_{HS} and $\beta_{\text{FR\_PR}}$ perform very similarly to each other especially in MLP, MLPLarge, MLPMultistep with 3 layers in predicting the test set and time efficiency. Both can predict the test set well using MLP and MLPLarge, and do poorly on MLPMultistep model, which is properly due to the same reason why LBFGS portrays the same behavior.

optimizer	model	lr	total time	func_calls	n_iter	maxit	no. params < 1e-6
SGD	MLP	0.0001	11.045	N/A	N/A	3000	49.623%
SGD	MLP	0.001	11.508	N/A	N/A	3000	49.789%
SGD	MLP	0.01	11.962	N/A	N/A	3000	49.597%
SGD	MLP	0.2	8.798	N/A	N/A	3000	51.725%
SGD	MLP	0.3	0.649	N/A	N/A	3000	56.149%
SGD	MLP	0.4	0.055	N/A	N/A	3000	67.391%
SGD	MLPLarge	0.0001	35.902	N/A	N/A	3000	49.863%
SGD	MLPLarge	0.001	36.817	N/A	N/A	3000	49.969%
SGD	MLPLarge	0.01	36.51	N/A	N/A	3000	49.831%
SGD	MLPLarge	0.2	36.049	N/A	N/A	3000	55.358%
SGD	MLPLarge	0.3	36.096	N/A	N/A	3000	56.308%
SGD	MLPLarge	0.4	0.085	N/A	N/A	3000	63.065%
SGD	MLPMultistep10	0.0001	2.76	N/A	N/A	3000	49.830%
SGD	MLPMultistep10	0.001	2.716	N/A	N/A	3000	49.701%
SGD	MLPMultistep10	0.01	2.731	N/A	N/A	3000	49.731%
SGD	MLPMultistep10	0.2	1.225	N/A	N/A	3000	51.690%
SGD	MLPMultistep10	0.3	1.109	N/A	N/A	3000	54.822%
SGD	MLPMultistep10	0.4	0.199	N/A	N/A	3000	58.604%
SGD	MLPMultistep20	0.0001	2.326	N/A	N/A	3000	49.902%
SGD	MLPMultistep20	0.001	1.089	N/A	N/A	3000	49.779%
SGD	MLPMultistep20	0.01	0.095	N/A	N/A	3000	49.415%
SGD	MLPMultistep20	0.2	0.034	N/A	N/A	3000	49.726%
SGD	MLPMultistep20	0.3	0.043	N/A	N/A	3000	49.911%
SGD	MLPMultistep20	0.4	0.045	N/A	N/A	3000	49.826%
SGD	MLPMultistep3	0.0001	5.115	N/A	N/A	3000	49.869%
SGD	MLPMultistep3	0.001	5.178	N/A	N/A	3000	49.705%
SGD	MLPMultistep3	0.01	5.309	N/A	N/A	3000	49.973%
SGD	MLPMultistep3	0.2	5.244	N/A	N/A	3000	53.229%
SGD	MLPMultistep3	0.3	4.856	N/A	N/A	3000	53.135%
SGD	MLPMultistep3	0.4	1.345	N/A	N/A	3000	59.329%
SGD	MLPMultistep5	0.0001	3.444	N/A	N/A	3000	49.977%
SGD	MLPMultistep5	0.001	3.461	N/A	N/A	3000	49.847%
SGD	MLPMultistep5	0.01	3.519	N/A	N/A	3000	50.119%
SGD	MLPMultistep5	0.2	2.167	N/A	N/A	3000	53.496%
SGD	MLPMultistep5	0.3	0.203	N/A	N/A	3000	59.227%
SGD	MLPMultistep5	0.4	0.017	N/A	N/A	3000	67.197%

Table (3) SGD Performance Profiler

From table 3, we notice that SGD tends to much less time than LBFGS, and is moderately faster than Nonlinear Congrugate Gradient in all models. The figures in section 6.5.3 show that SGD also perform well on MLP and MLPLarge, while failing to predict using the family of MLPMultistep models (figures (22) - (24)). Figure (20) and (21) shows the importance of choosing an appropriate learning rate since small learning 0.0001 leads to less accurate result and large learning leads to early stopping due to gradient explosion. Although not

demonstrated on the graph, the learning rate of 0.4 and 0.5 led to a explosion in gradient in around iteration 3 to 5, where the loss has exploded to O(1e18). Figures 44e,f, 45d,e,46c,d,e,f,47c,d,e,f,,48d,e,49d,e,f show spikes in the loss over the iterates as we would expect since SGD is not a descent algorithm and can produce ascent directions.

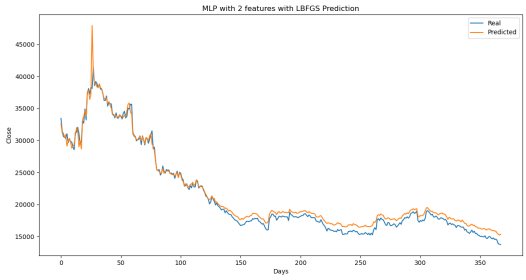
optimizer	model	lr	total time	func_calls	n_iter	maxit	no. params < 1e-6
Adam	MLP	0.0001	9.461	N/A	N/A	3000	49.444%
Adam	MLP	0.001	6.591	N/A	N/A	3000	49.719%
Adam	MLP	0.01	13.816	N/A	N/A	3000	61.278%
Adam	MLP	0.1	13.833	N/A	N/A	3000	74.649%
Adam	MLPLarge	0.0001	27.108	N/A	N/A	3000	49.680%
Adam	MLPLarge	0.001	12.782	N/A	N/A	3000	50.967%
Adam	MLPLarge	0.01	34.456	N/A	N/A	3000	66.122%
Adam	MLPLarge	0.1	0.048	N/A	N/A	3000	63.113%
Adam	MLPMultistep10	0.0001	2.249	N/A	N/A	3000	49.394%
Adam	MLPMultistep10	0.001	0.83	N/A	N/A	3000	50.129%
Adam	MLPMultistep10	0.01	0.683	N/A	N/A	3000	64.113%
Adam	MLPMultistep10	0.1	1.908	N/A	N/A	3000	69.491%
Adam	MLPMultistep20	0.0001	0.27	N/A	N/A	3000	48.990%
Adam	MLPMultistep20	0.001	0.21	N/A	N/A	3000	49.564%
Adam	MLPMultistep20	0.01	0.203	N/A	N/A	3000	56.928%
Adam	MLPMultistep20	0.1	0.314	N/A	N/A	3000	66.060%
Adam	MLPMultistep3	0.0001	7.088	N/A	N/A	3000	49.776%
Adam	MLPMultistep3	0.001	7.202	N/A	N/A	3000	51.222%
Adam	MLPMultistep3	0.01	7.037	N/A	N/A	3000	68.418%
Adam	MLPMultistep3	0.1	7.342	N/A	N/A	3000	73.899%
Adam	MLPMultistep5	0.0001	2.971	N/A	N/A	3000	50.059%
Adam	MLPMultistep5	0.001	1.583	N/A	N/A	3000	50.515%
Adam	MLPMultistep5	0.01	3.78	N/A	N/A	3000	66.360%
Adam	MLPMultistep5	0.1	5.114	N/A	N/A	3000	72.601%

Table (4) Adam Performance Profiler

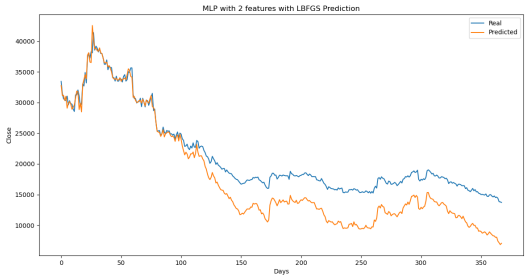
Out of the four algorithms, Adam is the fastest on all models. 4 and 3 also shares the pattern that as the learning rate increases (in any model), the number of parameters tend to 0 increases. Figure (26)d shows that the learning rate of 0.1 makes Adam perform very poorly on MLP and quite poor on MLPLarge whose number of zero parameters are 74.646% and 63.113% respectively, the highest percentage of all four algorithms. This result shows that smaller learning rate benefits Adam more. Smaller learning rates of 0.01, 0.001, 0.0001 result in good parameter estimations for MLP and MLPLarge as shown on the test set in figures (26) and (27). Just as the other 3 algorithms, Adam also perform poorly on the family of MLPMultistep models as shown in figures (28) - (31). The figures in section 6.6.4 show that like SGD, Adam also produces some spikes in the loss function. But unlike SGD, Adam's loss graph over the iterations are smoother and the spikes such as figure 52b,c are less perturbed compared to SGD's spikes such as ones in figure 49e,f. Therefore, Adam is a fast stochastic algorithm that is quite robust, especially compared to SGD.

6.5. **Prediction Figures**

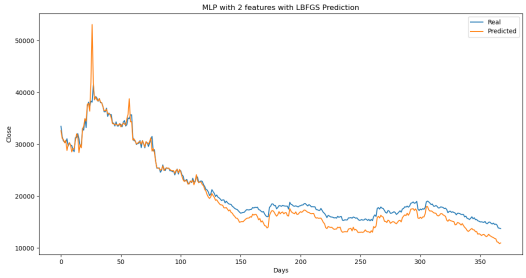
6.5.1. **LBFGS**



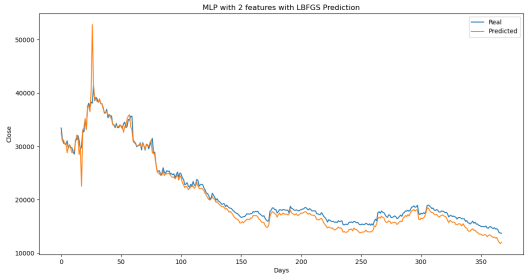
(a) MLP using LBFGS with learning rate 0.0001



(b) MLP using LBFGS with learning rate 0.001

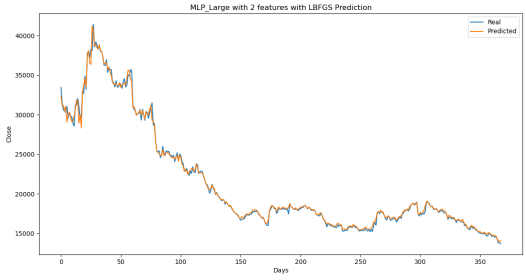


(c) MLP using LBFGS with learning rate 0.01

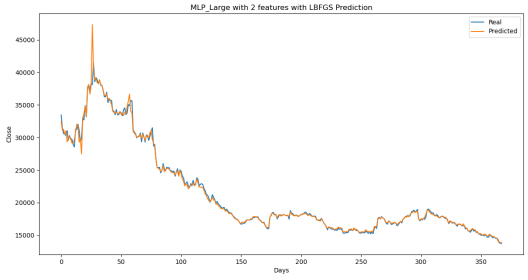


(d) MLP using LBFGS with learning rate 0.1

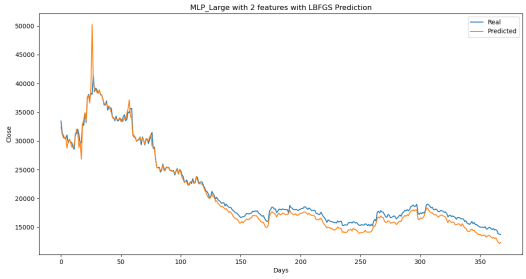
Figure (8) Prediction graphs using model MLP and optimizer LBFGS



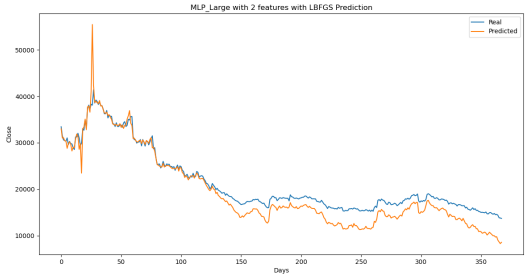
(a) MLP using LBFGS with learning rate 0.0001



(b) MLP using LBFGS with learning rate 0.001

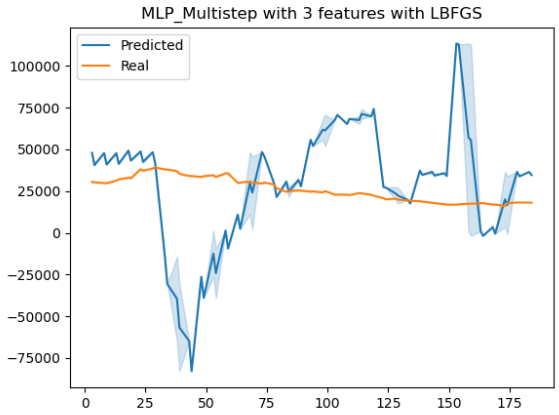


(c) MLP using LBFGS with learning rate 0.01

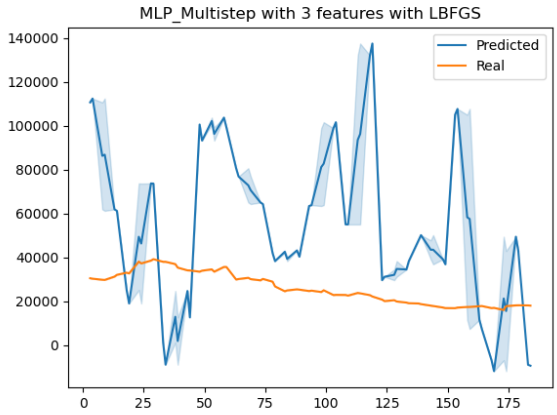


(d) MLP using LBFGS with learning rate 0.1

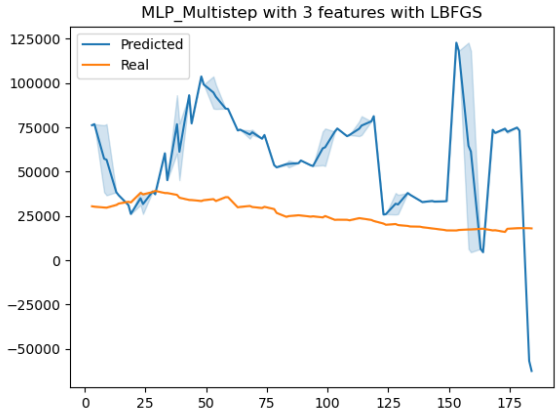
Figure (9) Prediction graphs using model MLPLarge and optimizer LBFGS



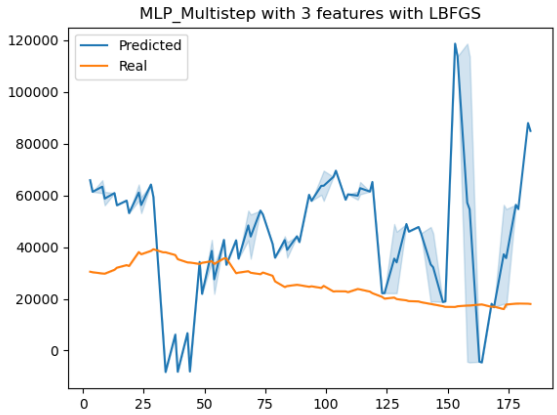
(a) MLPMultistep with 3 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 3 steps using LBFGS with learning rate 0.001

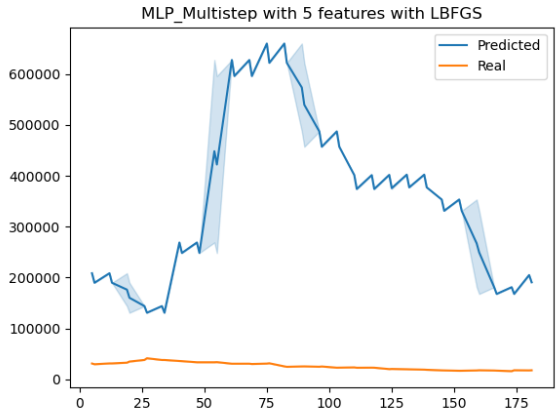


(c) MLPMultistep with 3 steps using LBFGS with learning rate 0.01

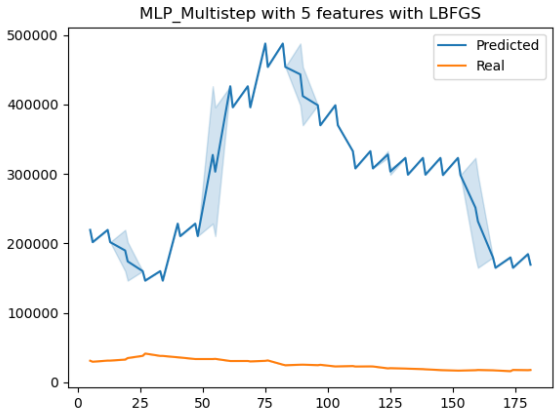


(d) MLPMultistep with 3 steps using LBFGS with learning rate 0.1

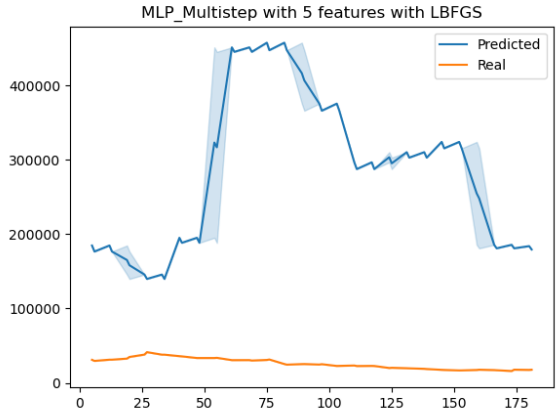
Figure (10) Prediction graphs using model MLPMultistep with 3 steps and optimizer LBFGS



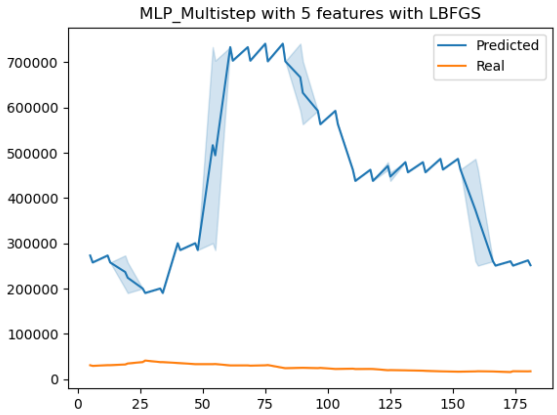
(a) MLPMultistep with 5 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 3 steps using LBFGS with learning rate 0.001

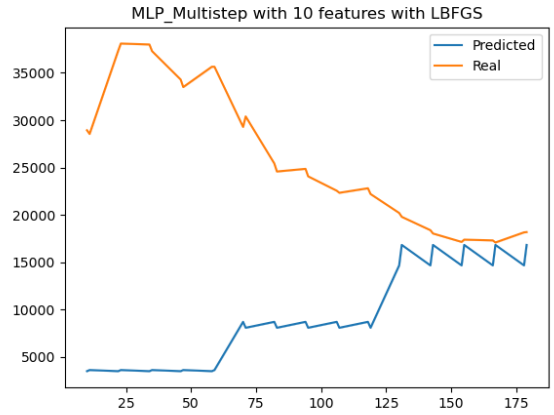


(c) MLPMultistep with 5 steps using LBFGS with learning rate 0.01

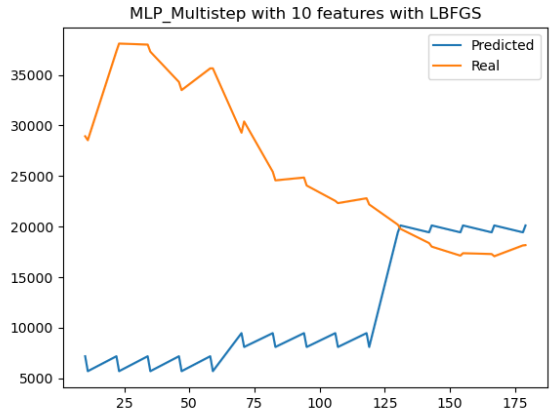


(d) MLPMultistep with 5 steps using LBFGS with learning rate 0.1

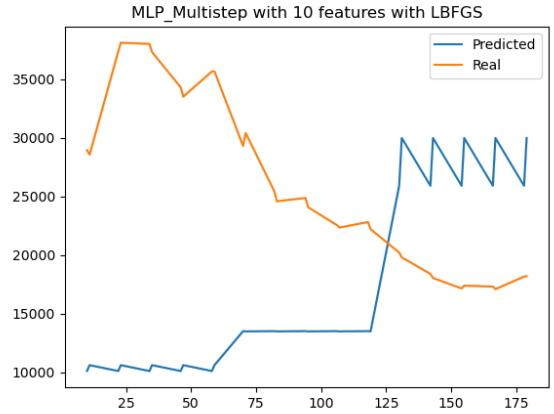
Figure (11) Prediction graphs using model MLPMultistep with 5 steps and optimizer LBFGS



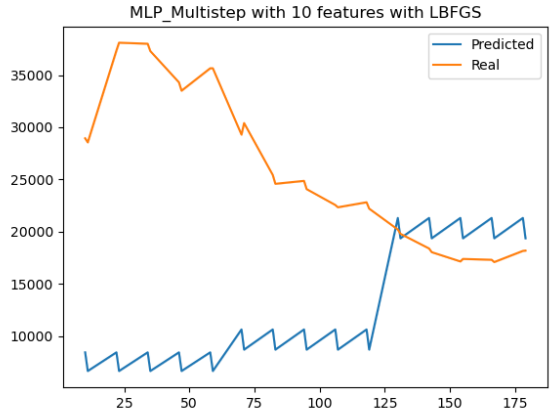
(a) MLPMultistep with 10 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 10 steps using LBFGS with learning rate 0.001

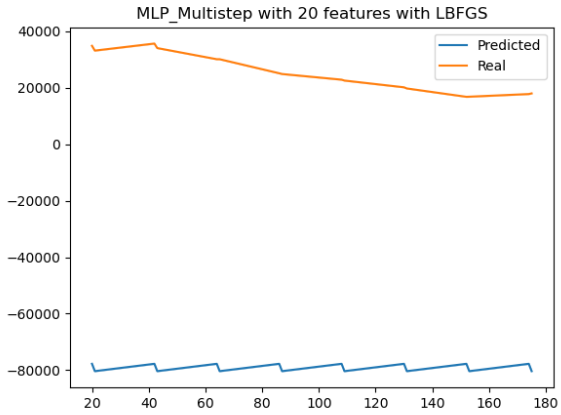


(c) MLPMultistep with 10 steps using LBFGS with learning rate 0.01

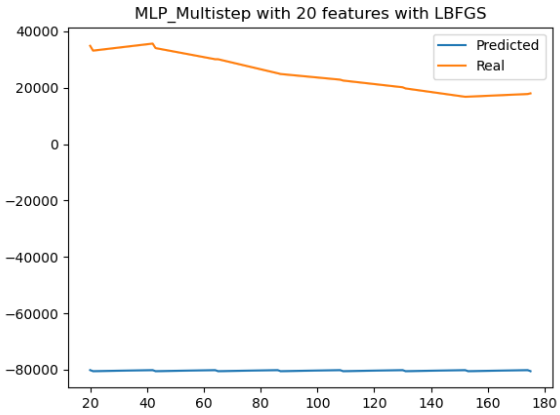


(d) MLPMultistep with 10 steps using LBFGS with learning rate 0.1

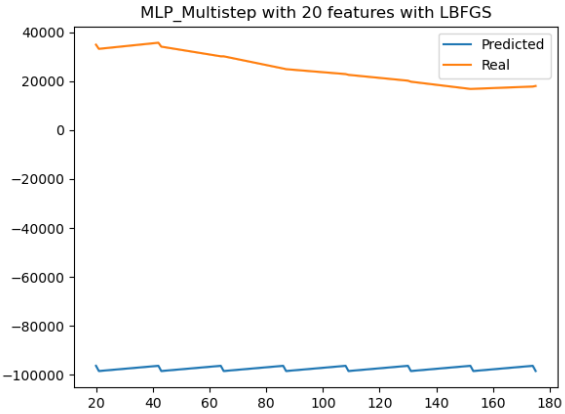
Figure (12) Prediction graphs using model MLPMultistep with 10 steps and optimizer LBFGS



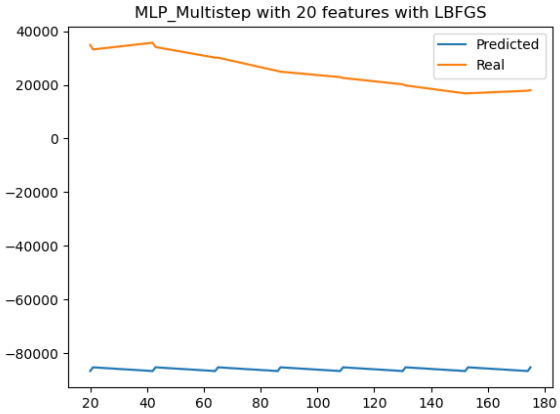
(a) MLPMultistep with 20 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 20 steps using LBFGS with learning rate 0.001



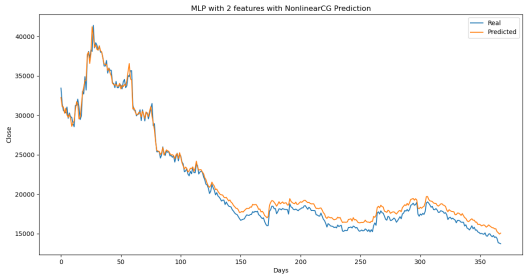
(c) MLPMultistep with 20 steps using LBFGS with learning rate 0.01



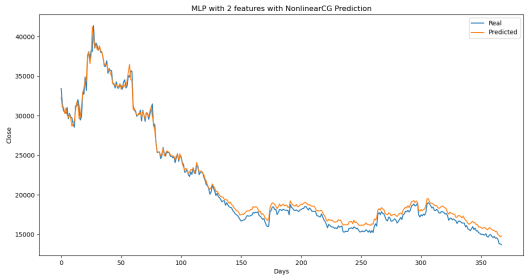
(d) MLPMultistep with 20 steps using LBFGS with learning rate 0.1

Figure (13) Prediction graphs using model MLPMultistep with 20 steps and optimizer LBFGS

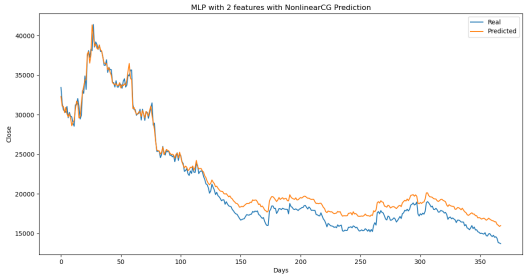
6.5.2. **NonlinearCG**



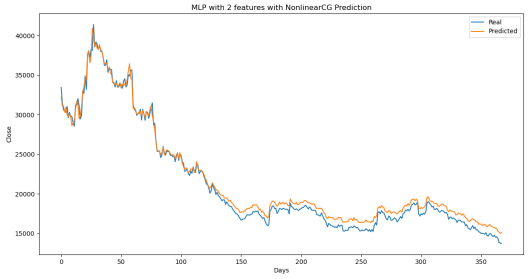
(a) MLP using NonlinearCG with learning rate 0.0001 and β^{HS}



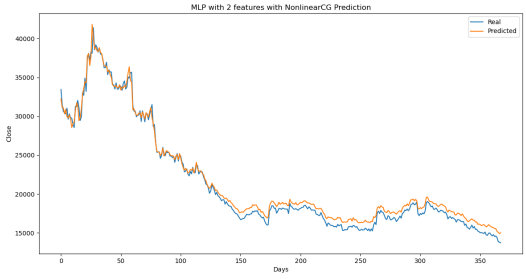
(b) MLP using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



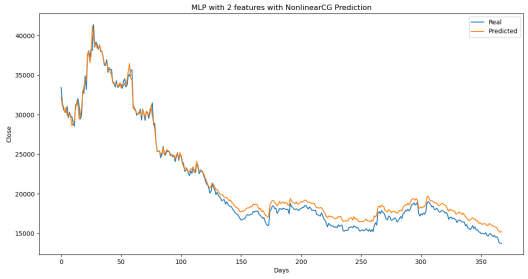
(c) MLP using NonlinearCG with learning rate 0.001 and β^{HS}



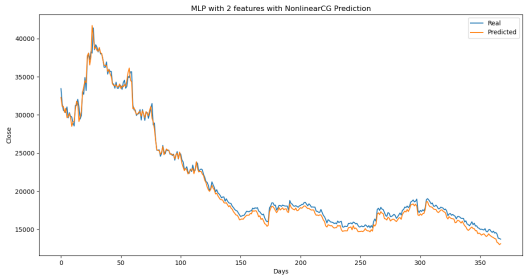
(d) MLP using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



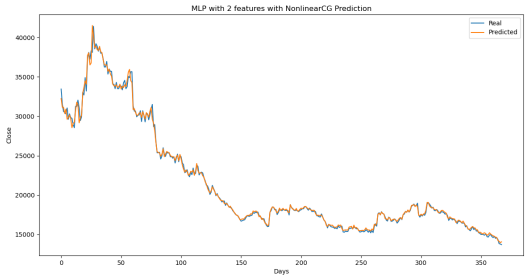
(e) MLP using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLP using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

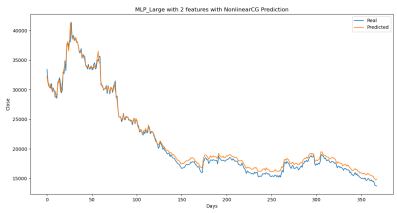


(g) MLP using NonlinearCG with learning rate 0.1 and β^{HS}

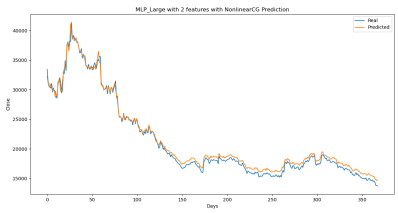


(h) MLP using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

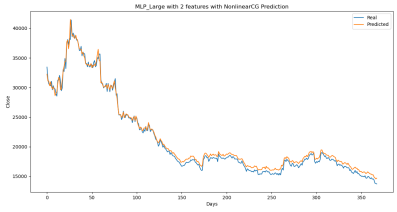
Figure (14) Prediction graphs using model MLP and optimizer NonlinearCG



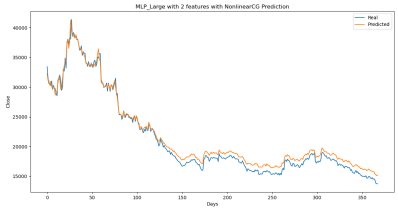
(a) MLPLarge using NonlinearCG with learning rate 0.0001 and β^{HS}



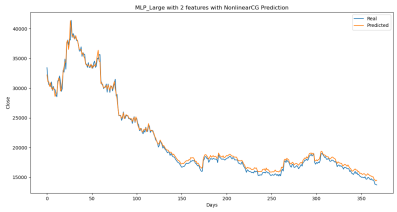
(b) MLPLarge using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



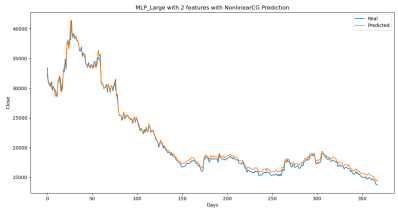
(c) MLPLarge using NonlinearCG with learning rate 0.001 and β^{HS}



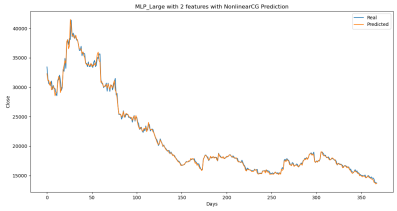
(d) MLPLarge using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



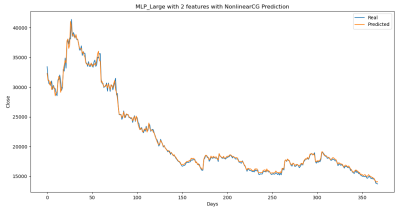
(e) MLPLarge using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPLarge using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

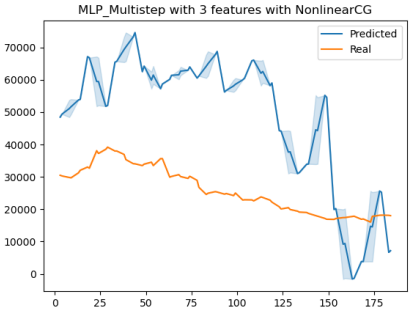


(g) MLPLarge using NonlinearCG with learning rate 0.1 and β^{HS}

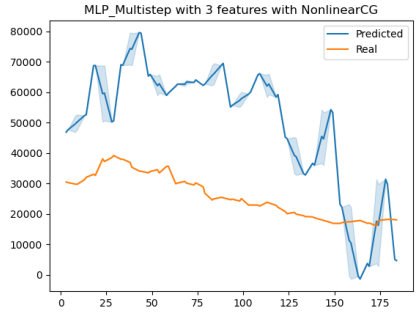


(h) MLPLarge using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

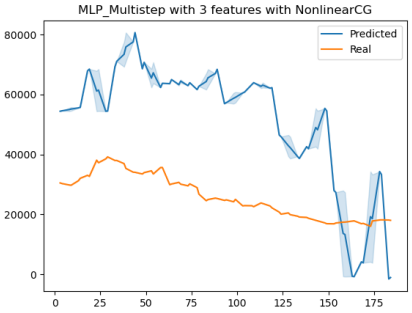
Figure (15) Prediction graphs using model MLPLarge and optimizer NonlinearCG



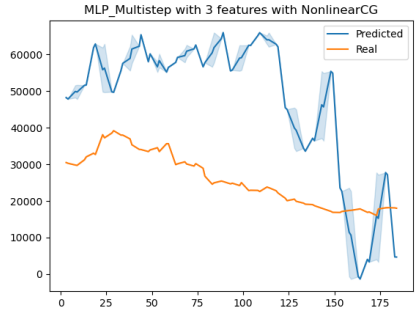
(a) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



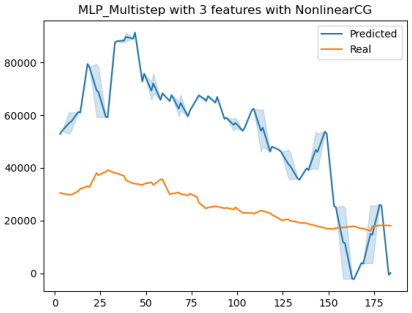
(b) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



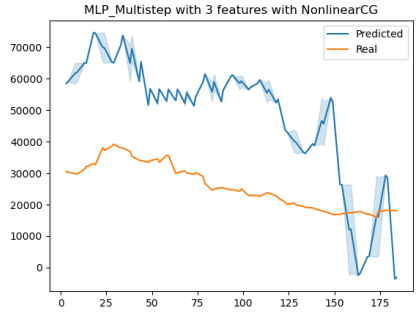
(c) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.001 and β^{HS}



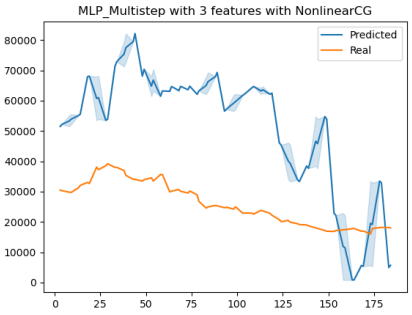
(d) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



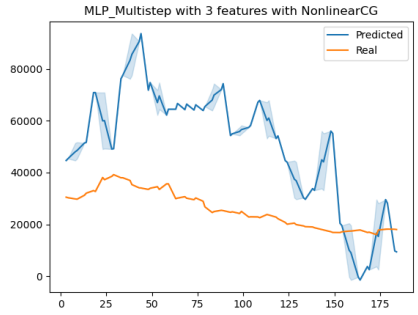
(e) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

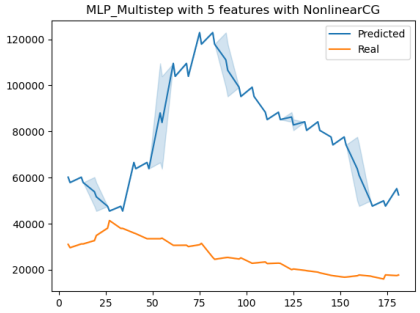


(g) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.1 and β^{HS}

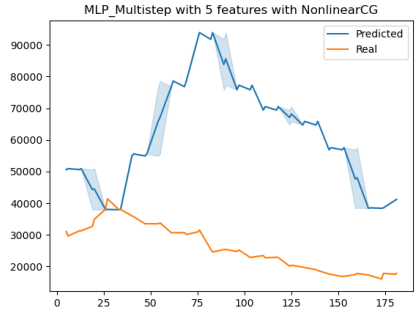


(h) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

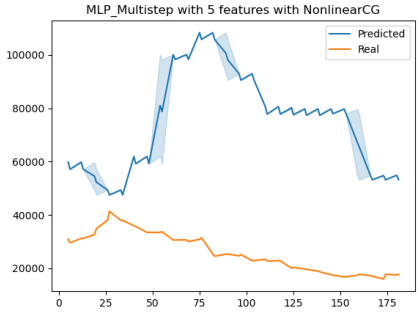
Figure (16) Prediction graphs using model MLPMultistep with 3 steps and optimizer NonlinearCG



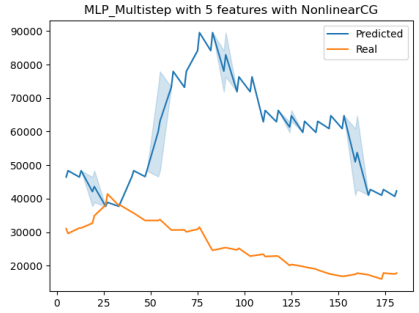
(a) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



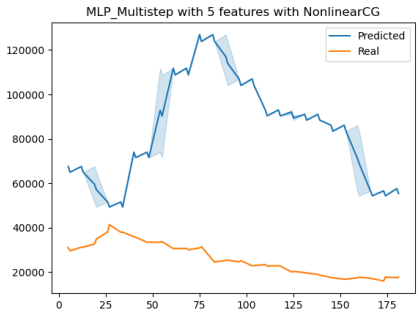
(b) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



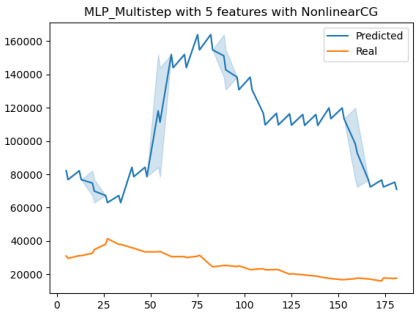
(c) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.001 and β^{HS}



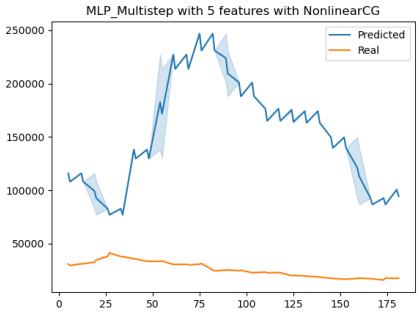
(d) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



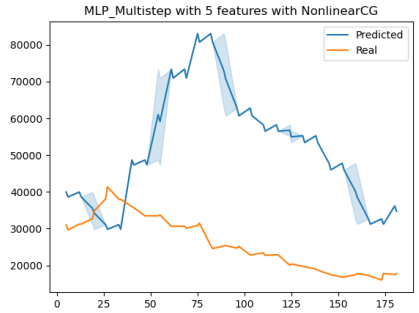
(e) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

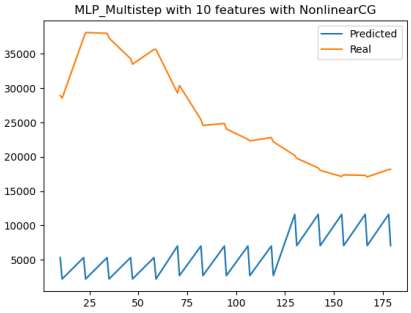


(g) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.1 and β^{HS}

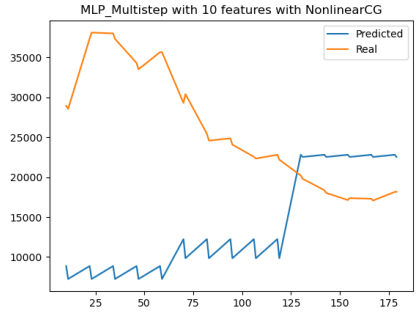


(h) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

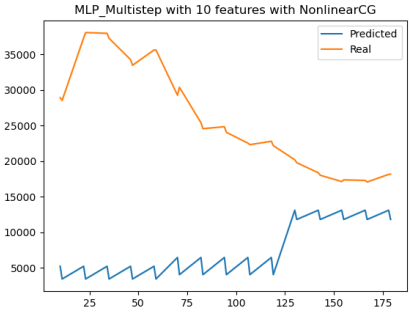
Figure (17) Prediction graphs using model MLPMultistep with 5 steps and optimizer NonlinearCG



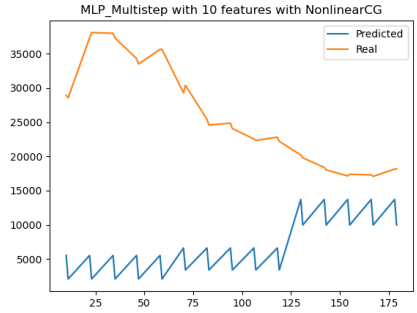
(a) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



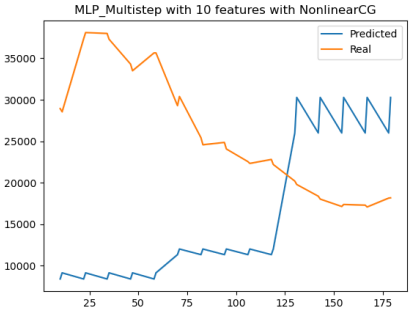
(b) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



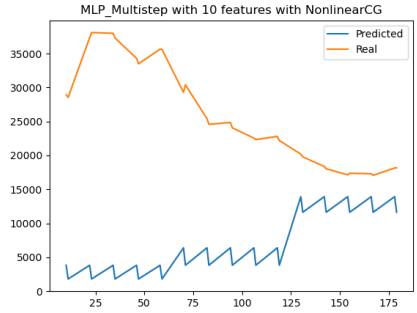
(c) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.001 and β^{HS}



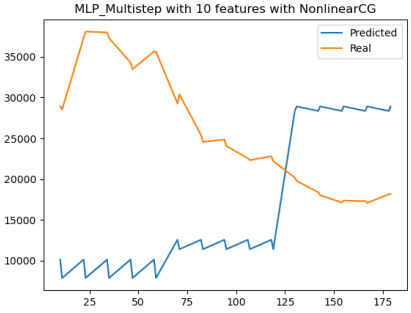
(d) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



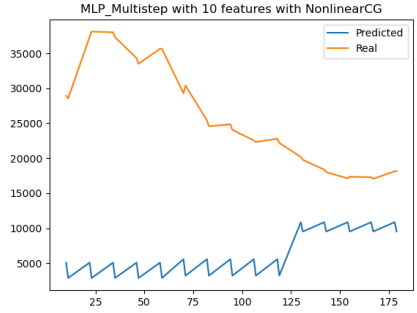
(e) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

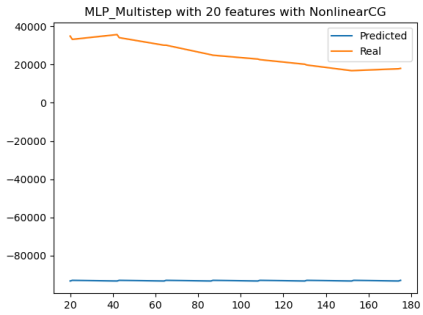


(g) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.1 and β^{HS}

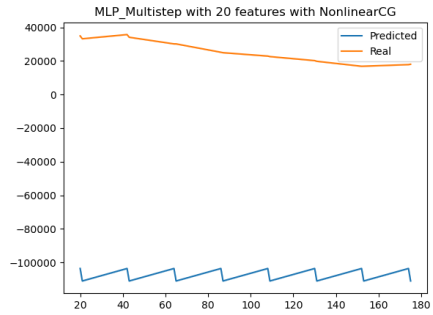


(h) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

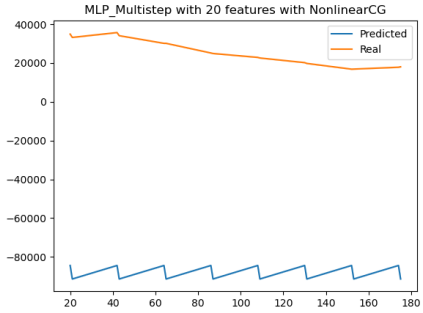
Figure (18) Prediction graphs using model MLPMultistep with 10 steps and optimizer NonlinearCG



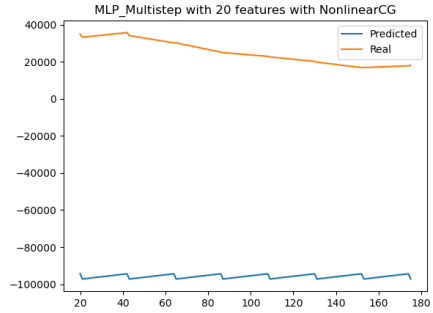
(a) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



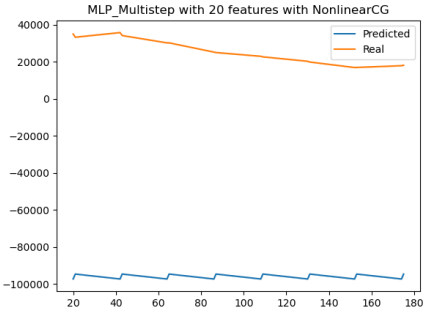
(b) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



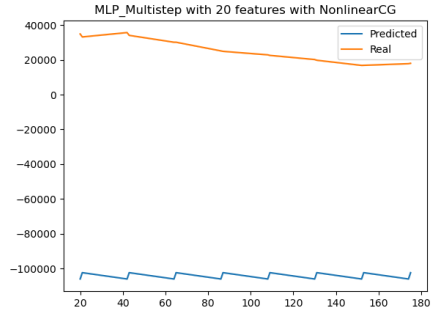
(c) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.001 and β^{HS}



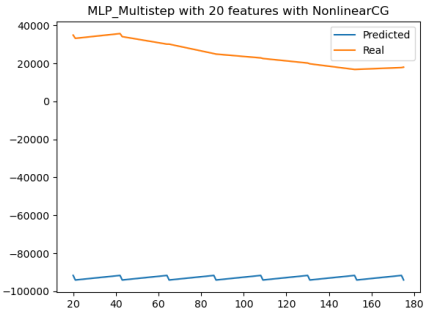
(d) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



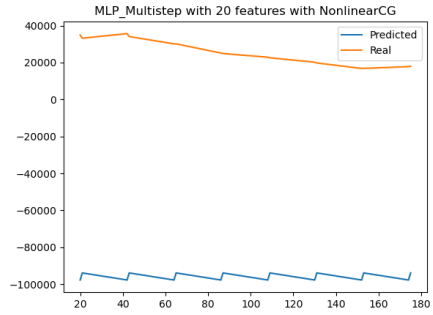
(e) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$



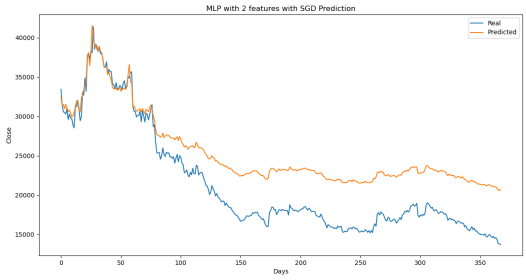
(g) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.1 and β^{HS}



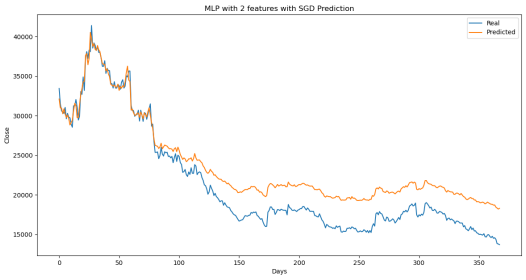
(h) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

Figure (19) Prediction graphs using model MLPMultistep with 10 steps and optimizer NonlinearCG

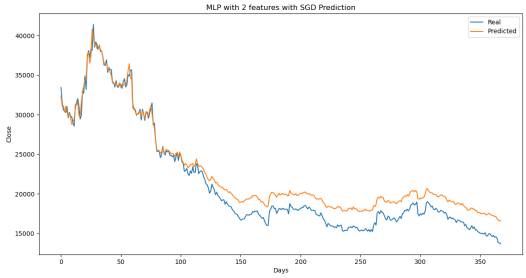
6.5.3. **SGD**



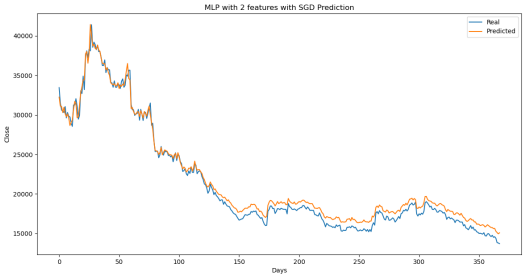
(a) MLP using SGD with learning rate 0.0001



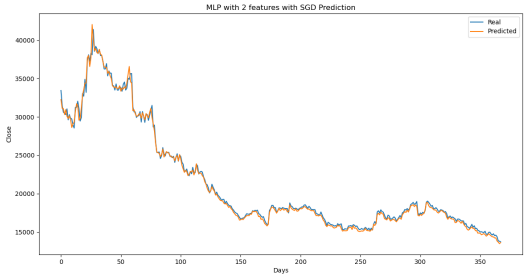
(b) MLP using SGD with learning rate 0.001



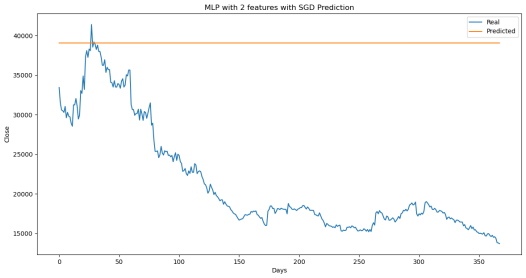
(c) MLP using SGD with learning rate 0.01



(d) MLP using SGD with learning rate 0.2

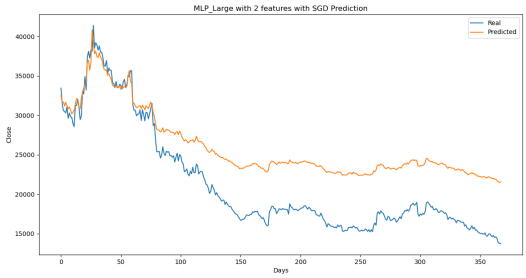


(e) MLP using SGD with learning rate 0.3

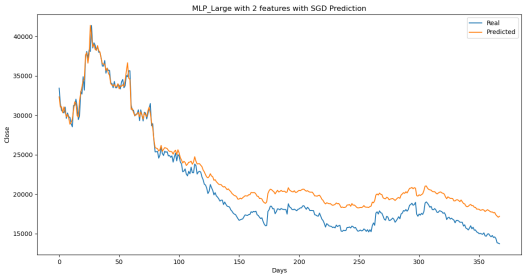


(f) MLP using SGD with learning rate 0.4

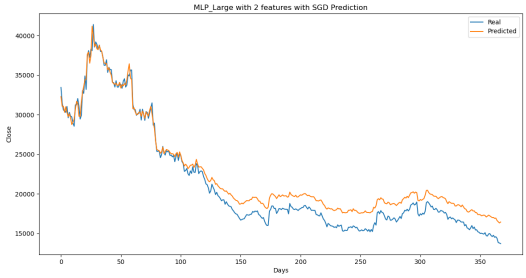
Figure (20) Prediction graphs using model MLP and optimizer SGD



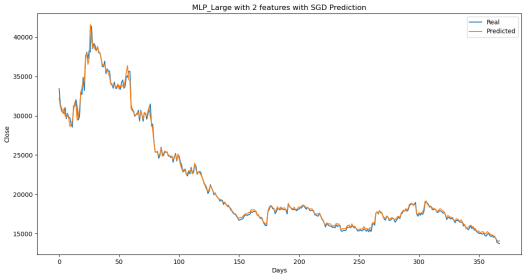
(a) MLPLarge using SGD with learning rate 0.0001



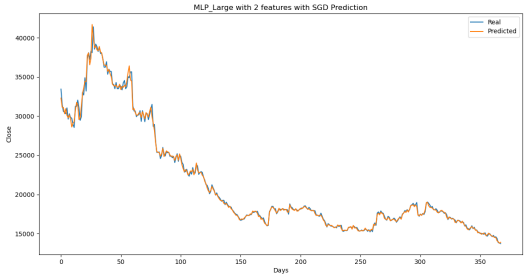
(b) MLPLarge using SGD with learning rate 0.001



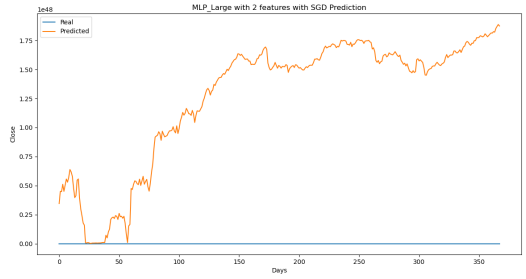
(c) MLPLarge using SGD with learning rate 0.01



(d) MLPLarge using SGD with learning rate 0.2

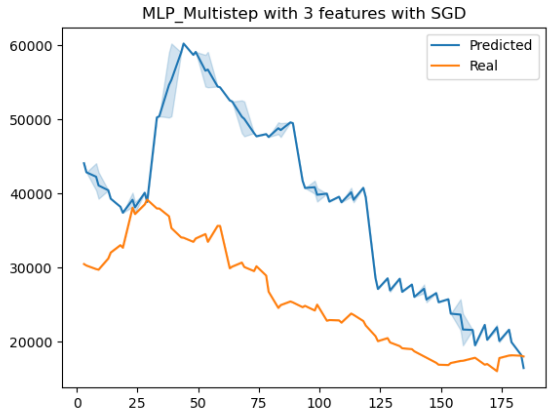


(e) MLPLarge using SGD with learning rate 0.3

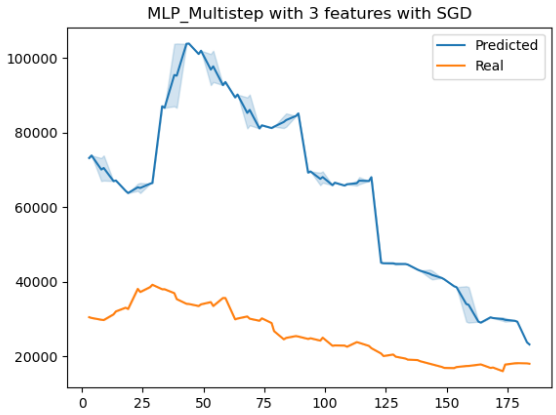


(f) MLPLarge using SGD with learning rate 0.4

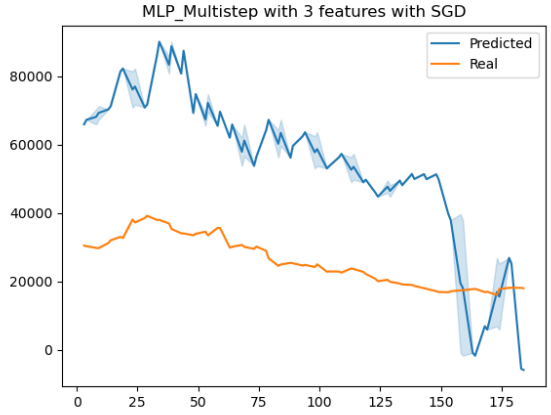
Figure (21) Prediction graphs using model MLPLarge and optimizer SGD



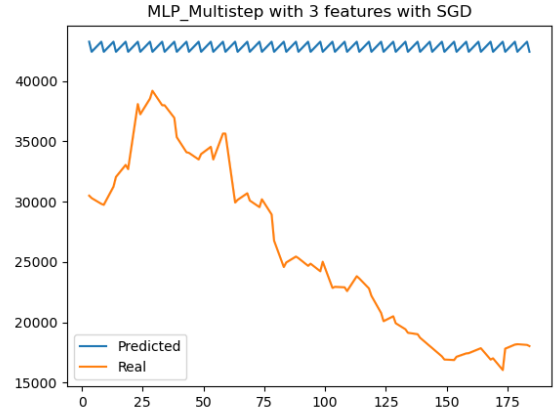
(a) MLPMultistep with 3 steps using SGD with learning rate 0.0001



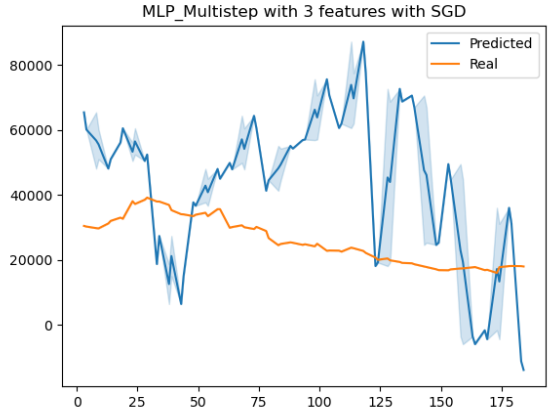
(b) MLPMultistep with 3 steps using SGD with learning rate 0.001



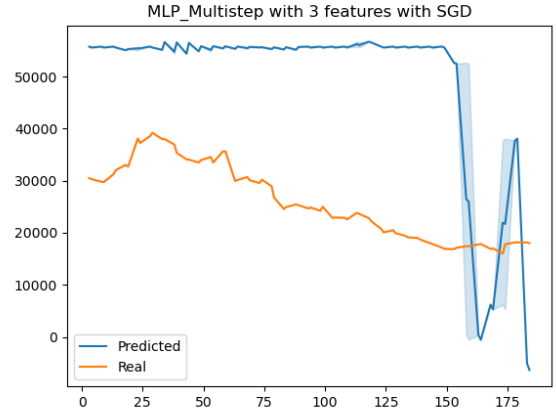
(c) MLPMultistep with 3 steps using SGD with learning rate 0.01



(d) MLPMultistep with 3 steps using SGD with learning rate 0.2

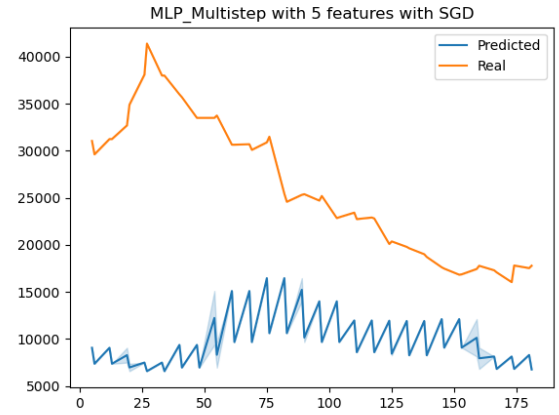


(e) MLPMultistep with 3 steps using SGD with learning rate 0.3

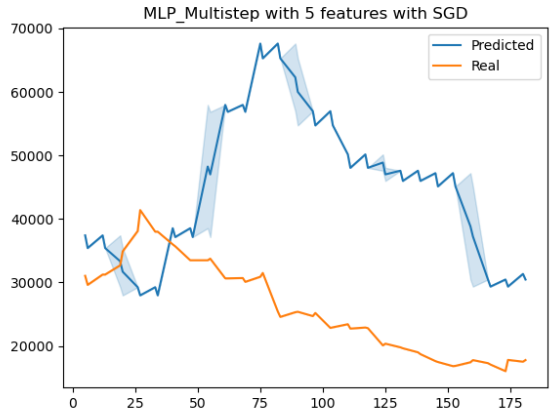


(f) MLPMultistep with 3 steps using SGD with learning rate 0.3

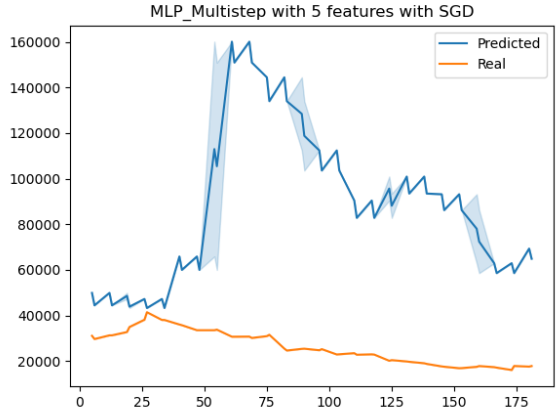
Figure (22) Prediction graphs using model MLPMultistep and optimizer SGD



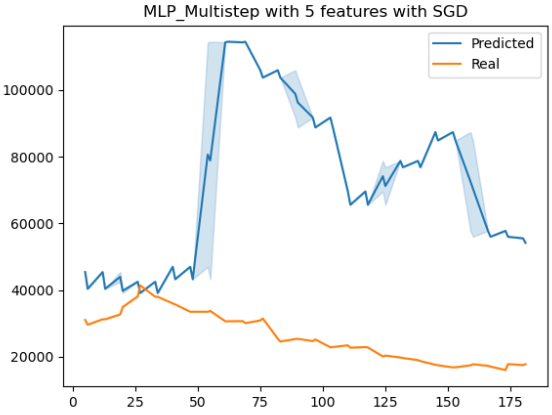
(a) MLPMultistep with 5 steps using SGD with learning rate 0.0001



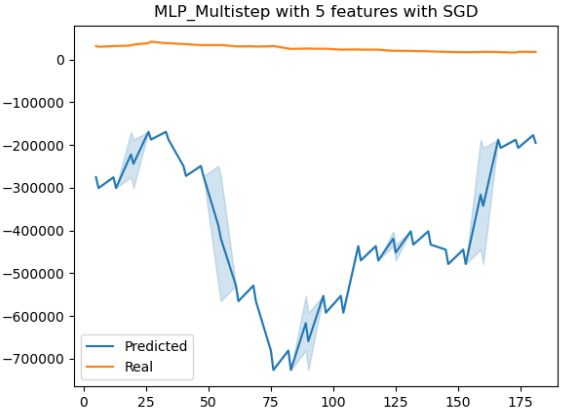
(b) MLPMultistep with 3 steps using SGD with learning rate 0.001



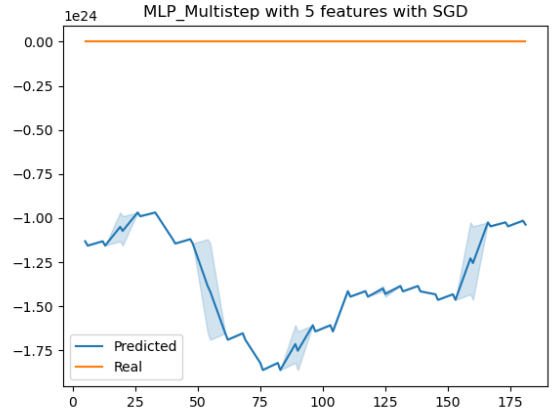
(c) MLPMultistep with 5 steps using SGD with learning rate 0.01



(d) MLPMultistep with 5 steps using SGD with learning rate 0.2

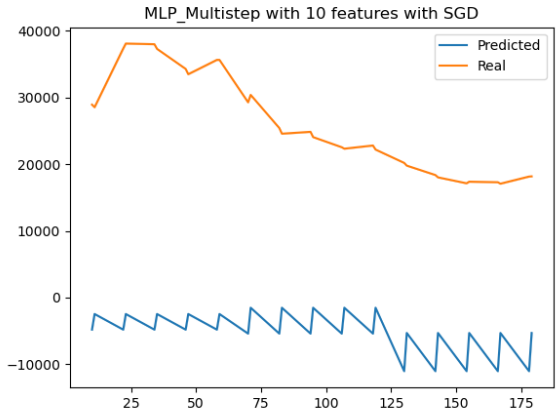


(e) MLPMultistep with 5 steps using SGD with learning rate 0.3

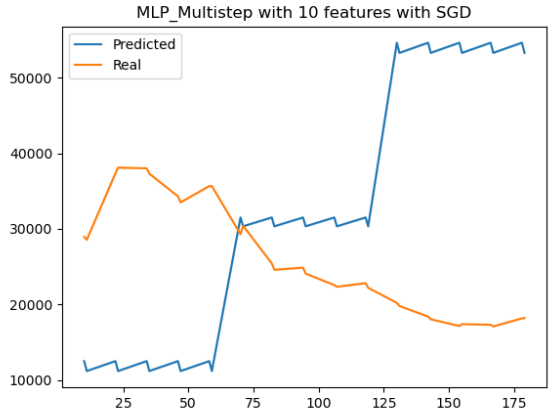


(f) MLPMultistep with 5 steps using SGD with learning rate 0.4

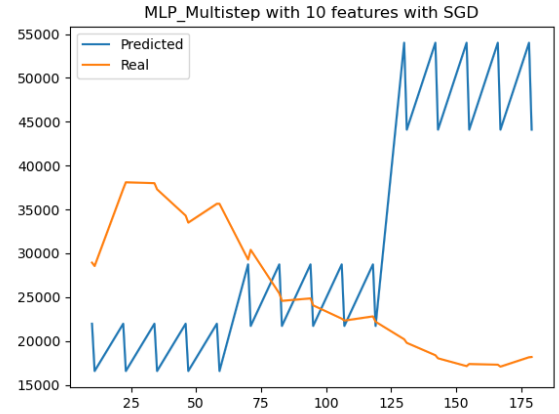
Figure (23) Prediction graphs using model MLPMultistep and optimizer SGD



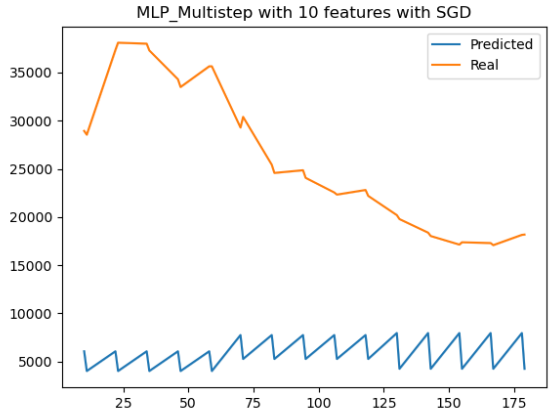
(a) MLPMultistep with 10 steps using SGD with learning rate 0.0001



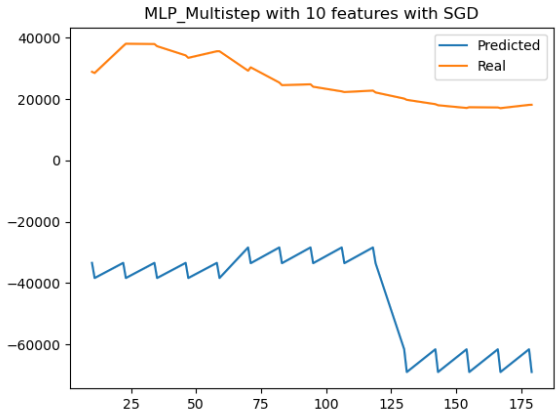
(b) MLPMultistep with 10 steps using SGD with learning rate 0.001



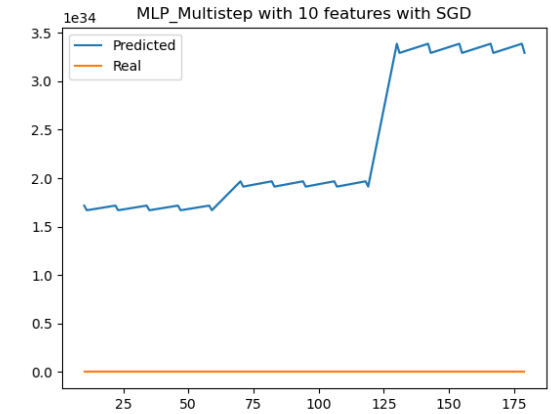
(c) MLPMultistep with 10 steps using SGD with learning rate 0.01



(d) MLPMultistep with 10 steps using SGD with learning rate 0.2

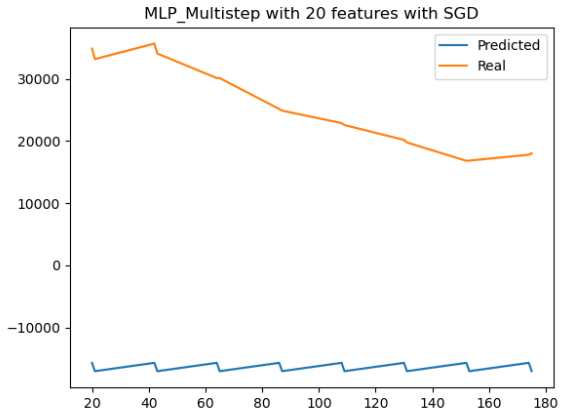


(e) MLPMultistep with 10 steps using SGD with learning rate 0.3

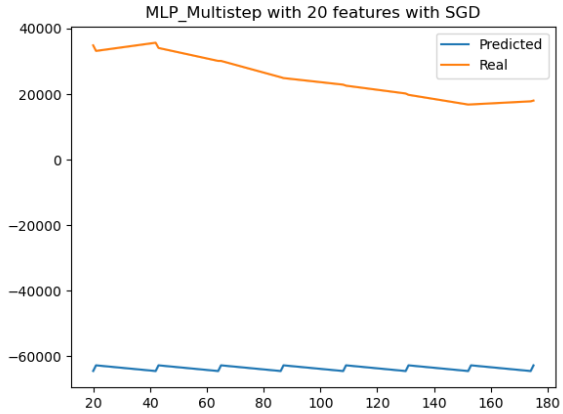


(f) MLPMultistep with 10 steps using SGD with learning rate 0.4

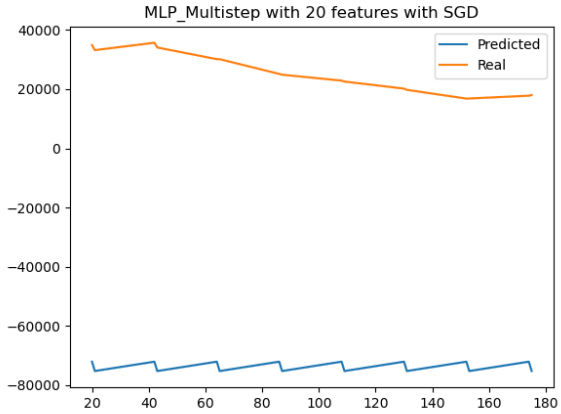
Figure (24) Prediction graphs using model MLPMultistep and optimizer SGD



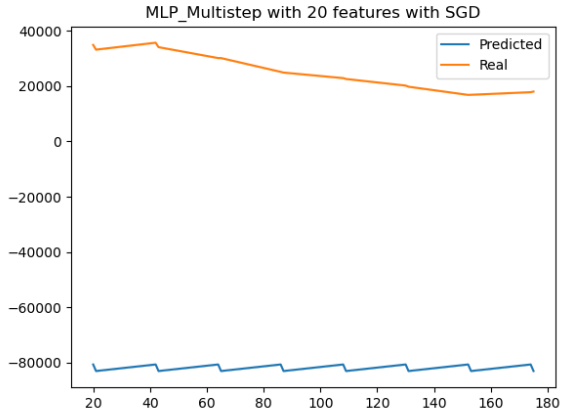
(a) MLPMultistep with 20 steps using SGD with learning rate 0.0001



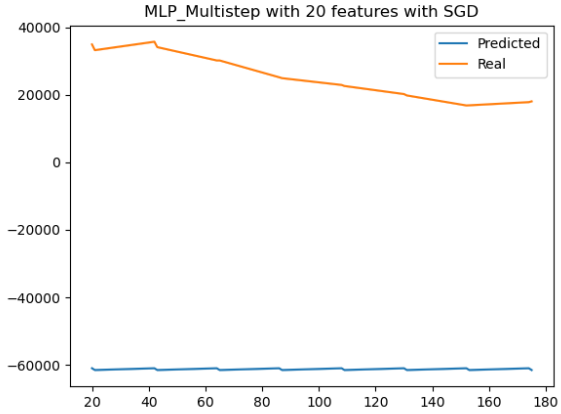
(b) MLPMultistep with 20 steps using SGD with learning rate 0.001



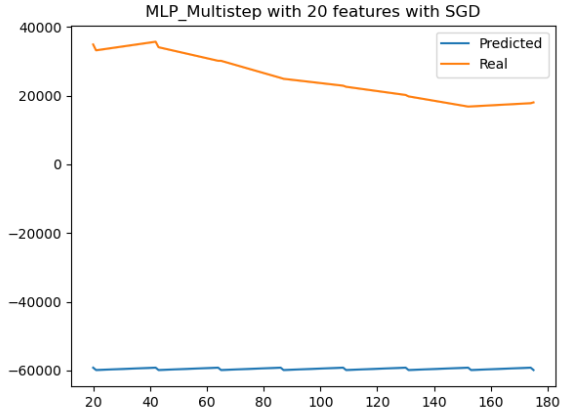
(c) MLPMultistep with 20 steps using SGD with learning rate 0.01



(d) MLPMultistep with 20 steps using SGD with learning rate 0.2



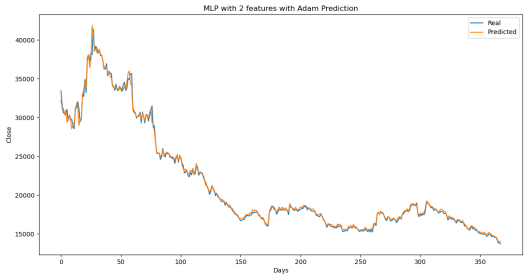
(e) MLPMultistep with 20 steps using SGD with learning rate 0.3



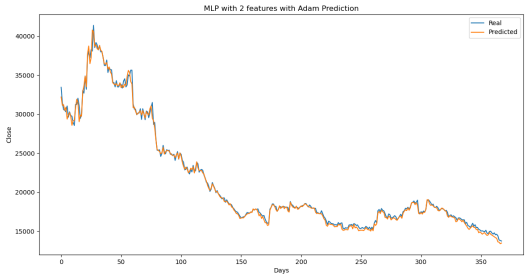
(f) MLPMultistep with 20 steps using SGD with learning rate 0.4

Figure (25) Prediction graphs using model MLPMultistep and optimizer SGD

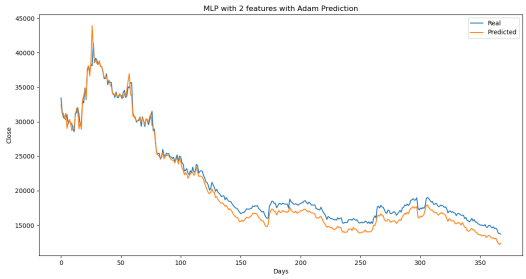
6.5.4. **Adam**



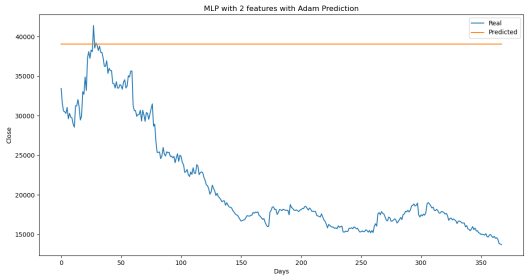
(a) MLP using Adam with learning rate 0.0001



(b) MLP using Adam with learning rate 0.001

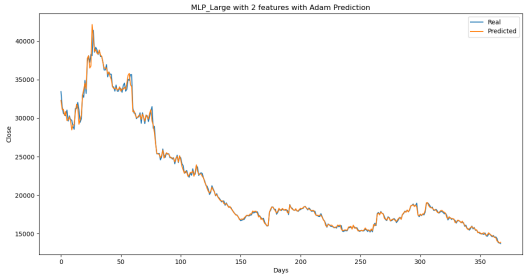


(c) MLP using Adam with learning rate 0.01

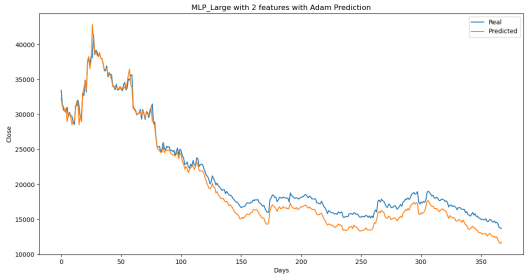


(d) MLP using Adam with learning rate 0.1

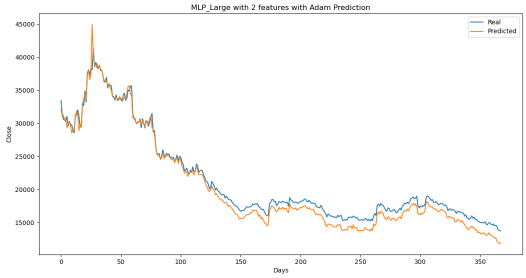
Figure (26) Prediction graphs using model MLP and optimizer Adam



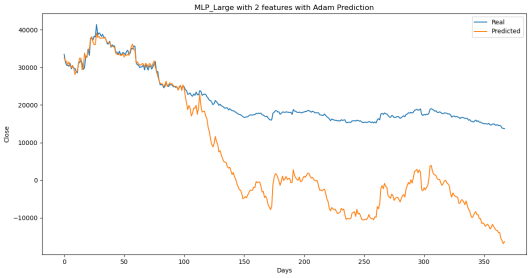
(a) MLP using Adam with learning rate 0.0001



(b) MLP using Adam with learning rate 0.001

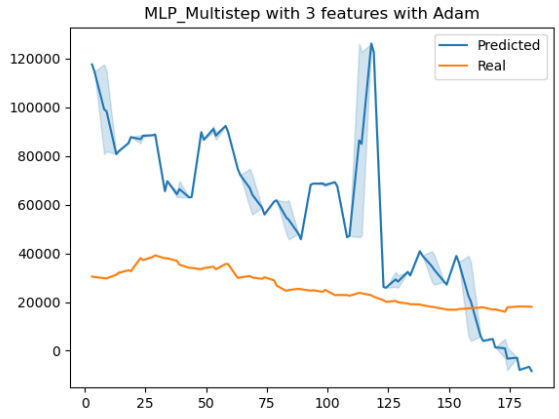


(c) MLP using Adam with learning rate 0.01

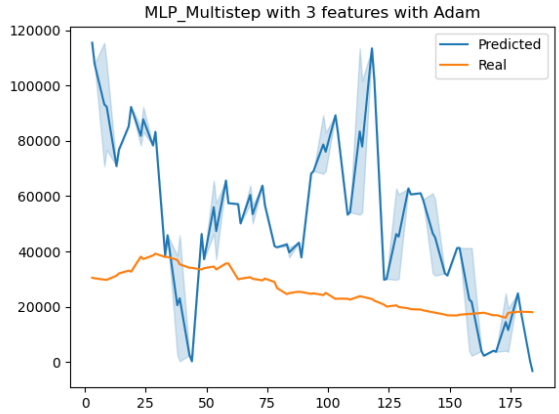


(d) MLP using Adam with learning rate 0.1

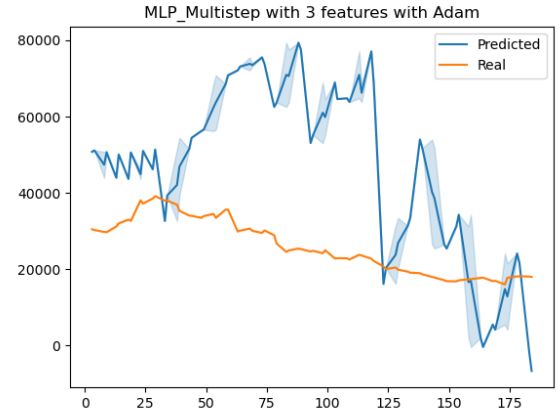
Figure (27) Prediction graphs using model MLPLarge and optimizer Adam



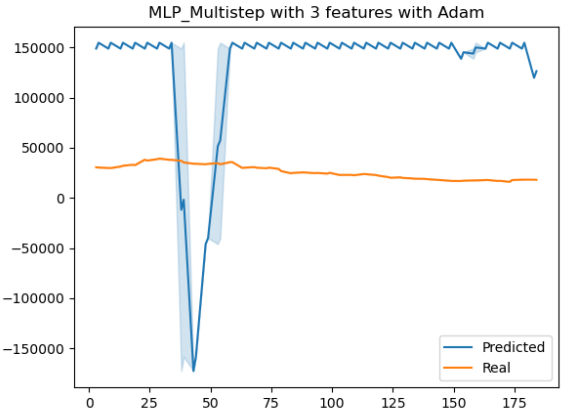
(a) MLPMultistep with 3 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 3 steps using Adam with learning rate 0.001

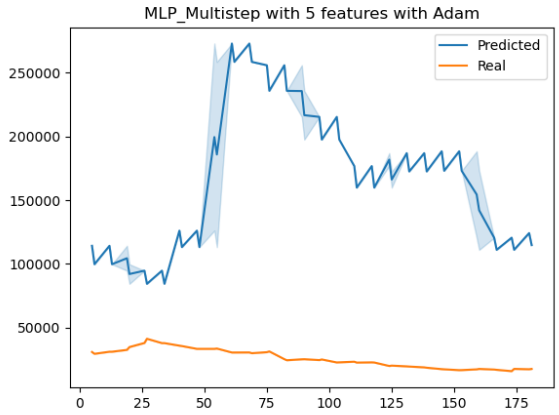


(c) MLPMultistep with 3 steps using Adam with learning rate 0.01

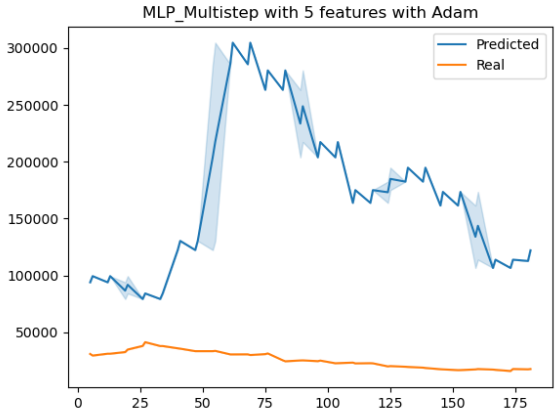


(d) MLPMultistep with 3 steps using Adam with learning rate 0.1

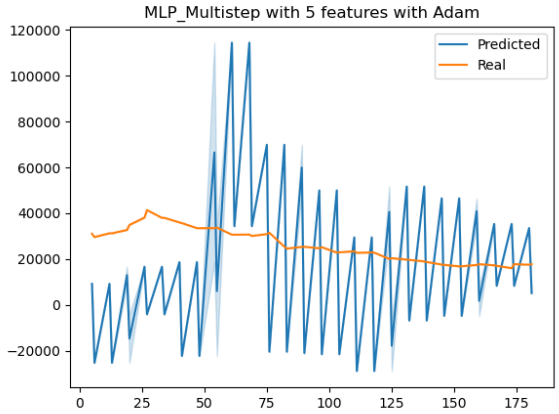
Figure (28) Prediction graphs using model MLPMultistep with 3 steps and optimizer Adam



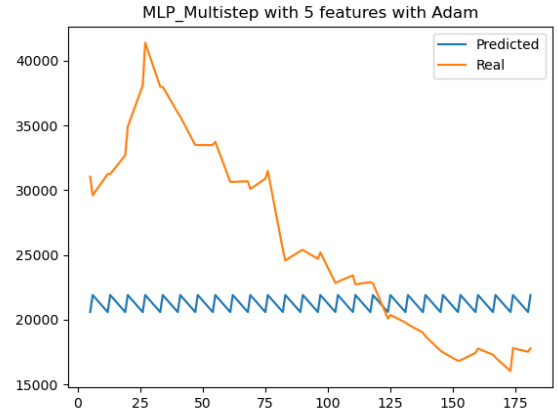
(a) MLPMultistep with 5 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 3 steps using Adam with learning rate 0.001

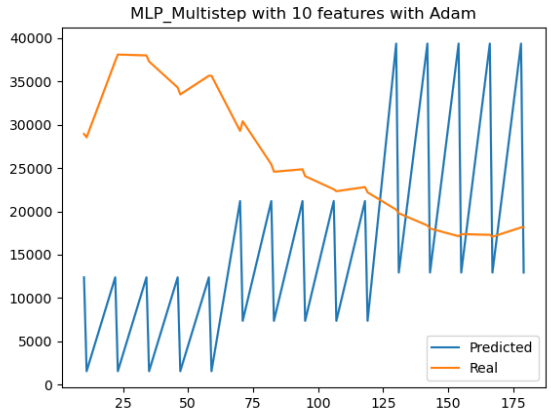


(c) MLPMultistep with 5 steps using Adam with learning rate 0.01

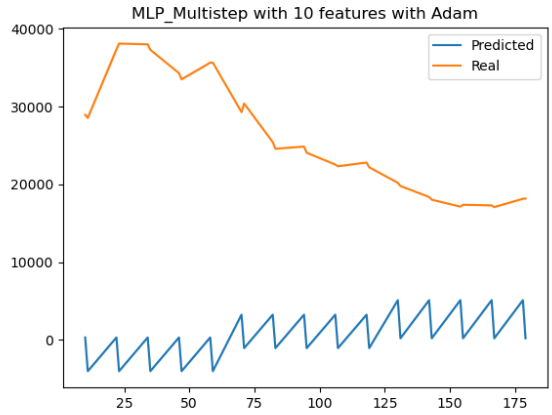


(d) MLPMultistep with 5 steps using Adam with learning rate 0.1

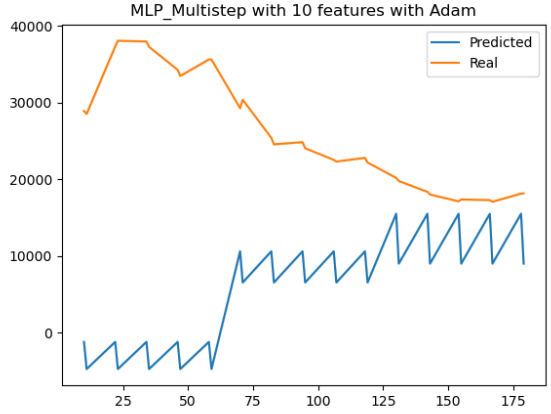
Figure (29) Prediction graphs using model MLPMultistep with 5 steps and optimizer Adam



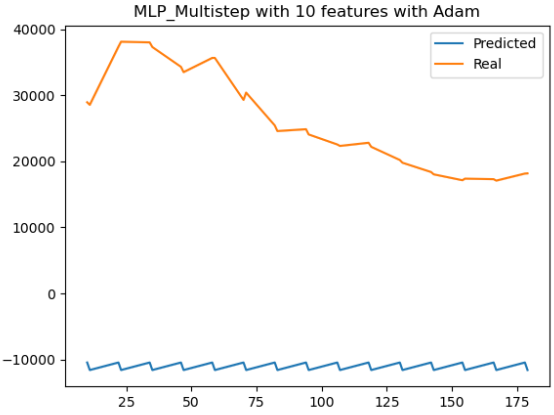
(a) MLPMultistep with 10 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 10 steps using Adam with learning rate 0.001

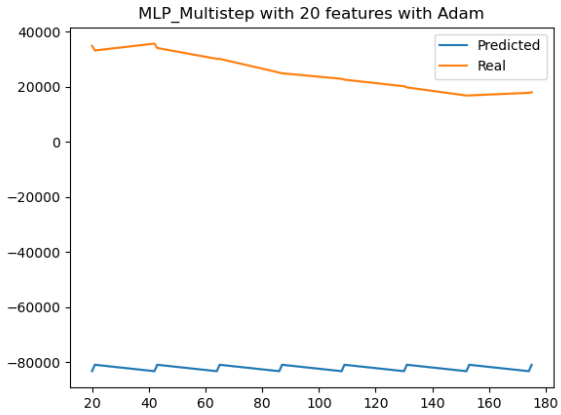


(c) MLPMultistep with 10 steps using Adam with learning rate 0.01

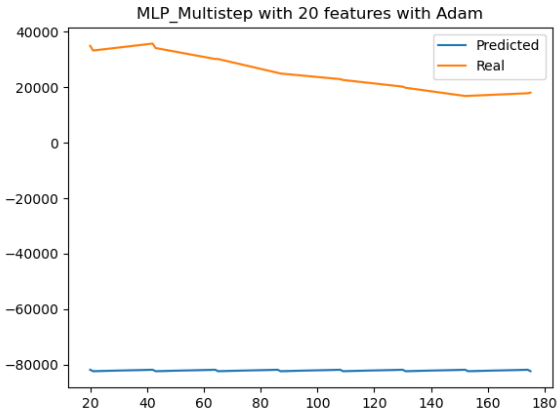


(d) MLPMultistep with 10 steps using Adam with learning rate 0.1

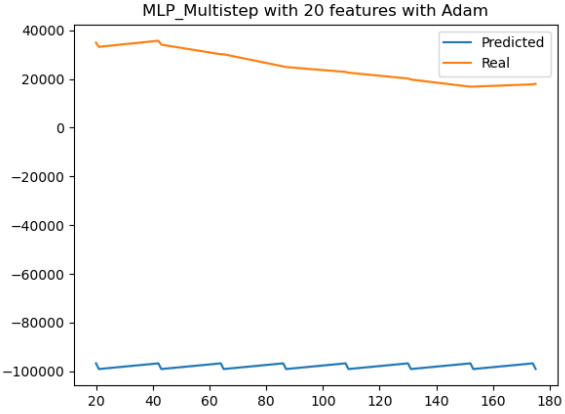
Figure (30) Prediction graphs using model MLPMultistep with 10 steps and optimizer Adam



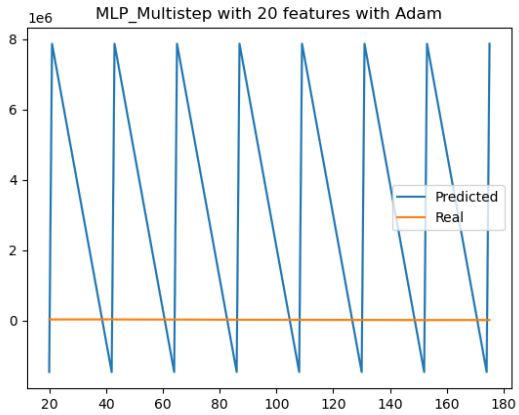
(a) MLPMultistep with 20 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 20 steps using Adam with learning rate 0.001



(c) MLPMultistep with 20 steps using Adam with learning rate 0.01

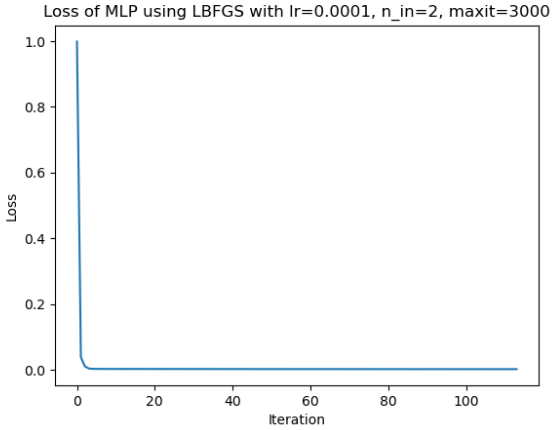


(d) MLPMultistep with 20 steps using Adam with learning rate 0.1

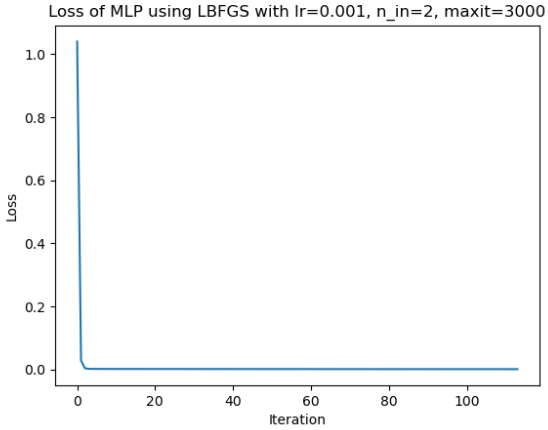
Figure (31) Prediction graphs using model MLPMultistep with 20 steps and optimizer Adam

6.6. Loss Figures

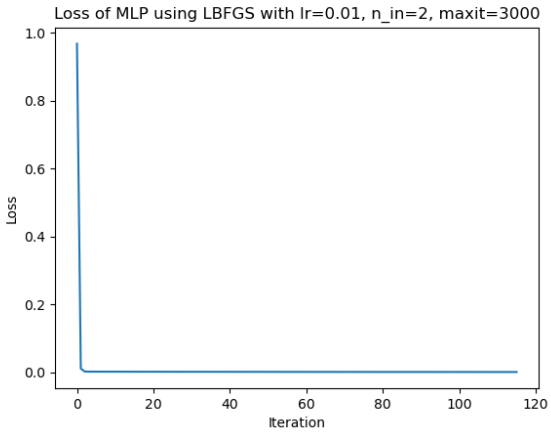
6.6.1. LBFGS



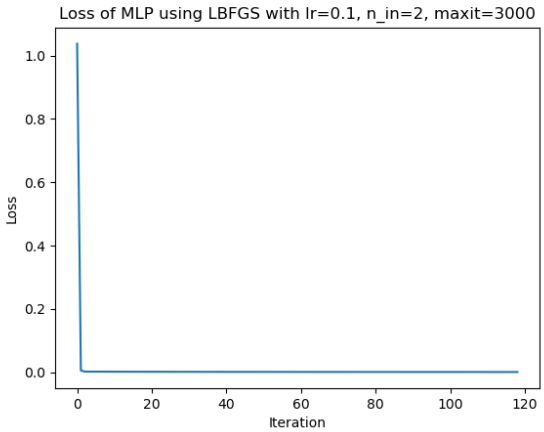
(a) MLP using LBFGS with learning rate 0.0001



(b) MLP using LBFGS with learning rate 0.001

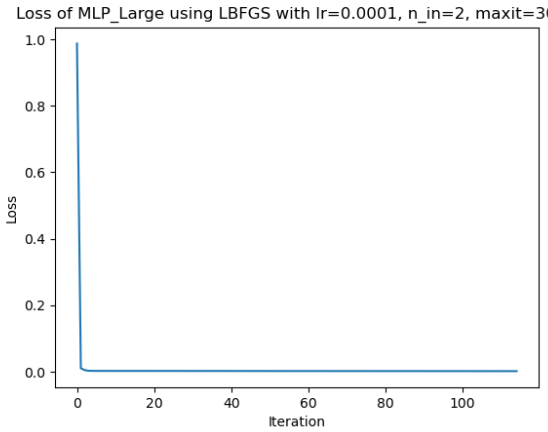


(c) MLP using LBFGS with learning rate 0.01

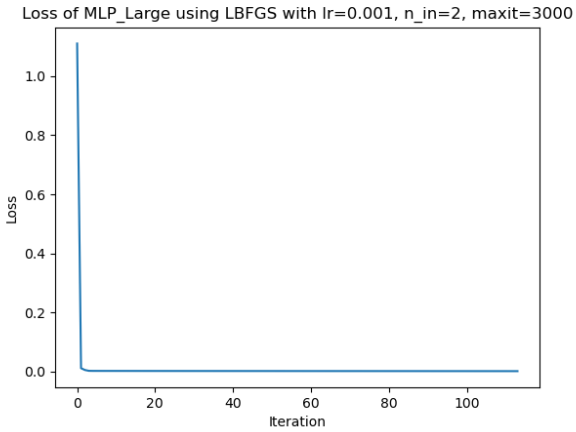


(d) MLP using LBFGS with learning rate 0.1

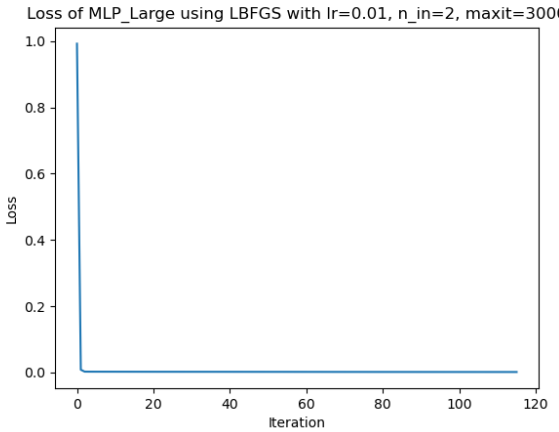
Figure (32) Prediction graphs using model MLP and optimizer LBFGS



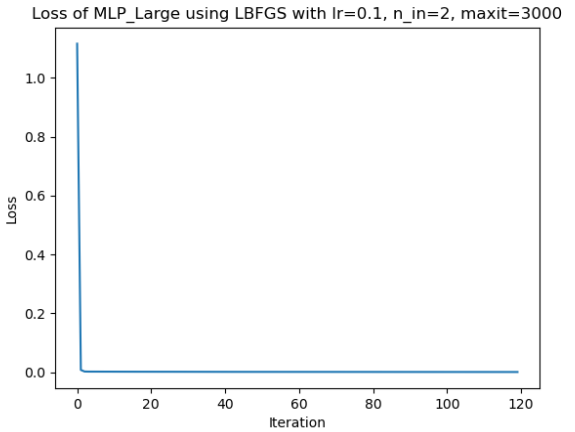
(a) MLP using LBFGS with learning rate 0.0001



(b) MLP using LBFGS with learning rate 0.001

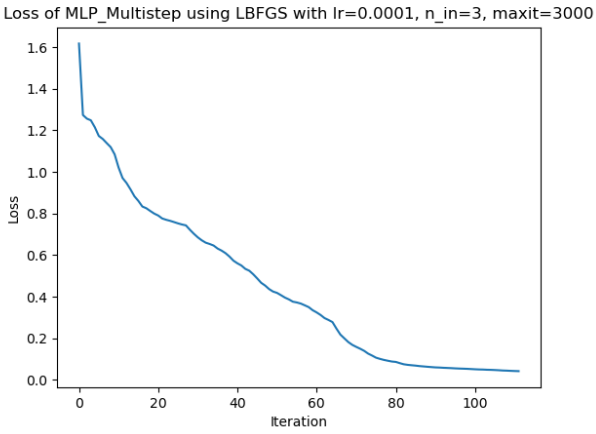


(c) MLP using LBFGS with learning rate 0.01

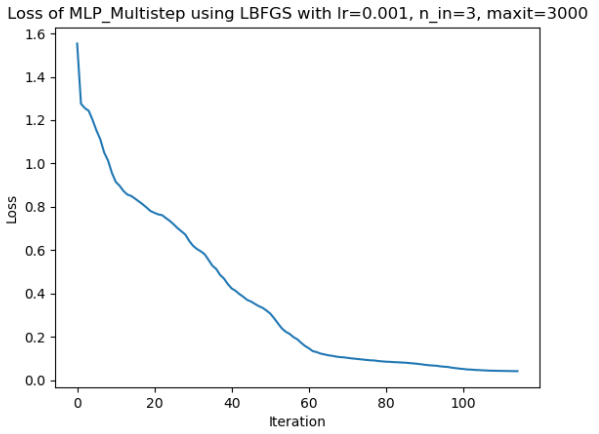


(d) MLP using LBFGS with learning rate 0.1

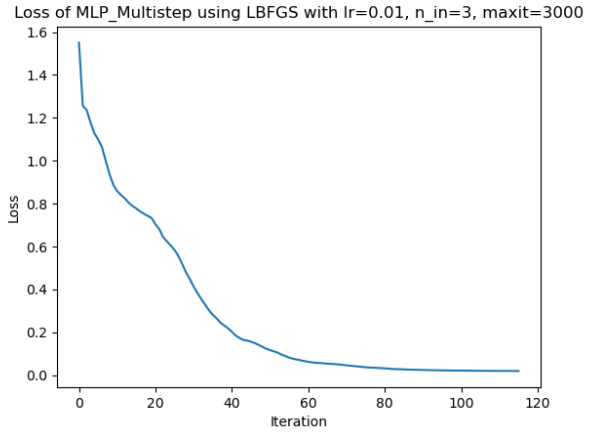
Figure (33) Prediction graphs using model MLPLarge and optimizer LBFGS



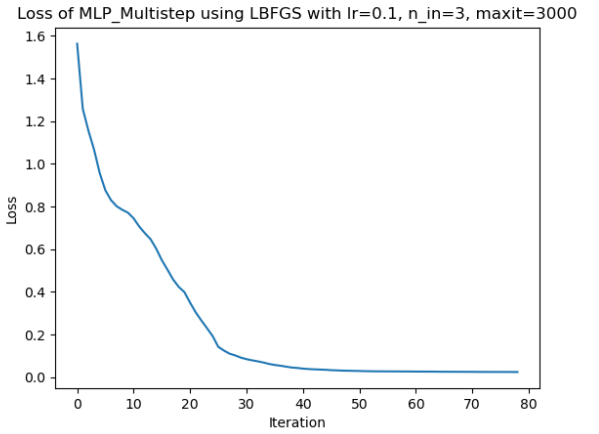
(a) MLPMultistep with 3 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 3 steps using LBFGS with learning rate 0.001

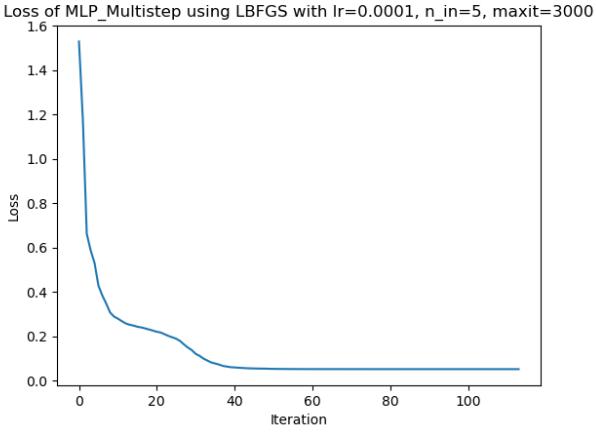


(c) MLPMultistep with 3 steps using LBFGS with learning rate 0.01

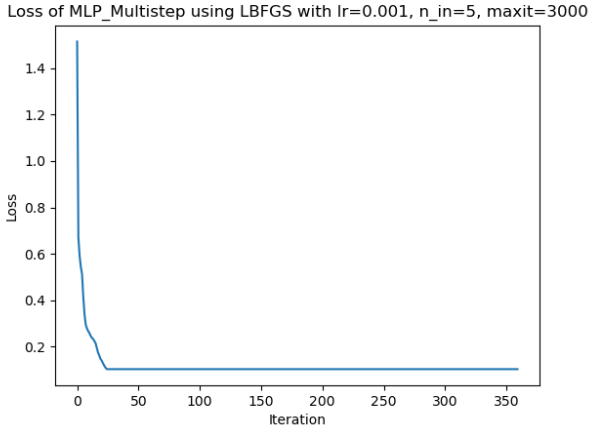


(d) MLPMultistep with 3 steps using LBFGS with learning rate 0.1

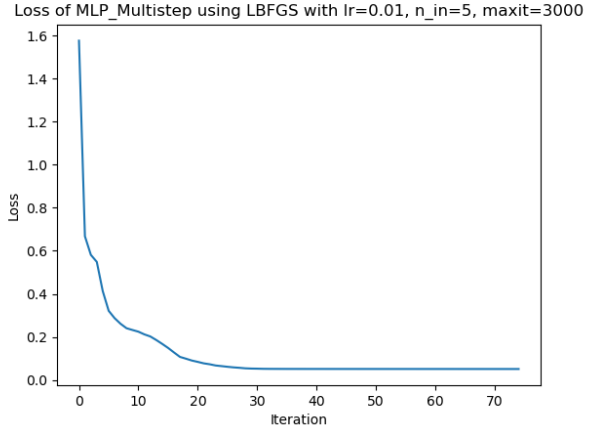
Figure (34) Prediction graphs using model MLPMultistep with 3 steps and optimizer LBFGS



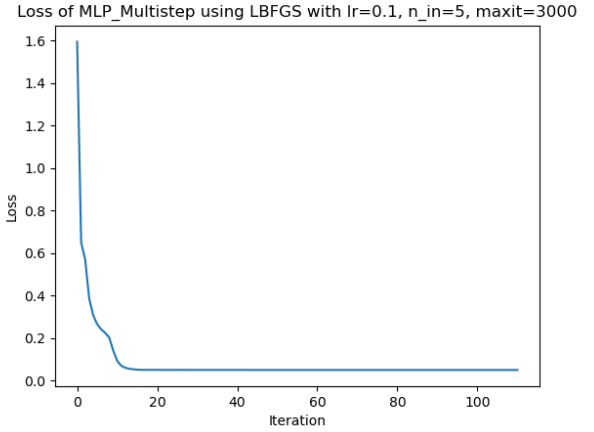
(a) MLPMultistep with 5 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 3 steps using LBFGS with learning rate 0.001

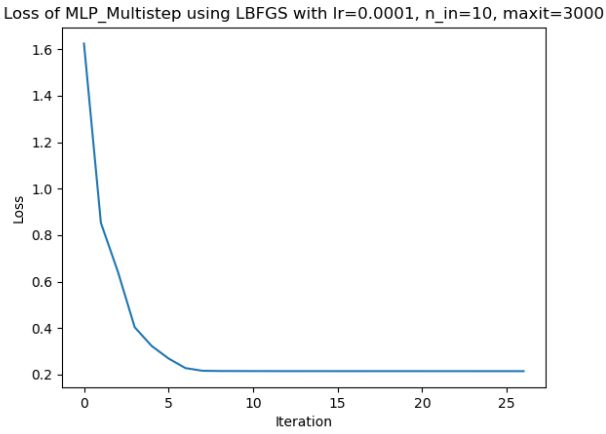


(c) MLPMultistep with 5 steps using LBFGS with learning rate 0.01

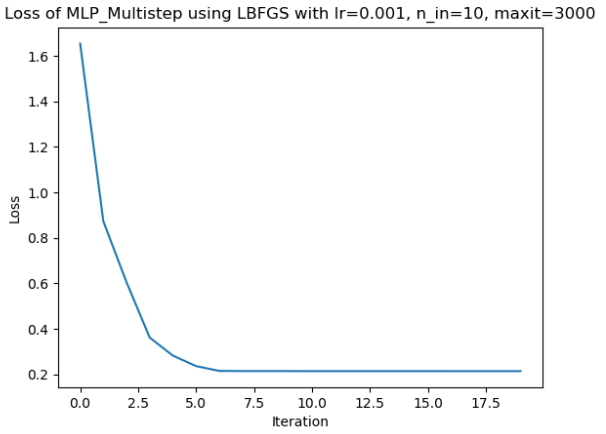


(d) MLPMultistep with 5 steps using LBFGS with learning rate 0.1

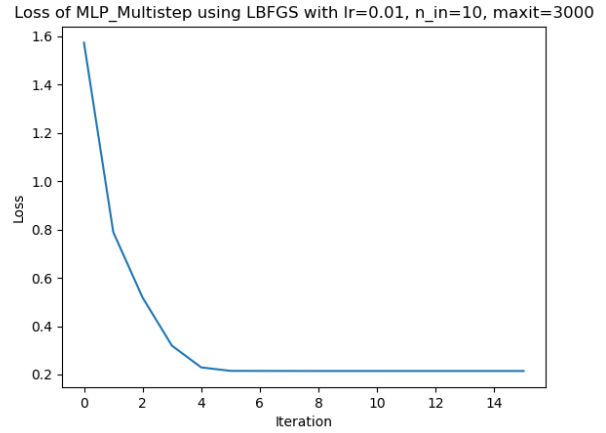
Figure (35) Prediction graphs using model MLPMultistep with 5 steps and optimizer LBFGS



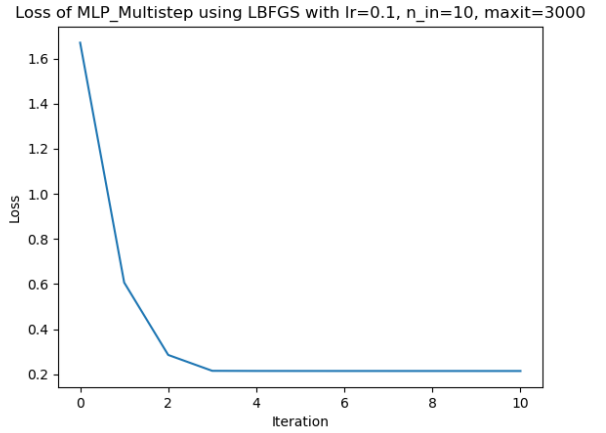
(a) MLPMultistep with 10 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 10 steps using LBFGS with learning rate 0.001

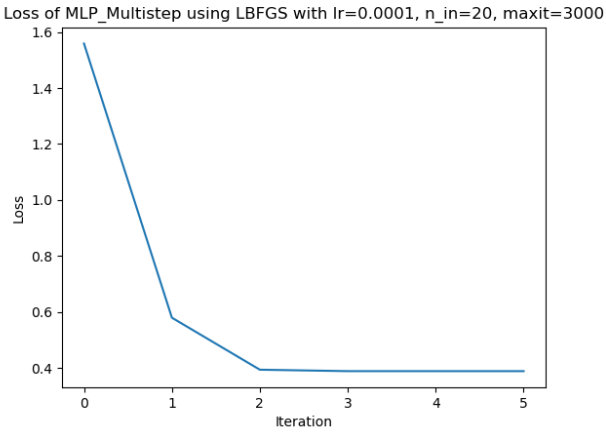


(c) MLPMultistep with 10 steps using LBFGS with learning rate 0.01

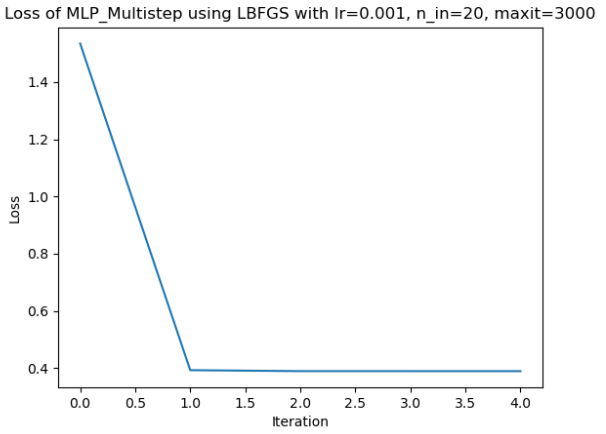


(d) MLPMultistep with 10 steps using LBFGS with learning rate 0.1

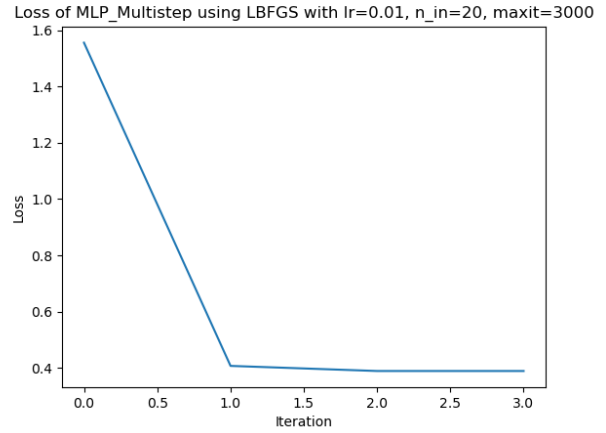
Figure (36) Prediction graphs using model MLPMultistep with 10 steps and optimizer LBFGS



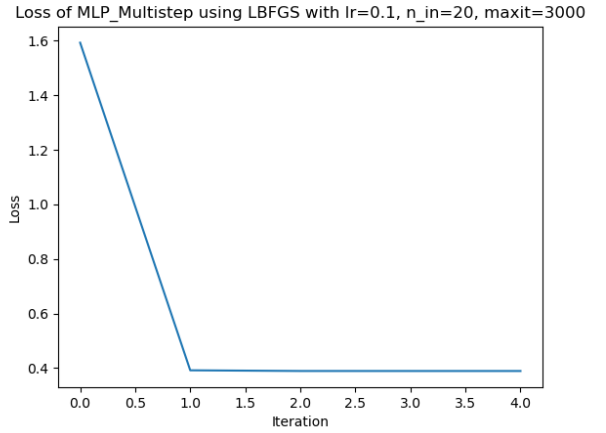
(a) MLPMultistep with 20 steps using LBFGS with learning rate 0.0001



(b) MLPMultistep with 20 steps using LBFGS with learning rate 0.001



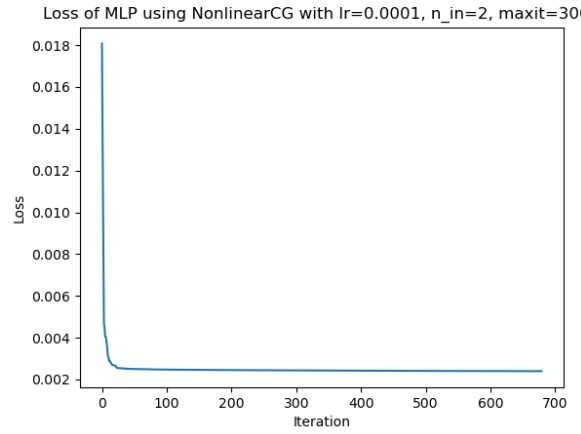
(c) MLPMultistep with 20 steps using LBFGS with learning rate 0.01



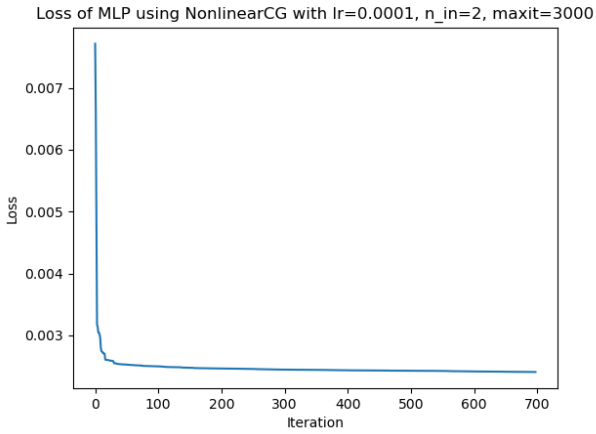
(d) MLPMultistep with 20 steps using LBFGS with learning rate 0.1

Figure (37) Prediction graphs using model MLPMultistep with 20 steps and optimizer LBFGS

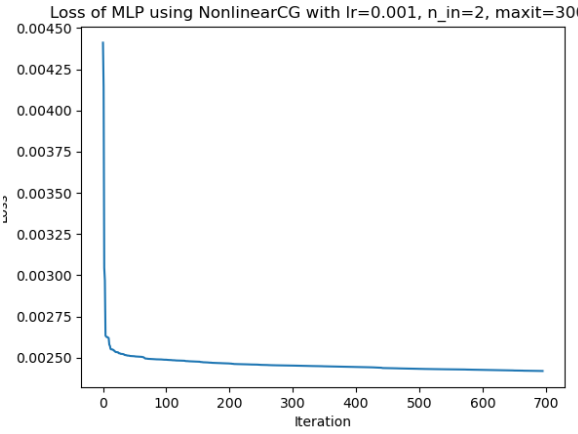
6.6.2. **NonlinearCG**



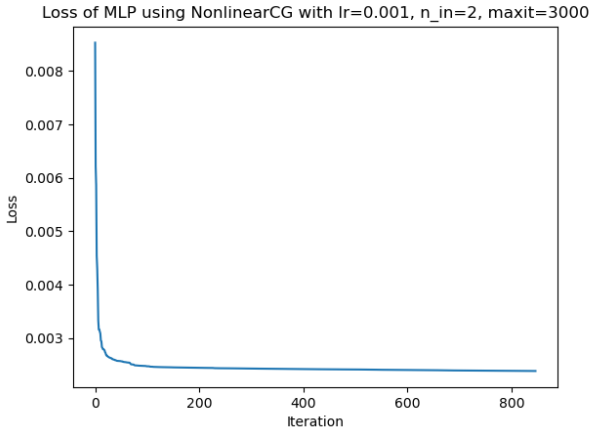
(a) MLP using NonlinearCG with learning rate 0.0001 and β^{HS}



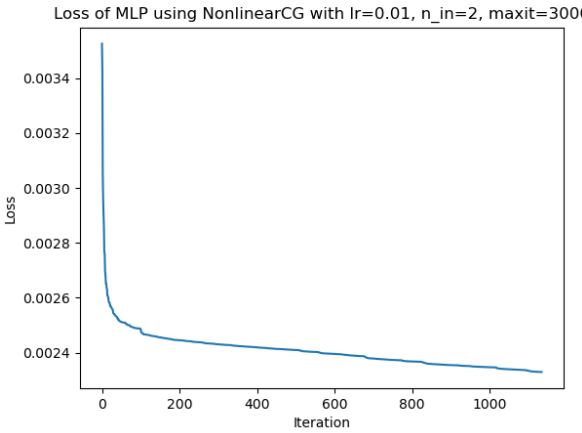
(b) MLP using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



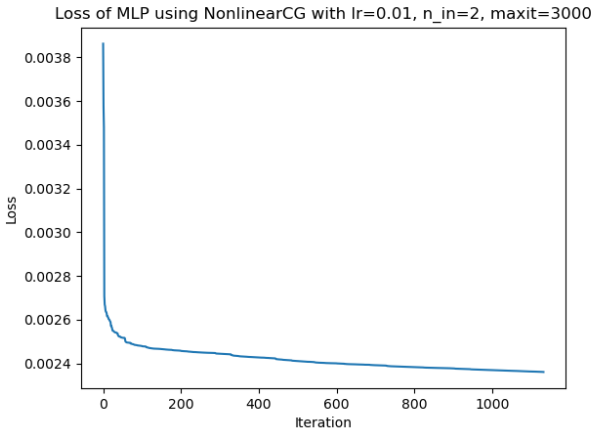
(c) MLP using NonlinearCG with learning rate 0.001 and β^{HS}



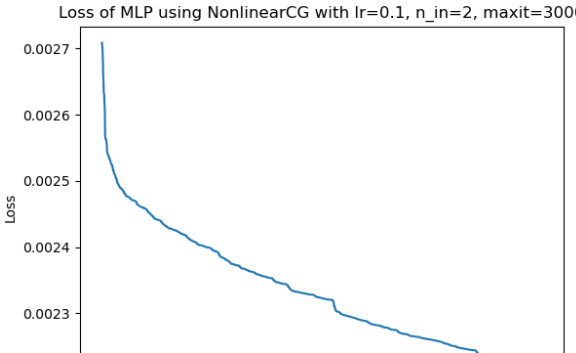
(d) MLP using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$

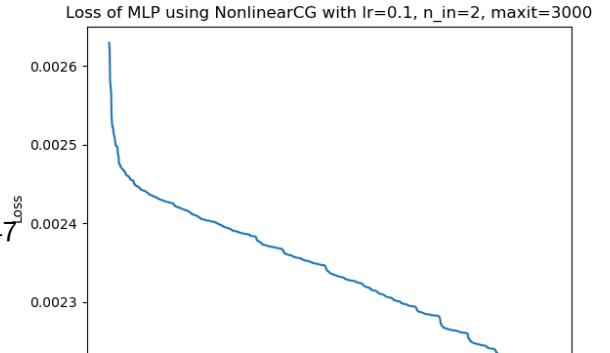


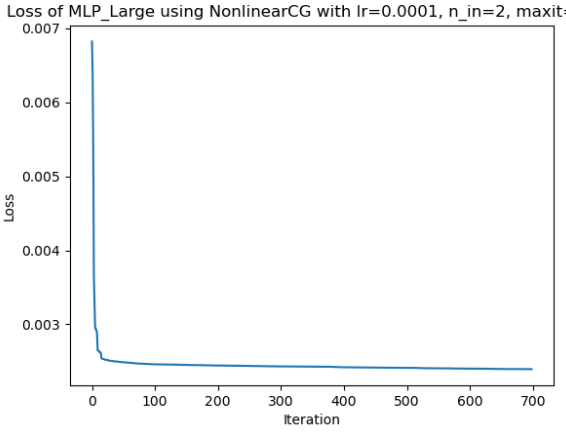
(e) MLP using NonlinearCG with learning rate 0.01 and β^{HS}



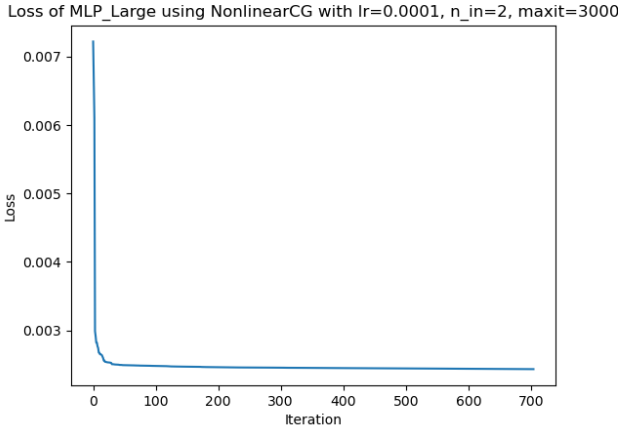
(f) MLP using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$



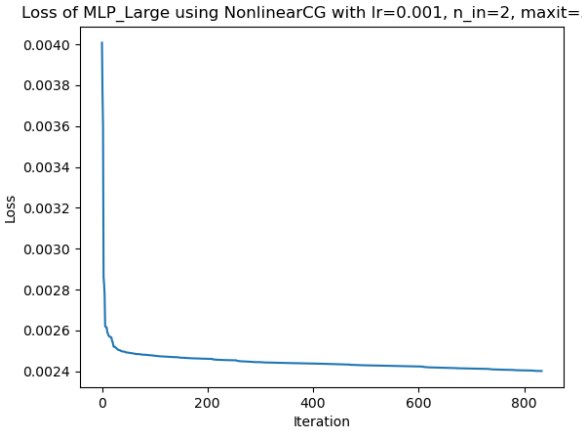




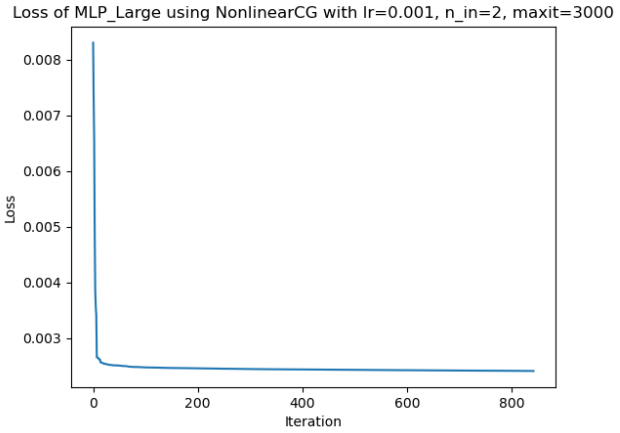
(a) MLPLarge using NonlinearCG with learning rate 0.0001 and β^{HS}



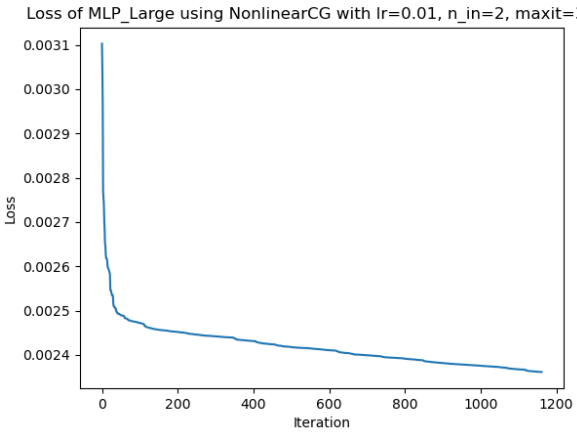
(b) MLPLarge using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



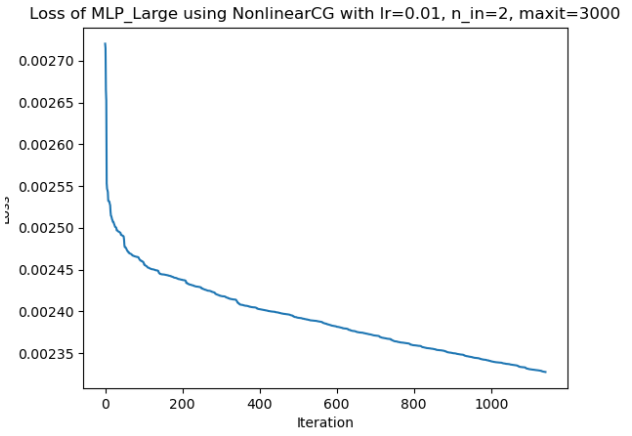
(c) MLPLarge using NonlinearCG with learning rate 0.001 and β^{HS}



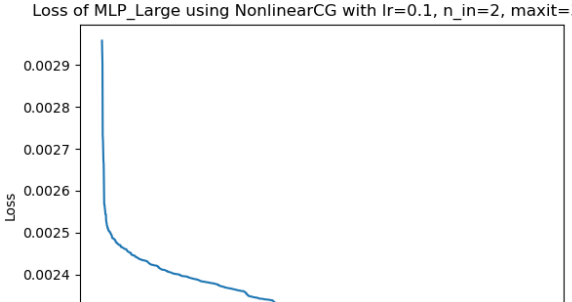
(d) MLPLarge using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$

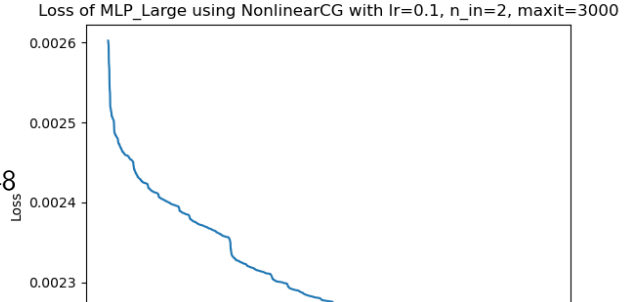


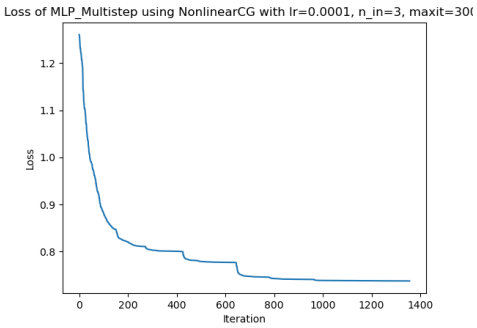
(e) MLPLarge using NonlinearCG with learning rate 0.01 and β^{HS}



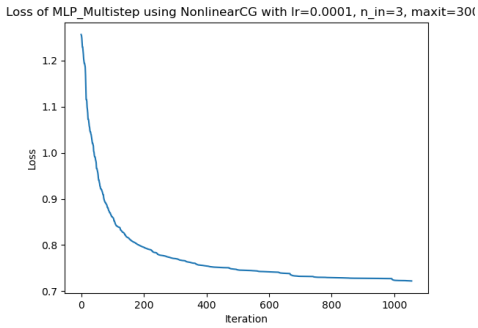
(f) MLPLarge using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$



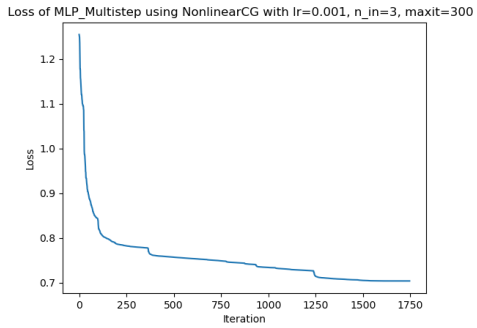




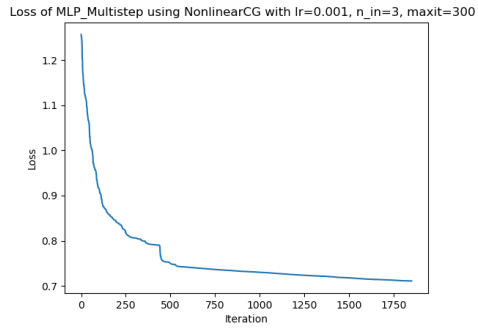
(a) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



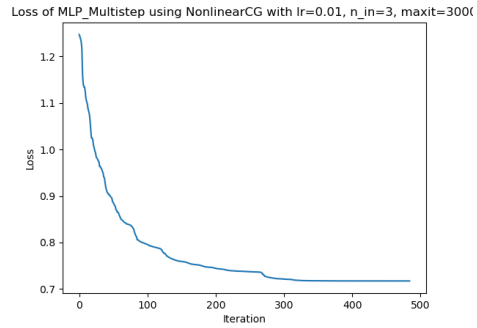
(b) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



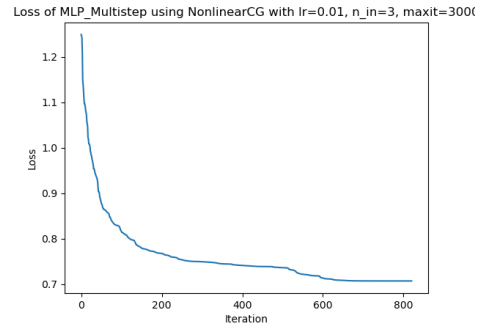
(c) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.001 and β^{HS}



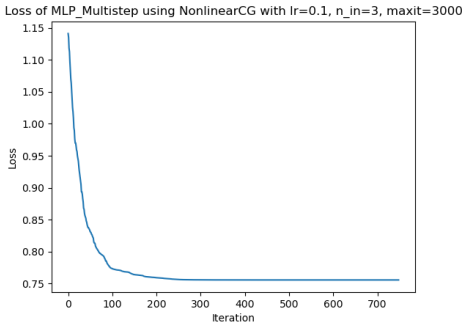
(d) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



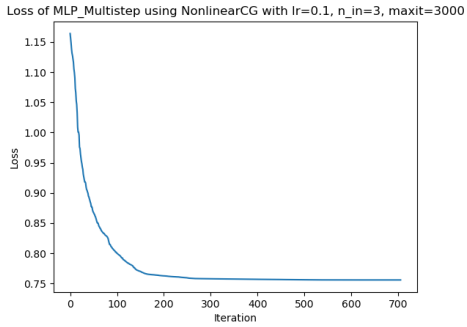
(e) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

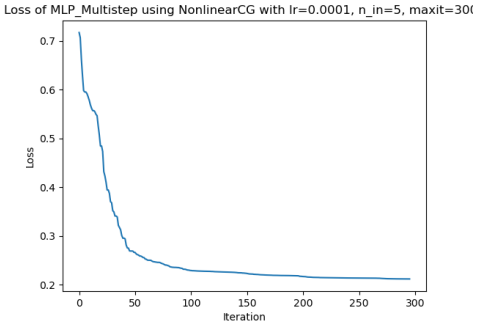


(g) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.1 and β^{HS}

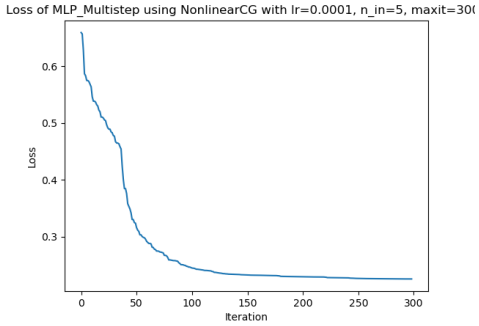


(h) MLPMultistep with 3 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

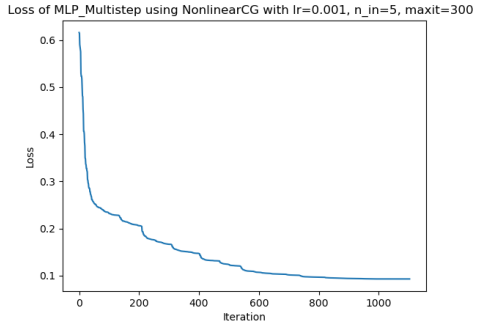
Figure (40) Prediction graphs using model MLPMultistep with 3 steps and optimizer NonlinearCG



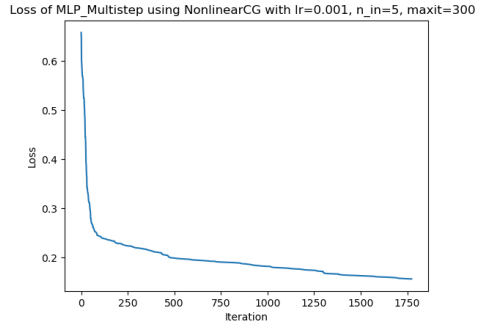
(a) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



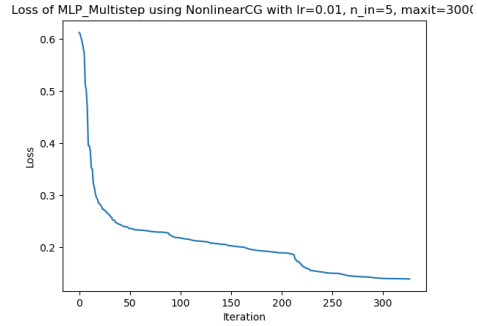
(b) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



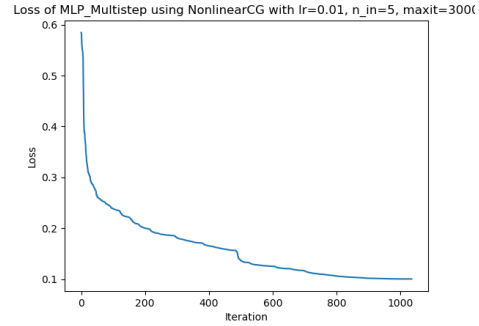
(c) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.001 and β^{HS}



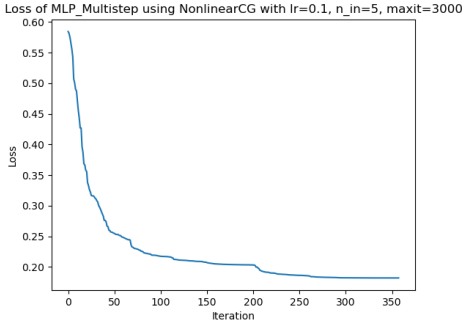
(d) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



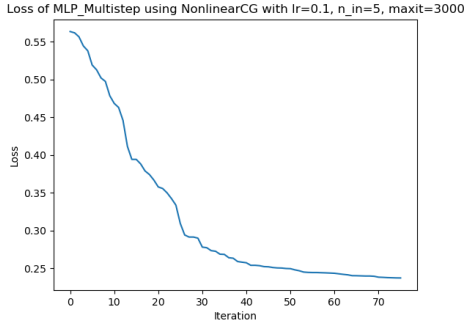
(e) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

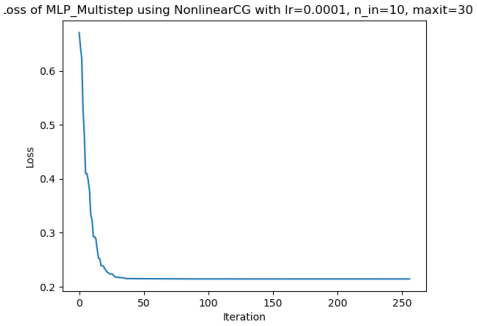


(g) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.1 and β^{HS}

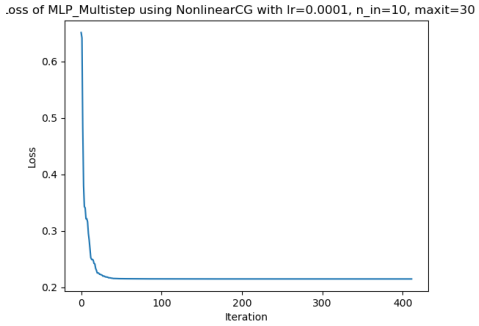


(h) MLPMultistep with 5 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

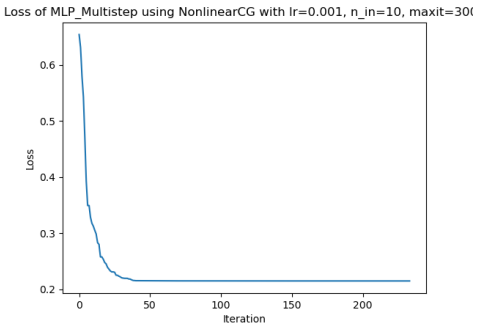
Figure (41) Prediction graphs using model MLPMultistep with 5 steps and optimizer NonlinearCG



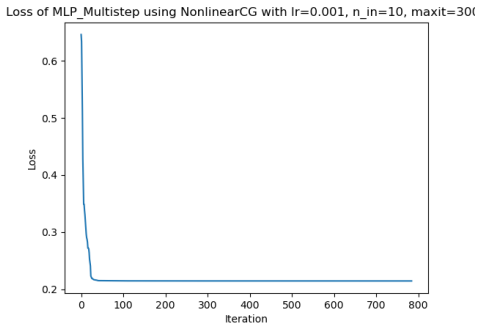
(a) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



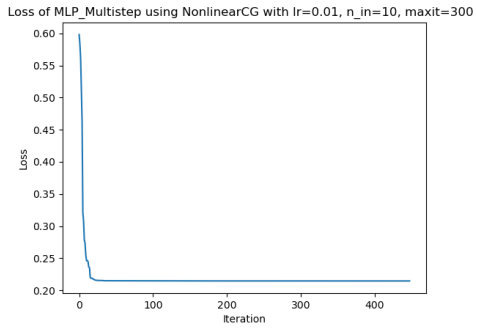
(b) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



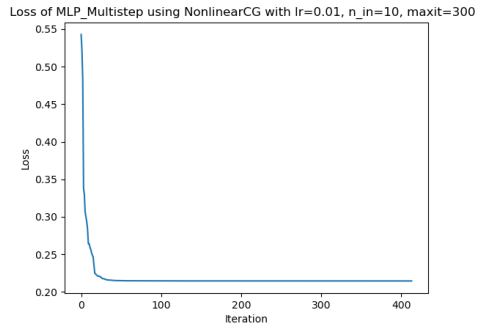
(c) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.001 and β^{HS}



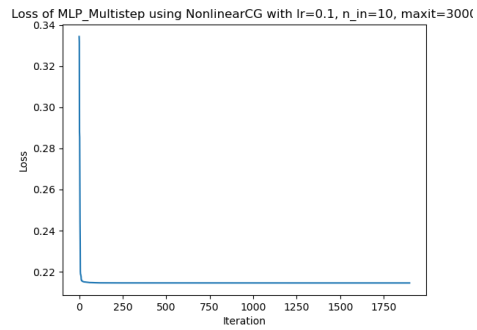
(d) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



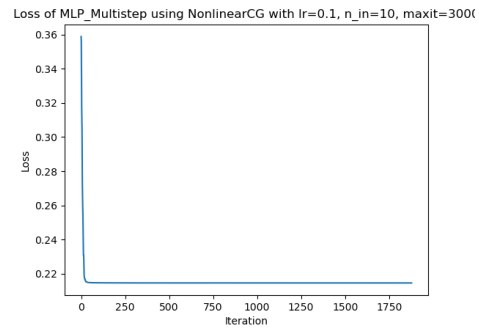
(e) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$

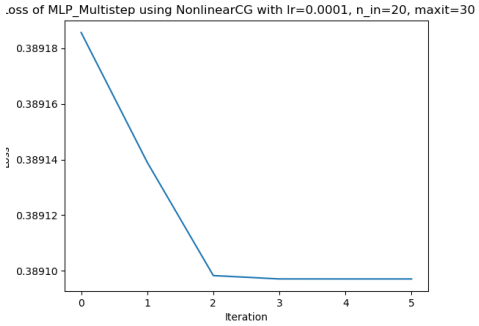


(g) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.1 and β^{HS}

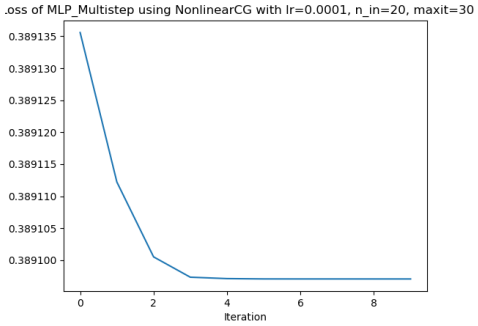


(h) MLPMultistep with 10 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

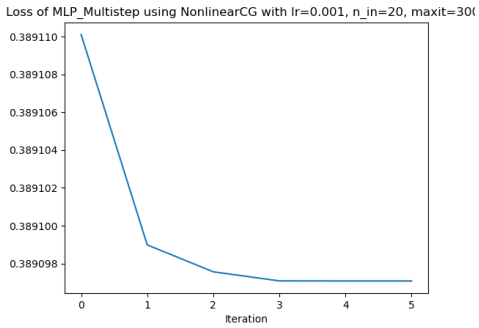
Figure (42) Prediction graphs using model MLPMultistep with 10 steps and optimizer NonlinearCG



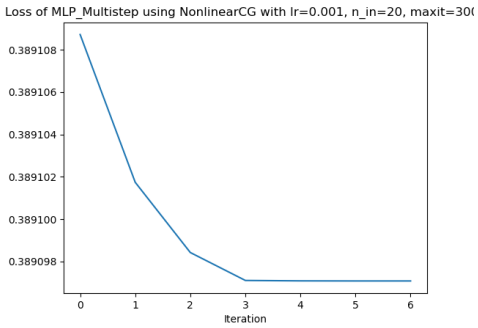
(a) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.0001 and β^{HS}



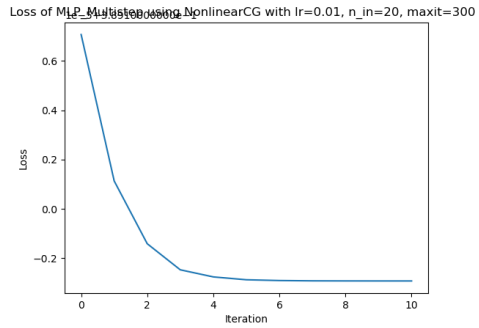
(b) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.0001 and $\beta^{\text{FR\_PR}}$



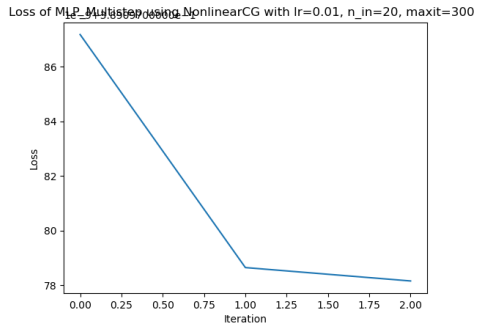
(c) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.001 and β^{HS}



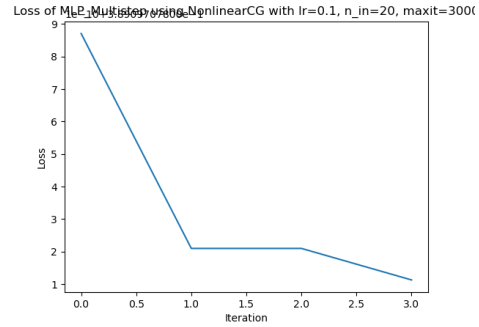
(d) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.001 and $\beta^{\text{FR\_PR}}$



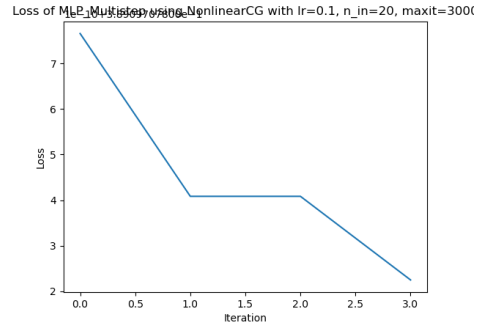
(e) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.01 and β^{HS}



(f) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.01 and $\beta^{\text{FR\_PR}}$



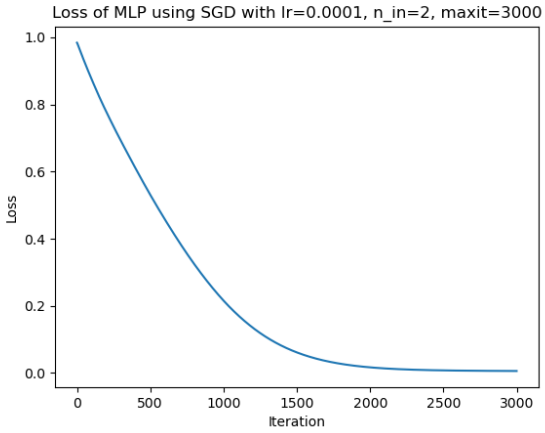
(g) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.1 and β^{HS}



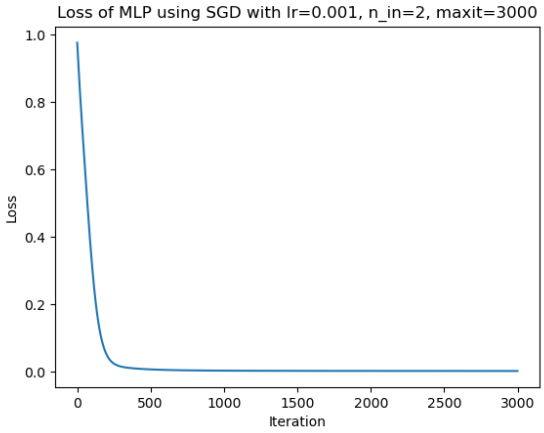
(h) MLPMultistep with 20 steps using NonlinearCG with learning rate 0.1 and $\beta^{\text{FR\_PR}}$

Figure (43) Prediction graphs using model MLPMultistep with 10 steps and optimizer NonlinearCG

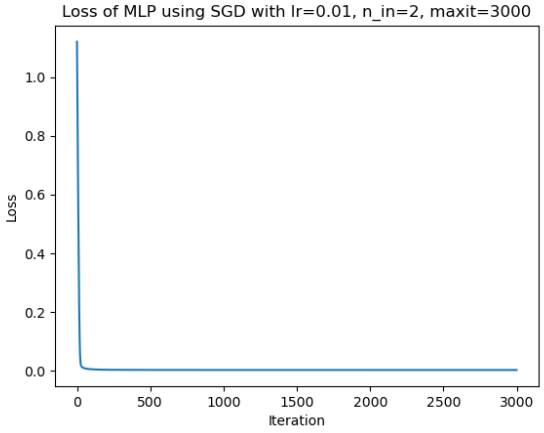
6.6.3. **SGD**



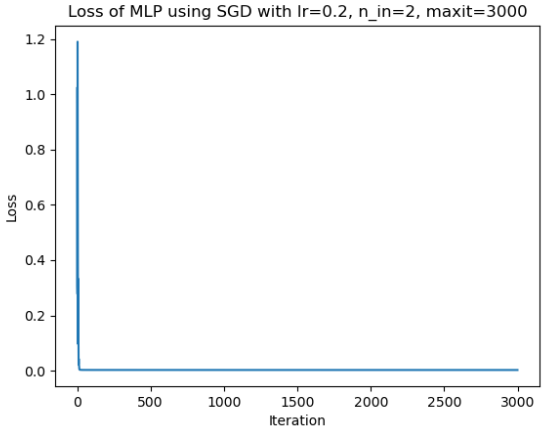
(a) MLP using SGD with learning rate 0.0001



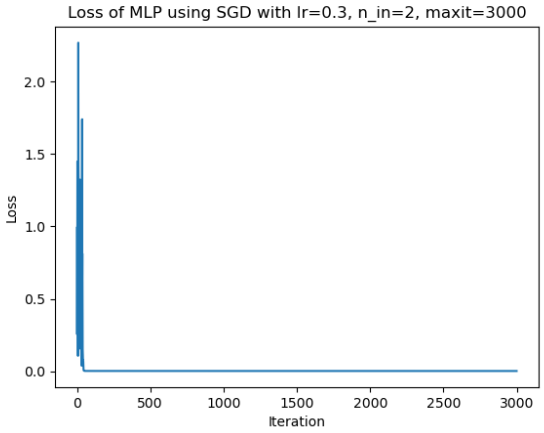
(b) MLP using SGD with learning rate 0.001



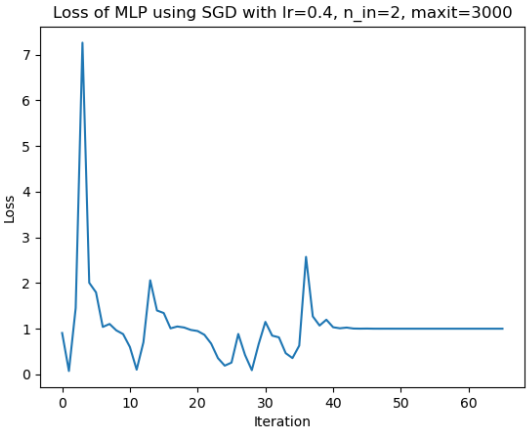
(c) MLP using SGD with learning rate 0.01



(d) MLP using SGD with learning rate 0.2

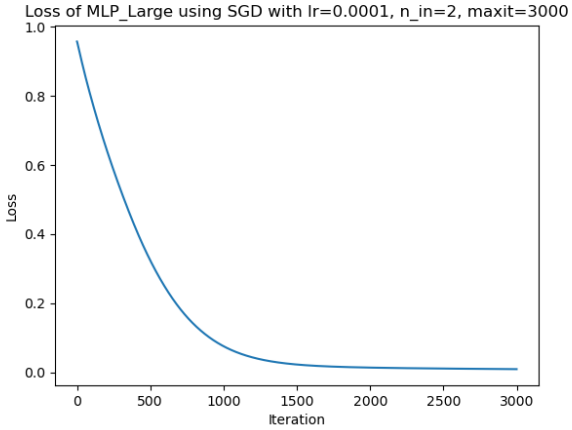


(e) MLP using SGD with learning rate 0.3

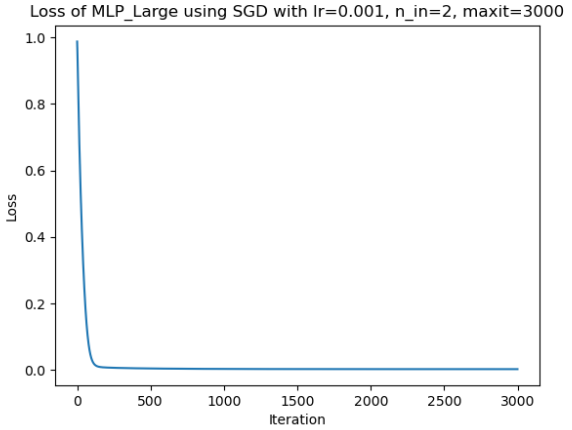


(f) MLP using SGD with learning rate 0.4

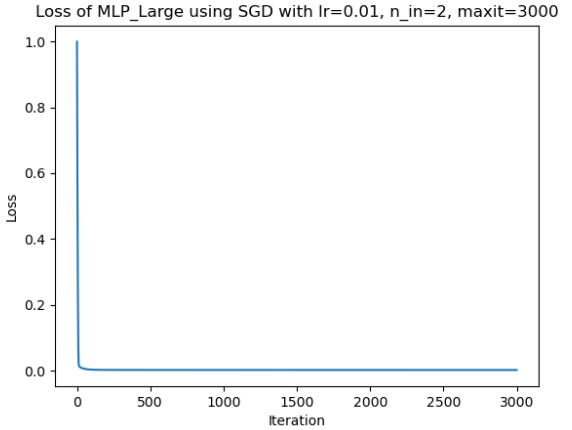
Figure (44) Prediction graphs using model MLP and optimizer SGD



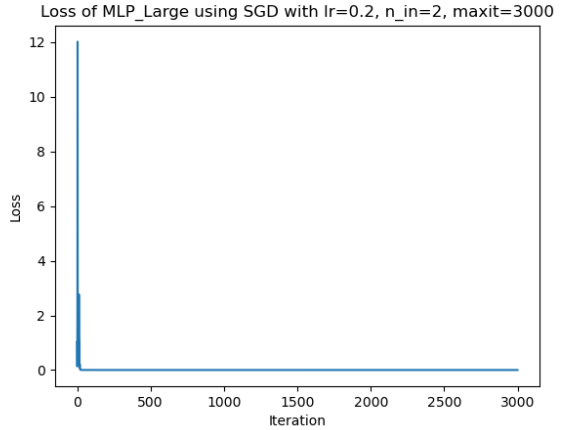
(a) MLPLarge using SGD with learning rate 0.0001



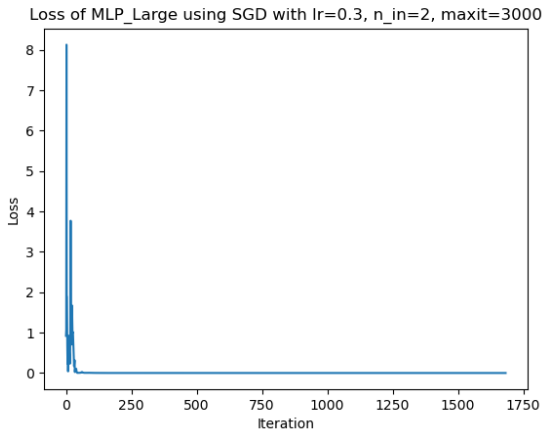
(b) MLPLarge using SGD with learning rate 0.001



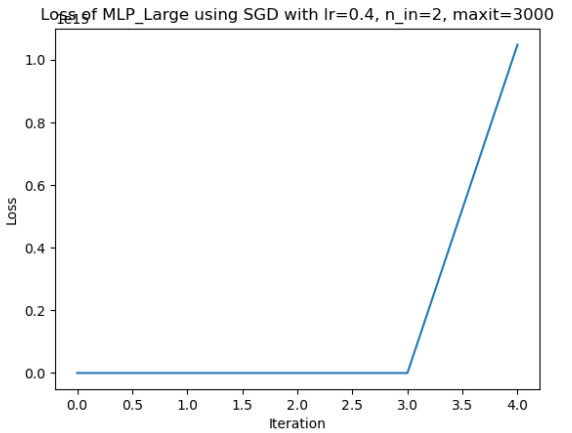
(c) MLPLarge using SGD with learning rate 0.01



(d) MLPLarge using SGD with learning rate 0.2

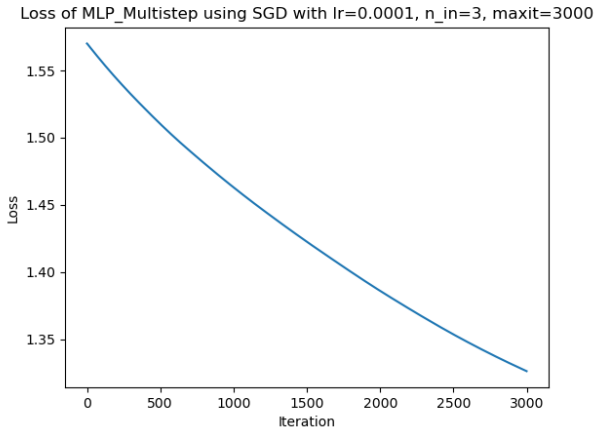


(e) MLPLarge using SGD with learning rate 0.3

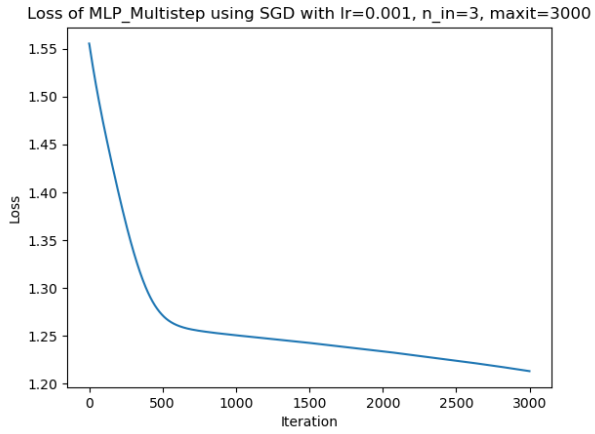


(f) MLPLarge using SGD with learning rate 0.4

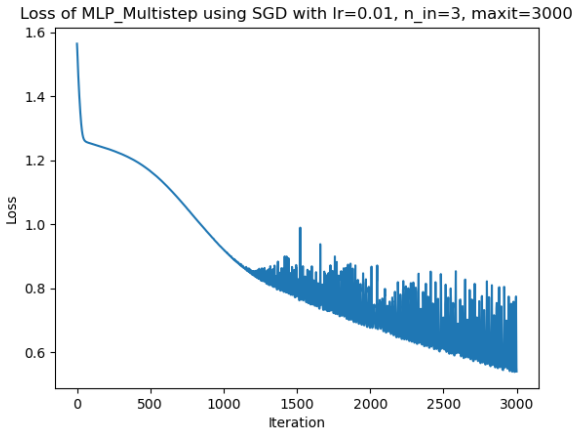
Figure (45) Prediction graphs using model MLPLarge and optimizer SGD



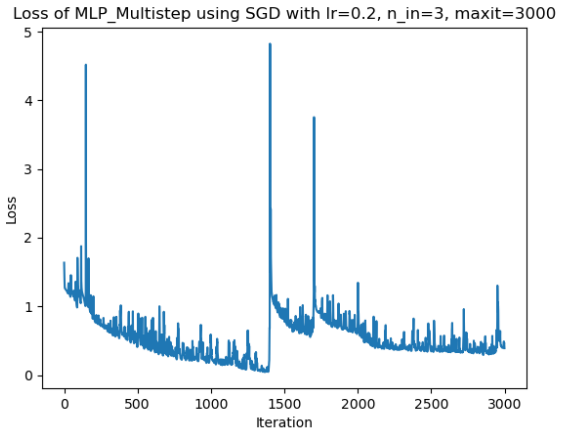
(a) MLPMultistep with 3 steps using SGD with learning rate 0.0001



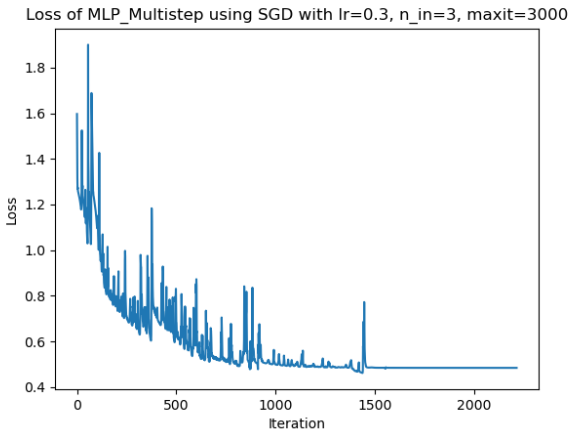
(b) MLPMultistep with 3 steps using SGD with learning rate 0.001



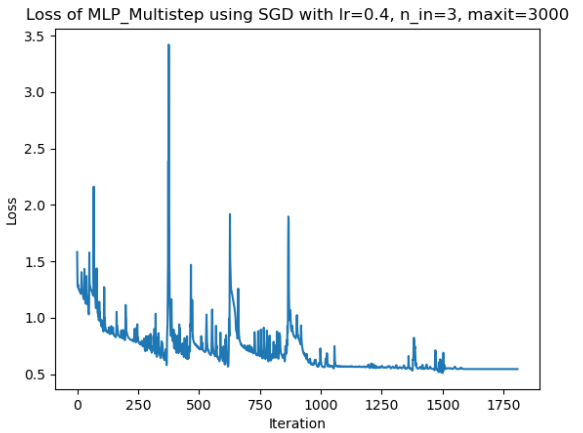
(c) MLPMultistep with 3 steps using SGD with learning rate 0.01



(d) MLPMultistep with 3 steps using SGD with learning rate 0.2

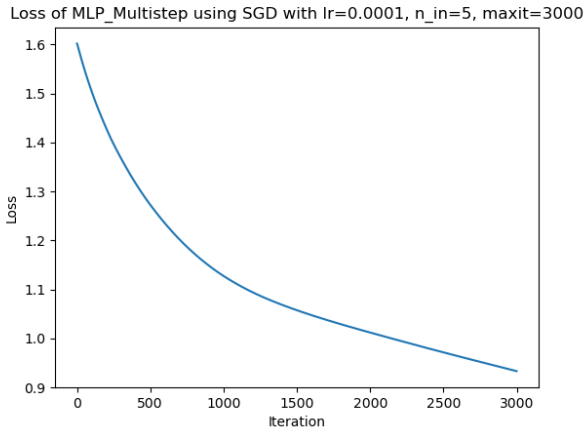


(e) MLPMultistep with 3 steps using SGD with learning rate 0.3

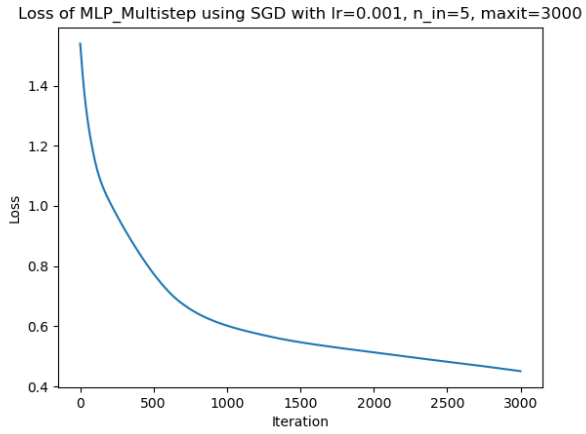


(f) MLPMultistep with 3 steps using SGD with learning rate 0.3

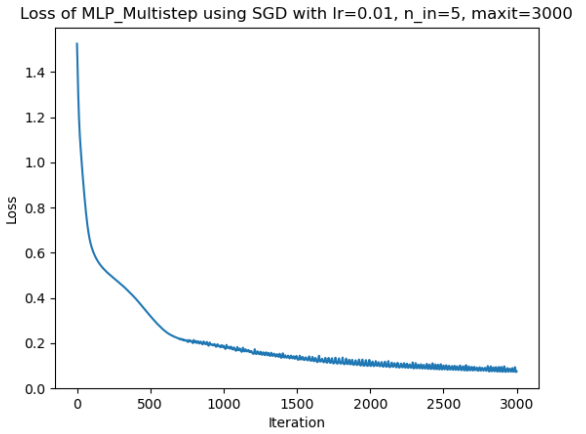
Figure (46) Prediction graphs using model MLPMultistep and optimizer SGD



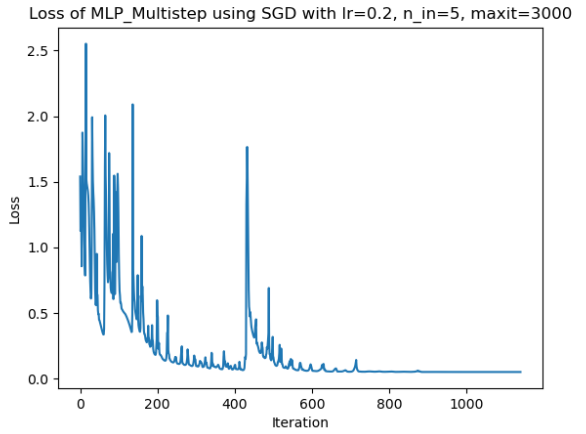
(a) MLPMultistep with 5 steps using SGD with learning rate 0.0001



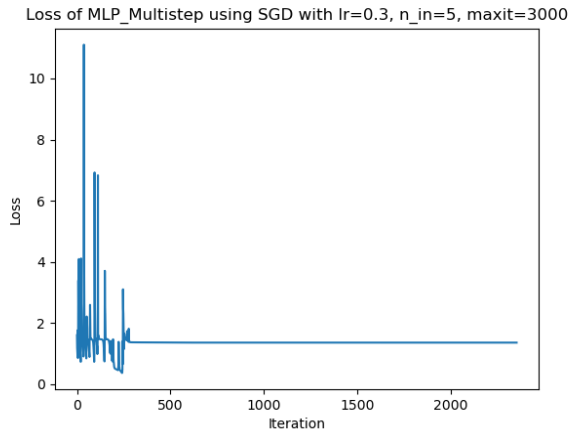
(b) MLPMultistep with 3 steps using SGD with learning rate 0.001



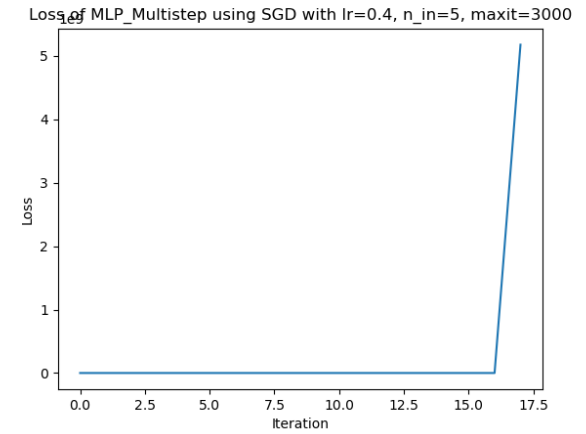
(c) MLPMultistep with 5 steps using SGD with learning rate 0.01



(d) MLPMultistep with 5 steps using SGD with learning rate 0.2

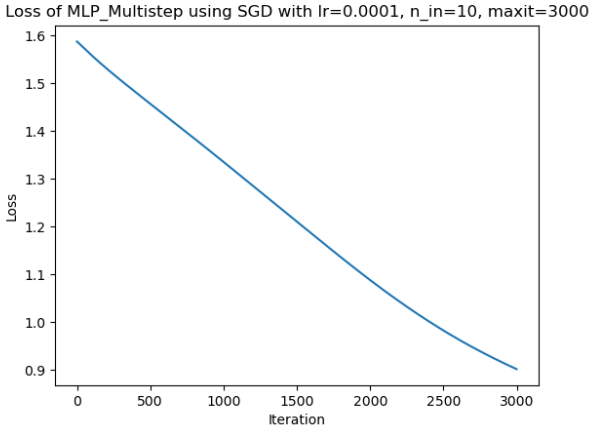


(e) MLPMultistep with 5 steps using SGD with learning rate 0.3

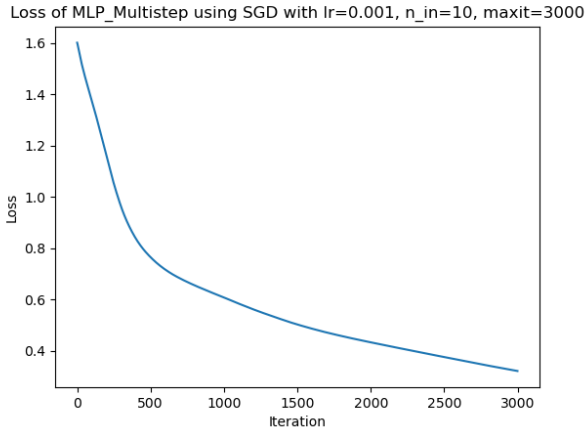


(f) MLPMultistep with 5 steps using SGD with learning rate 0.4

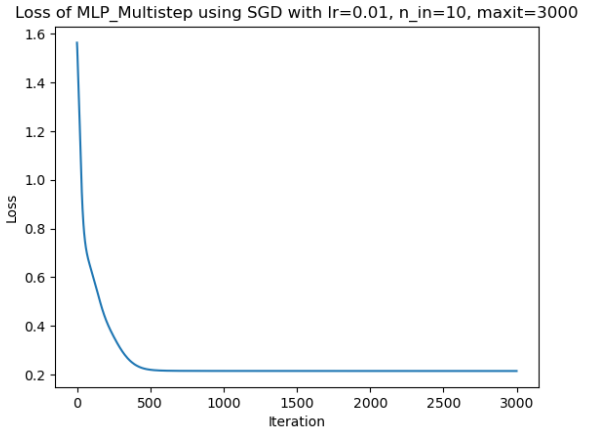
Figure (47) Prediction graphs using model MLPMultistep and optimizer SGD



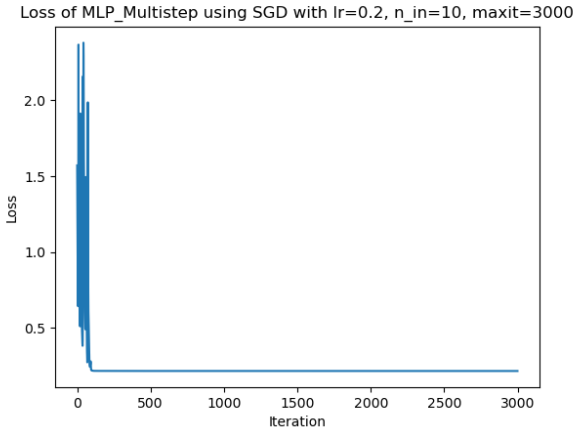
(a) MLPMultistep with 10 steps using SGD with learning rate 0.0001



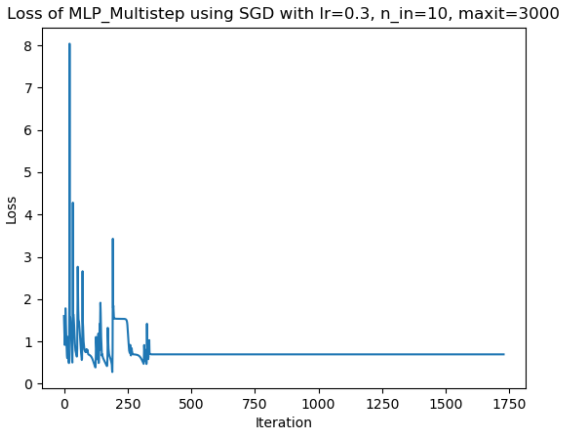
(b) MLPMultistep with 10 steps using SGD with learning rate 0.001



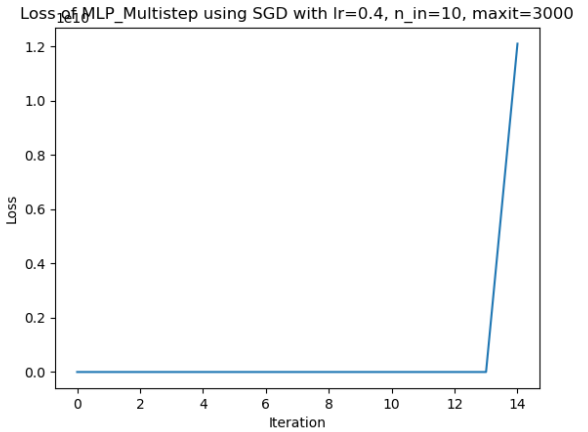
(c) MLPMultistep with 10 steps using SGD with learning rate 0.01



(d) MLPMultistep with 10 steps using SGD with learning rate 0.2

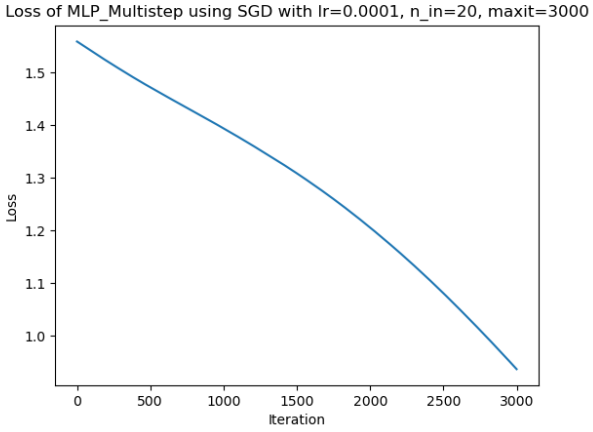


(e) MLPMultistep with 10 steps using SGD with learning rate 0.3

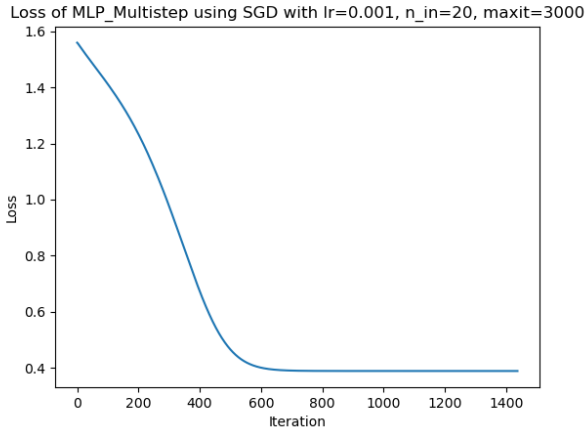


(f) MLPMultistep with 10 steps using SGD with learning rate 0.4

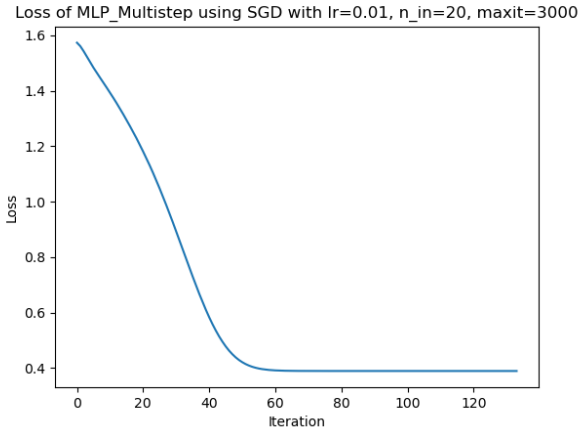
Figure (48) Prediction graphs using model MLPMultistep and optimizer SGD



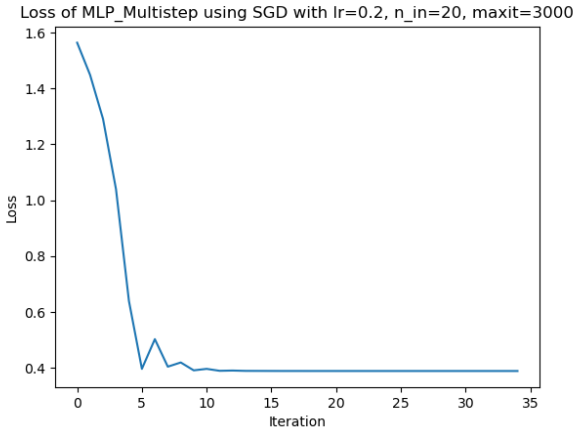
(a) MLPMultistep with 20 steps using SGD with learning rate 0.0001



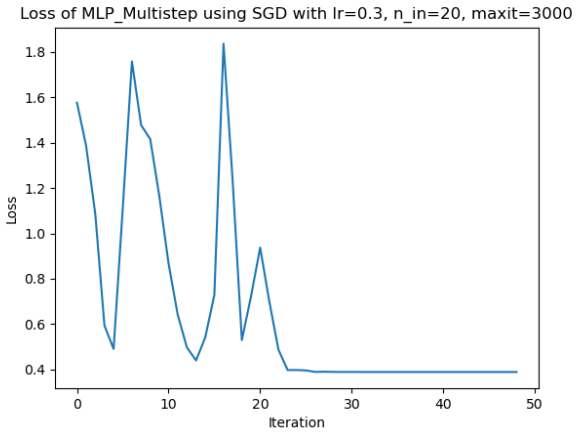
(b) MLPMultistep with 20 steps using SGD with learning rate 0.001



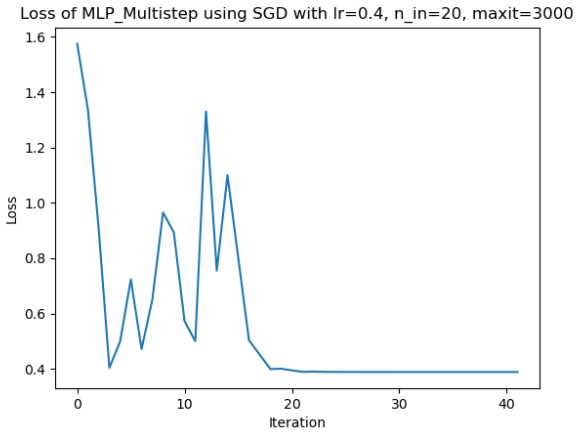
(c) MLPMultistep with 20 steps using SGD with learning rate 0.01



(d) MLPMultistep with 20 steps using SGD with learning rate 0.2



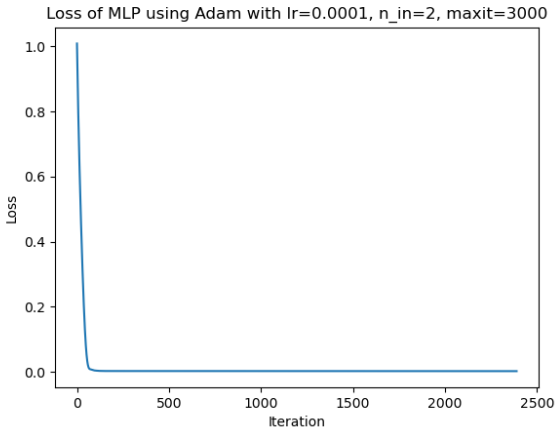
(e) MLPMultistep with 20 steps using SGD with learning rate 0.3



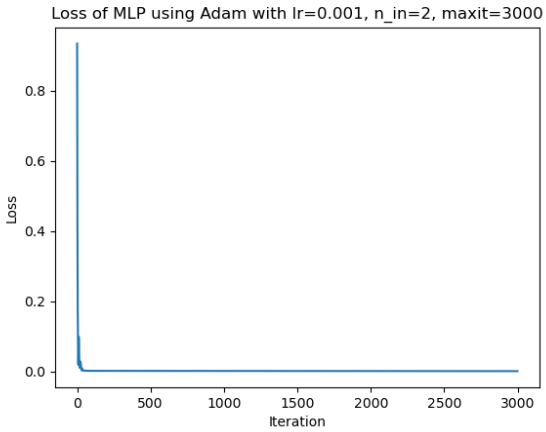
(f) MLPMultistep with 20 steps using SGD with learning rate 0.4

Figure (49) Prediction graphs using model MLPMultistep and optimizer SGD

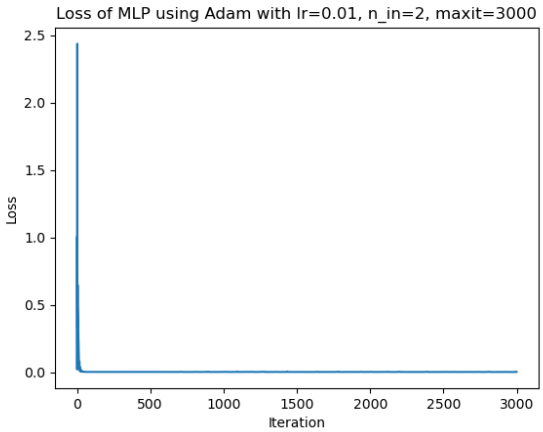
6.6.4. **Adam**



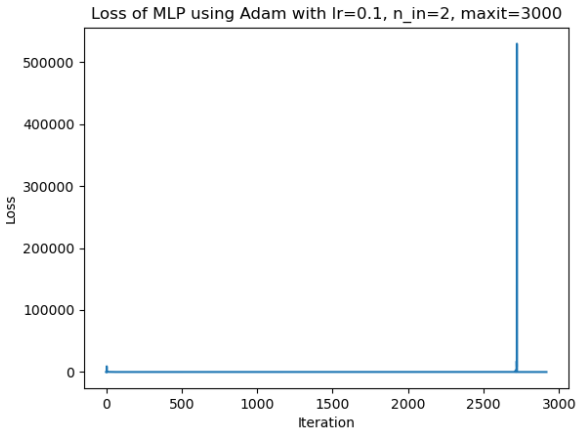
(a) MLP using Adam with learning rate 0.0001



(b) MLP using Adam with learning rate 0.001

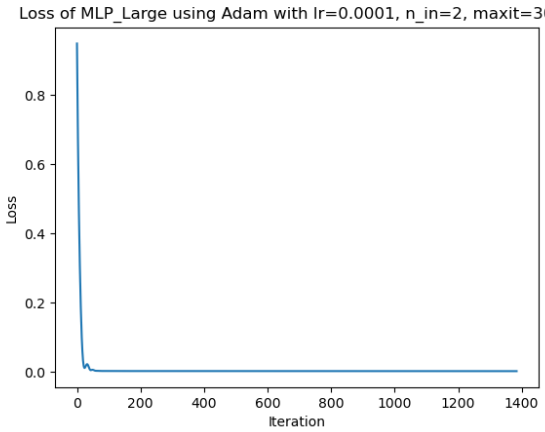


(c) MLP using Adam with learning rate 0.01

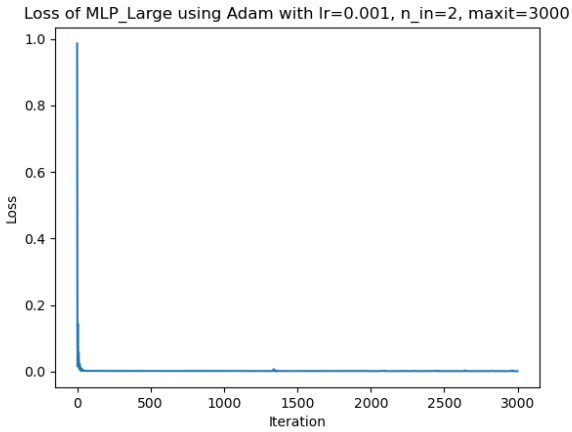


(d) MLP using Adam with learning rate 0.1

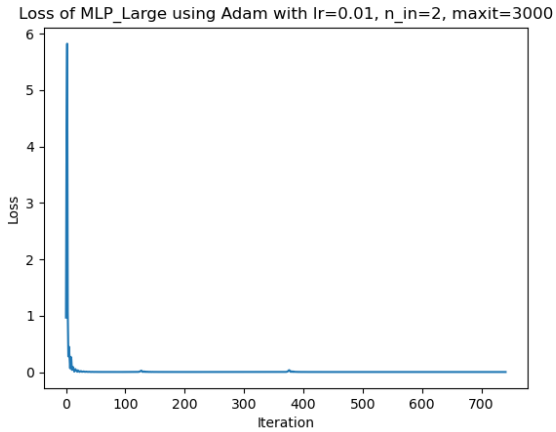
Figure (50) Prediction graphs using model MLP and optimizer Adam



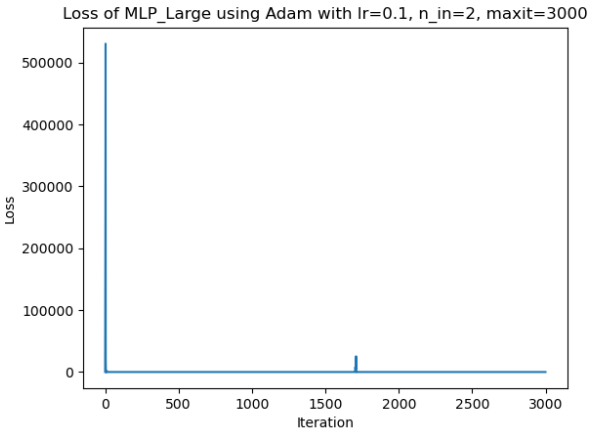
(a) MLP using Adam with learning rate 0.0001



(b) MLP using Adam with learning rate 0.001

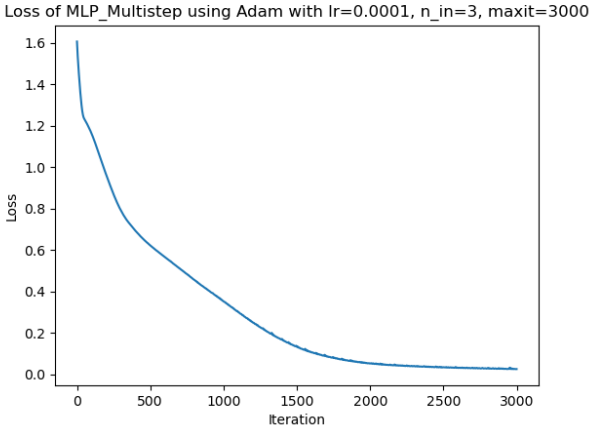


(c) MLP using Adam with learning rate 0.01

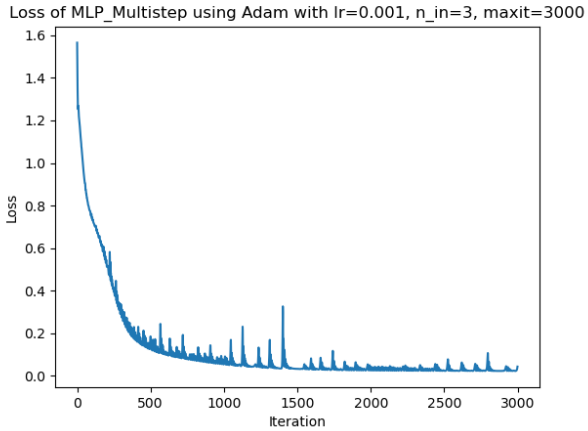


(d) MLP using Adam with learning rate 0.1

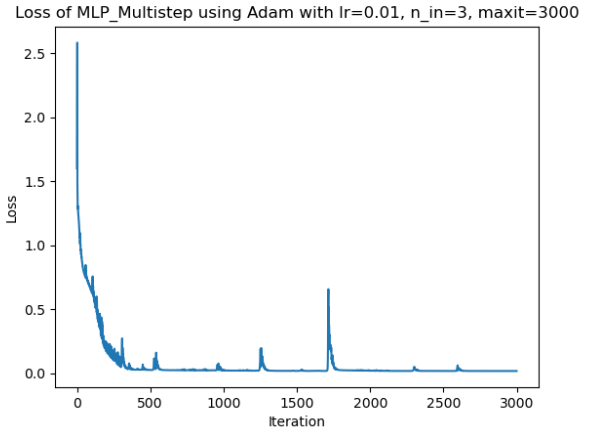
Figure (51) Prediction graphs using model MLPLarge and optimizer Adam



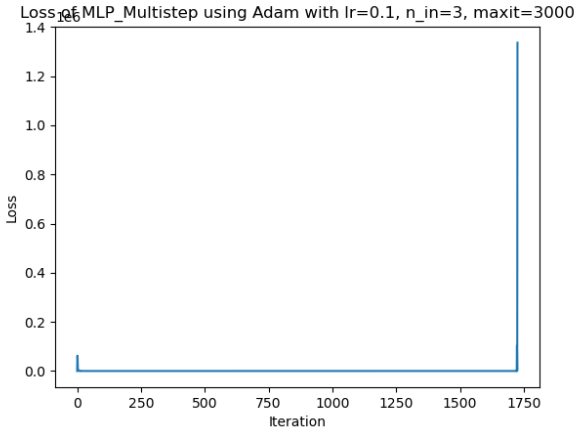
(a) MLPMultistep with 3 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 3 steps using Adam with learning rate 0.001

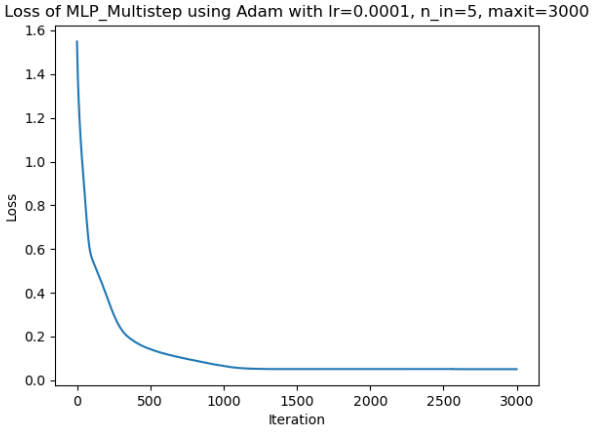


(c) MLPMultistep with 3 steps using Adam with learning rate 0.01

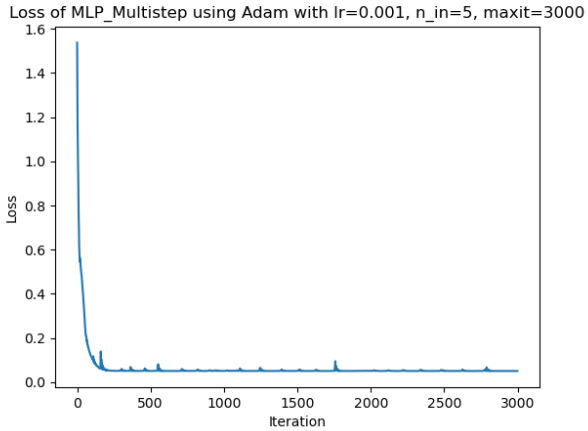


(d) MLPMultistep with 3 steps using Adam with learning rate 0.1

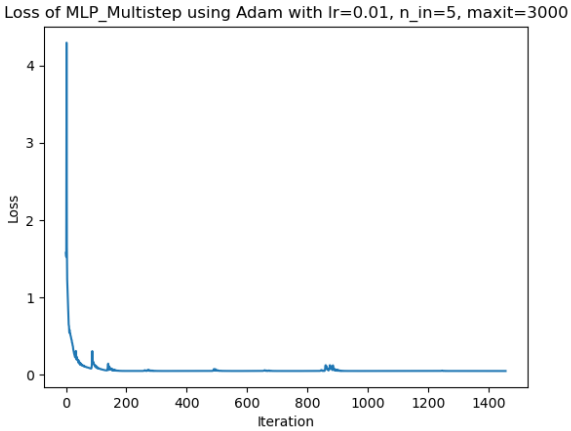
Figure (52) Prediction graphs using model MLPMultistep with 3 steps and optimizer Adam



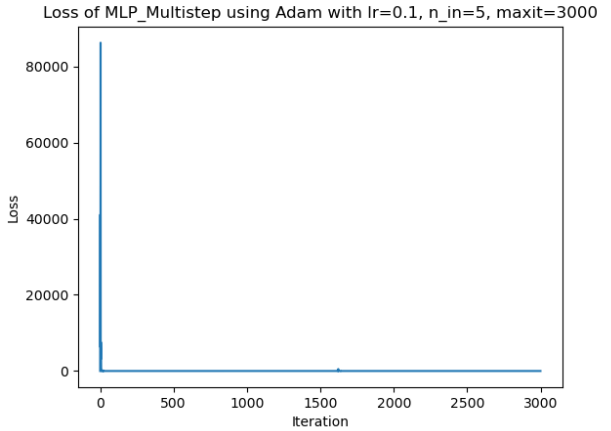
(a) MLPMultistep with 5 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 3 steps using Adam with learning rate 0.001

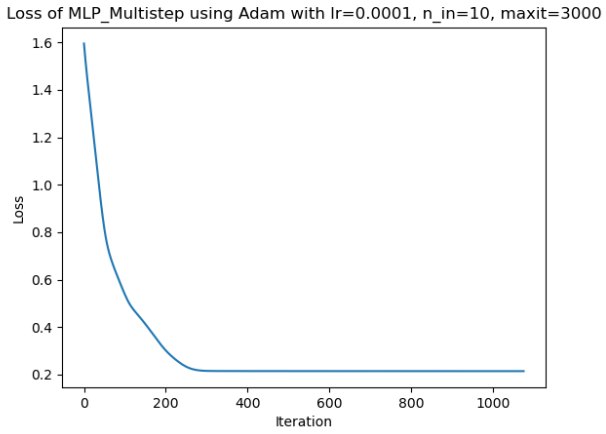


(c) MLPMultistep with 5 steps using Adam with learning rate 0.01

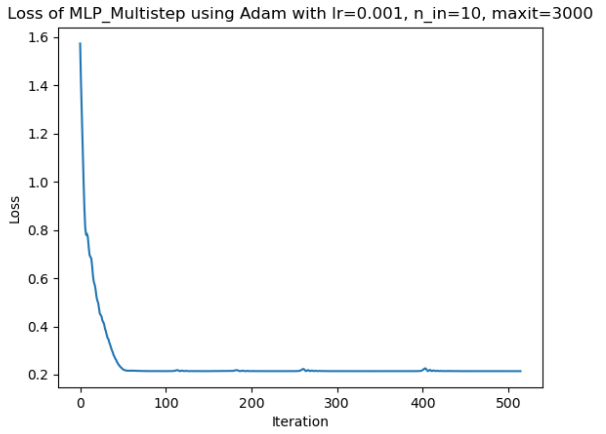


(d) MLPMultistep with 5 steps using Adam with learning rate 0.1

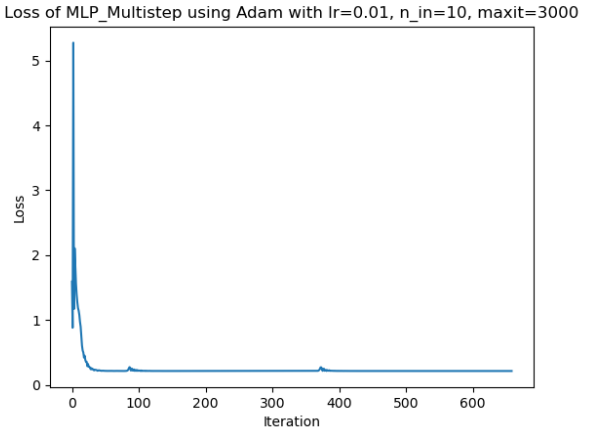
Figure (53) Prediction graphs using model MLPMultistep with 5 steps and optimizer Adam



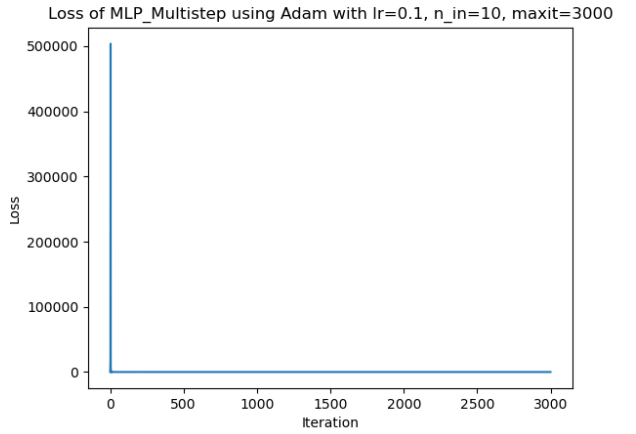
(a) MLPMultistep with 10 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 10 steps using Adam with learning rate 0.001

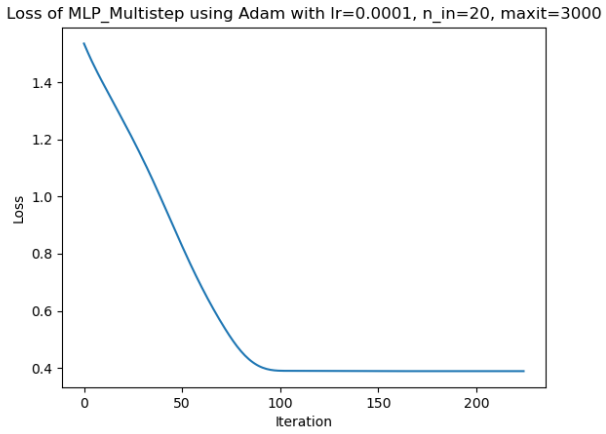


(c) MLPMultistep with 10 steps using Adam with learning rate 0.01

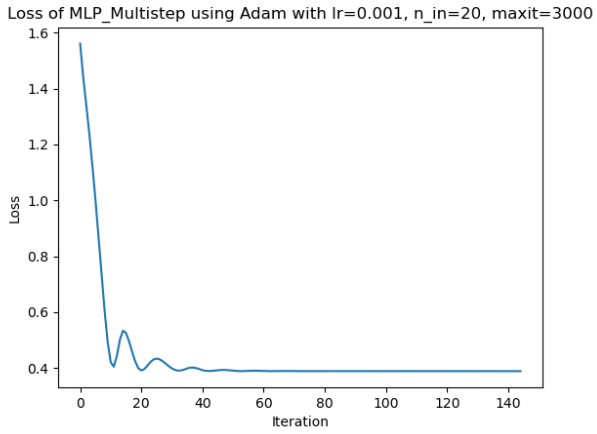


(d) MLPMultistep with 10 steps using Adam with learning rate 0.1

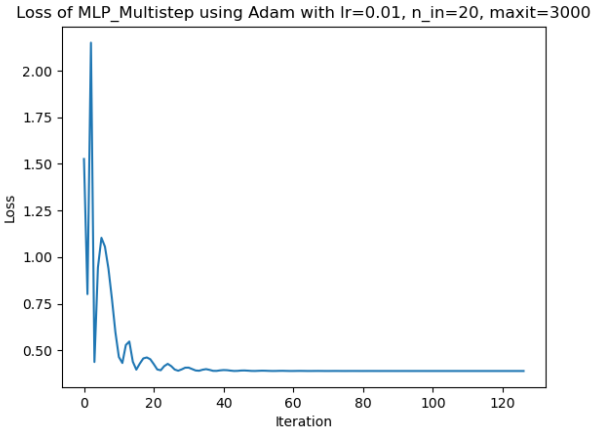
Figure (54) Prediction graphs using model MLPMultistep with 10 steps and optimizer Adam



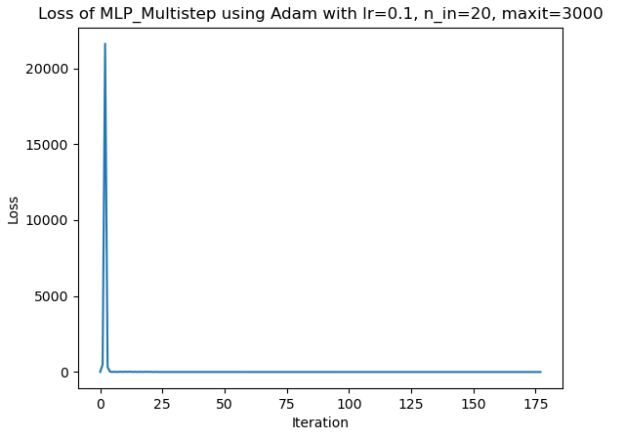
(a) MLPMultistep with 20 steps using Adam with learning rate 0.0001



(b) MLPMultistep with 20 steps using Adam with learning rate 0.001



(c) MLPMultistep with 20 steps using Adam with learning rate 0.01



(d) MLPMultistep with 20 steps using Adam with learning rate 0.1

Figure (55) Prediction graphs using model MLPMultistep with 20 steps and optimizer Adam

References

[1] Boris Banushev. *stockpredictionai*. https://github.com/borisbanushev/stockpredictionai. 2018.

[2] Chris Bishop. "Exact Calculation of the Hessian Matrix for the Multilayer Perceptron". In: *Neural Computation* 4.4 (July 1992), pp. 494–501. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.4.494. eprint: https://direct.mit.edu/neco/article-pdf/4/4/494/812313/neco.1992.4.4.494.pdf. URL: https://doi.org/10.1162/neco.1992.4.4.494.

[3] Dozat. *Incorporating Nesterov Momentum into Adam*. URL: https://cs229.stanford.edu/proj2015/054_report.pdf.

[4] Daisuke Ishii. *Uniqlo (FastRetailing) Stock Price Prediction*. data retrieved from Kaggle, https://www.kaggle.com/daiearth22/uniqlo-fastretailing-stock-price-prediction. 2016.

[5] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[6] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[7] Margaret Wright. *Notes 11 in Advanced Topics in Numerical Analysis: Numerical Optimization*. Dec. 2021.