COMPARING EXTENDED KALMAN ALGORITHM WITH STEEPEST DESCENT WITH BACK PROPAGATION

ECEN 5733 – Neural Networks Final Project

Instructor – Dr. Martin Hagan

Prepared by - Srinivas Sambaraju

5/3/2013

This Project report compares the Extended Kalman Filter (EKF) Algorithm with the Steepest Descent with Back Propagation Algorithm(SDBP) to determine the convergence criteria and efficiency of the EKF Algorithm by viewing the problem of training feed forward Neural Networks as an identification problem for solving non-linear dynamic system problems. A sine wave is taken as the target and the Network is trained by both EKF and SDBP and the results are compared. This project is based on the paper titled "Training Feed Forward Networks with the Extended Kalman Algorithm" written by Sharad Singhal and Lance Wu

Contents

Introduction

The project compares the back propagation algorithm with steepest descent(BPSD) with the Extended Kalman Filter Algorithm(EKF). The main goal is to compare the convergence criteria, sum squared errors, regression plots, computation time and resources.

The paper is titled "Training Feed Forward Networks with the Extended Kalman Algorithm". It was published by Sharad Singhal and Lance Wu in 1989. In the historical perspective the Paper gives a hint about the status of the Research and applications of Neural Networks around 1989.

The background work for the field of Neural Networks started in the late 19th and early 20th centuries. The modern view of neural networks began in the 1940's with the work of Warren McCulloch and Walter Pitts who showed that artificial neurons could in principle compute an arithmetic or logical function. This was followed by Donald Hebb who proposed a mechanism for learning in biological neurons.

The perceptron learning rule proposed by Frank Rosenblatt in the 1950's was able to solve certain pattern recognition problems and it was the first practical application of artificial Neural Networks. Around 1960, Bernard Widrow and Marcian Hoff introduced the Widrow-Hoff learning rule to train adaptive linear neural networks(ADALINE) using the LMS Algorithm.

The ADALINE was similar to the perceptron except that the transfer function was linear instead of hard limit. Both were designed to train single-layer perceptron like networks and could only solve linearly separable problems. Marvin Minsky and Seymour Papert criticized Neural networks in the 1960's about the single layer perceptron not being to solve elementary problems like XOR gate.

The 1980's saw a revival of interest in Neural Networks; the important was the publication of the back propagation algorithm by David Rumelhart and James McClelland which many researchers also discovered independently. This was an answer to the criticisms of Marvin Minsky and Seymour Papert. The other publication was Associative memory using a recurrent network by John Hopfield.

Around this time computing was an expensive resource and most of the research was focused on optimization. The research as illustrated in this papers references section hints at areas like Speech recognition, parallel distributed processing, Kalman Filter, linear and non-linear filtering and prediction, optimal filtering and estimation etc. This paper seems to have shown a computationally efficient way of solving these complex problems using the Extended Kalman Filter.

Paper Review

The conversion of Paper notation to Class notation is added in the appendix as it was not possible to add the scanned pdf here.

The Back Propagation algorithm converges slowly for large or complex problems such as speech recognition and requires thousands of iterations even for relatively small data sets. The paper presents the Extended Kalman algorithm as an alternative to Back Propagation in some cases where it computes parameters quickly although its computationally more expensive. The Kalman Filter Algorithm works only with linear systems. In the paper, an idea is presented with the Extended Kalman Filter. In the paper, training feed forward networks is shown as an identification problem for a non-linear dynamic system which can be solved using the Extended Kalman algorithm. Multi-layer perceptron's with back propagation algorithm is time consuming as it is a LMS (Least mean squared) gradient algorithm that ignores information from previous data after the weight update. The convergence of the back propagation algorithm is poor if the problem is complex or the number of nodes is large.

Extended Kalman algorithm updates parameters consistent with all previously seen data – The updated estimate of aposteriori covariance is carried forward from the previous iteration. The Kalman algorithm computes the optimum value of the system parameters as each new data point becomes available. By linearizing the system around the current estimate of the parameters, the Kalman algorithm can be applied to a non-linear dynamic system.

The initial state is of the network before training {x(0)} is defined by populating the network with Gaussian random variables with normal distribution. To prevent data saturation (a situation where the error curve stays flat and appears to have converged), windowing approach is used in the paper where more weight is given to the new data.

Program Description

In this project I am only comparing EKF with BPSD for a sine function given below.

g(p) = exp(p/2)*sin(pi*p/2)

Two different 1-N-1 Networks with the below configuration were used.

- @1-2-1 Network
- @1-4-1 Network

Two different weights and biases were used for the above two networks.
Thus there are a total of 4 test cases involved.

A test case includes the following plots for each algorithm to test the performance of the two algorithms on the network.

- Network Function vs Actual function
- Sumsquared error and validation error over the number of iterations
- The overlay plot of sum squared error for EKF and BPSD
- A regression plot of targets vs output from the Weights and biases obtained from EKF and BPSD Algorithms

The program flow is illustrated in the flow chart in the next page.

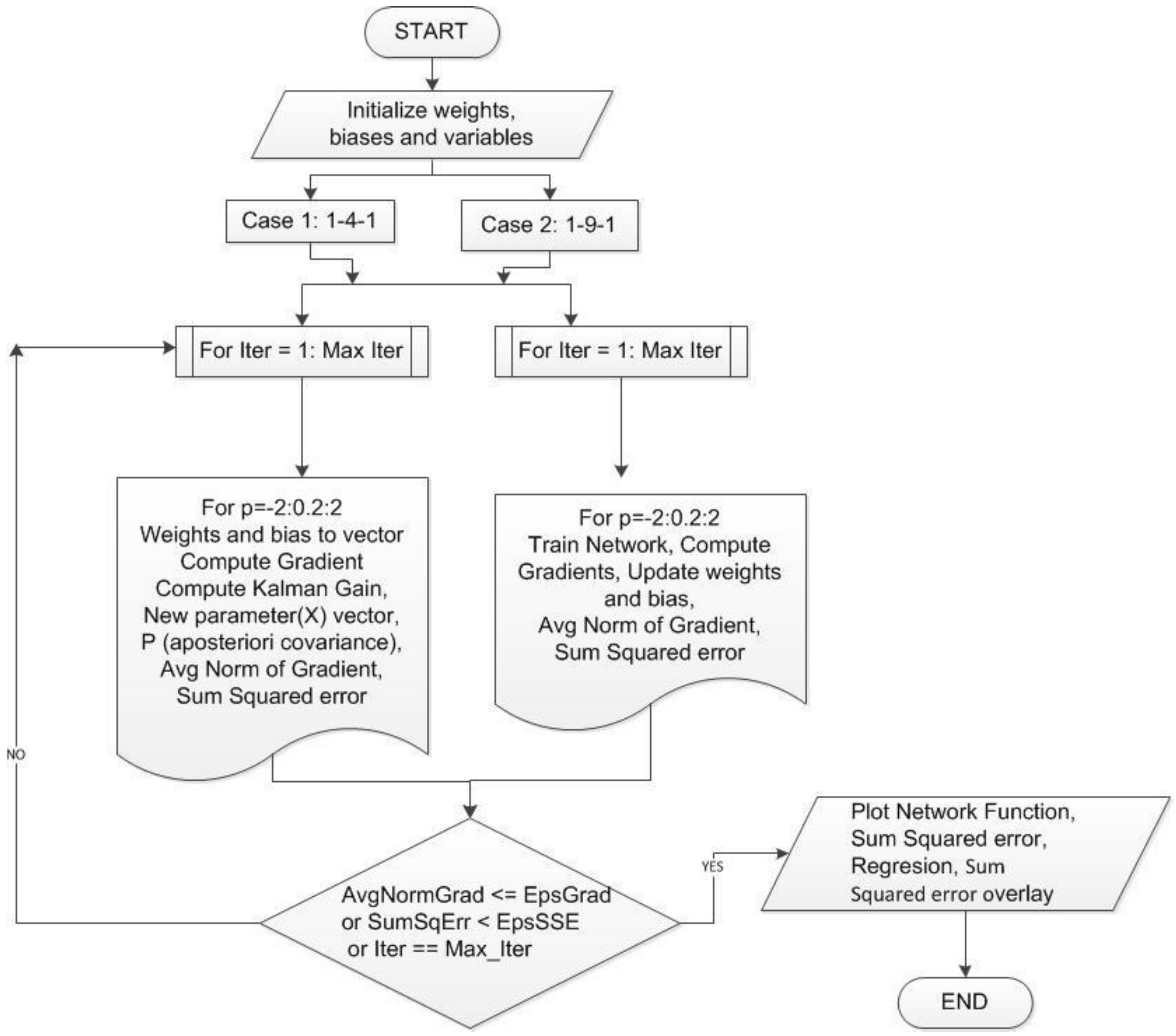
The program randomly initializes the weights and biases first. First the EKF Algorithm is called and it computes the network parameters and creates the plots using the EKF equations. It uses the Gradient matrix (H) that is computed as illustrated in Chapter 11 of the Neural Networks text book. Next the SDBP Algorithm is called and it also computes the network parameters and creates the plots. It uses the Steepest descent with Back propagation equations also from Chapter 11 of the Neural Networks text book.

IN the program, for every iteration, the Average norm of the gradient and sum squared error is calculated for the entire range of points after the training is completed. These values are continuously checked for convergence.

If the Network fails to converge by these two parameters, it is bound to converge by the Max iterations but the results obtained in this way might not be the best compared to those obtained by the first two convergence criteria.

Finally the Sum squared error from both the algorithms is plotted in an overlay plot.

Program Flow Chart



Results

Overall the main goal was to train the same network using EKF and BPSD. There were four test cases with two different Neural Networks and two different weights and biases. The Neural network had a configuration of 1-4-1 and 1-9-1 which means it's a two layer network with 4 and 9 neurons in the hidden layer.

The EKF algorithm converged quickly in less number of iterations compared to the BPSD algorithm.

The plots included in this section are a

- Network Function vs Actual function
- Sumsquared error and validation error over the number of iterations
- The overlay plot of sum squared error for EKF and BPSD
- A regression plot of targets vs output from the Weights and biases obtained from EKF and BPSD Algorithms

Max_Iter is the maximum number of iterations
EpsGrad is the Epsilon convergence factor for the Average Norm of the Gradient
EpsSSE is the Epsilon convergence factor for the Sum Squared Error
SumSq_Err is the Sum Squared Error
SSE is the Sum Squared Error

For the Extended Kalman Algorithm
```
Max_Iter = 8000
EpsGrad = 1*10^-2; EpsSSE = 1*10^-5;
```

For the Back Propagation Algorithm
```
Max_Iter = 60000
EpsGrad = 1*10^-2; EpsSSE = 1*10^-5;
```

Comparison of the Avg Norm of the Gradient and Sum Squared Error

S.No	Iterations	Convergence Criteria	SSE Value	Avg SSE	Avg NormGrad
1-4-1 W_1 B_1 EKF	489	SumSq_Err <= EpsSSE	9.9812e-006	0.0028	2.0065
SDBP	25486	SumSq_Err <= EpsSSE	9.9999e-006	0.0024	0.0167
1-4-1 W_2 B_2 EKF	735	SumSq_Err <= EpsSSE	9.9693e-006	0.0025	2.0630
SDBP	5430	SumSq_Err <= EpsSSE	9.9962e-006	0.0103	0.0129
1-9-1 W_1 B_1 EKF	184	SumSq_Err <= EpsSSE	9.9684e-006	0.0078	2.2397
SDBP	9124	SumSq_Err <= EpsSSE	9.9965e-006	0.0085	0.0176
1-9-1 W_2 B_2 EKF	332	SumSq_Err <= EpsSSE	9.9458e-006	0.0173	2.5258
SDBP	60000	Max Iterations=60000	NA	0.0011	0.0213

Comparison of the Extended Kalman and Back Propagation Algorithms

Network	Kalman Algorithm		Back Propagation	
	Iterations	Avg Sum Sq Err	Iterations	Avg Sum Sq Err
1-4-1 W_1 B_1	489	0.0028	25486	0.0024
1-4-1 W_2 B_2	735	0.0025	5430	0.0103
1-9-1 W_1 B_1	184	0.0078	9124	0.0085
1-9-1 W_2 B_2	332	0.0173	60000	0.0011

Overall Extended Kalman Filter Algorithm converges is less number of iterations than Back Propagation Algorithm. Both Converged for the same criteria of Sum Squared error less than the threshold Epsilon for the Network except for one case where the BPSD converged for max iterations

The Average sum squared error for the Back Propagation was less in case 1 and case 4. But in case 1 EKF converged in 489 iterations vs 25846 for the BPSD. In case 4 EKF converged in 332 iterations vs 60000 for BPSD. **In this regard the Project results compare very well to the results illustrated in the Paper.**

Overall BPSD converged with a less Average Norm of the Gradient than EKF. This could be due to the fact that EKF took very less iterations to reduce the Sum squared error vs BPSD**.** The **Regression** plots of the targets and outputs for EKF and BPSD show that both algorithms are predicting a reliable set of parameters.

The Validation and testing errors were computed to get an understanding of the concepts in Neural Networks training but not used to make any decisions regarding convergence. A sample Validation and test set was also taken from the range of [-2 2].

The validation error was plotted along with the sum squared error for all the test cases. It was observed sometimes the validation error was smaller than the sum squared error but this was not considered in any decisions because it might not be a reliable indicator. The Network being a simple case and the validation error not increasing with reduction in training error, the range of input space being small between [-2 2] etc,. were the reasons.

Detailed results from the Program

Starting Ext Kalman Alg Running Ext Kalman Alg
W1_0 = **[** 0.1110 0.2788 -0.0765 -0.4092 **]** ^{T}
b1_0 = **[** -0.2335 -0.3463 -0.2190 -0.0599 **]** ^{T}
W2_0 = **[** 0.0271 -0.0426 0.3754 0.0181**]**
b2_0 = **[** 0.4436 **]**
Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9812e-006

Results for@1-4-1-Wgt_1;Bias_1-EKF
Iter = 489
AvgNormGrad = 2.0065
Average Sum Squared Error value = 0.0028
Average Validation Error value = 0.0024
Average Testing Sum Sq Error value
Test_Err = 9.0451e-006
Final Weights and biases
W1f = **[** -3.1535 -1.9368 -2.6366 3.8774 **]** ^{T}
b1f = **[** 6.5177 -3.1344 1.3943 0.8056 **]** ^{T}
W2f = **[** 5.4179 1.5190 -2.9496 0.5655 **]**
b2f = **[** -3.4978 **]**

Completed Ext Kalman Alg, Starting Steepest Descent BackPropagation
Running SDBackPropagation
W1_0 = **[** 0.1110 0.2788 -0.0765 -0.4092 **]** ^{T}
b1_0 = **[** -0.2335 -0.3463 -0.2190 -0.0599 **]** ^{T}
W2_0 = **[** 0.0271 -0.0426 0.3754 0.0181 **]**
b2_0 = **[** 0.4436 **]**

Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9999e-006

Results for@1-4-1-Wgt_1;Bias_1-SDBP-Alpha=0.04
Iter = 25486
AvgNormGrad = 0.0167
Average Sum Squared Error value = 0.0024
Average Validation Error value = 0.0019
Average Testing Sum Sq Error value
Test_Err = 1.2762e-005
Final Weights and biases
W1_0 = **[** -3.1634 -1.5360 -1.7175 -1.7743 **]** ^{T}
b1_0 = **[** 6.5536 -2.0204 2.7425 0.8251 **]** ^{T}
W2_0 = **[** 4.9634 2.5649 2.0594 -5.8864 **]**
b2_0 = **[** -3.1013 **]**

Completed Steepest Descent BackPropagation
Completed EKF vs SDBP Run

Starting Ext Kalman Alg
Running Ext Kalman Alg
W1_0 = **[** 0.1377 0.4577 -0.2593 0.1761 **]** ^{T}
b1_0 = **[** -0.2109 0.1718 0.1951 -0.4320 **]** ^{T}
W2_0 = **[** -0.2452 -0.2760 0.1678 0.3444 **]**
b2_0 = **[** -0.1555 **]**

Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9693e-006
SumSq_Err = 9.9693e-006

Results for@1-4-1-Wgt_2;Bias_2-EKF
Iter = 735
AvgNormGrad = 2.0630
Average Sum Squared Error value = 0.0025
Average Validation Error value = 0.0016
Average Testing Sum Sq Error value
Test_Err = 6.4963e-006
Final Weights and biases
W1f = **[** 1.9071 -0.5250 -1.6898 2.8643 **]** ^{T}
b1f = **[** -0.7561 3.0313 -2.3009 -5.9089 **]** ^{T}
W2f = **[** 4.5169 -3.2142 2.2530 -6.4321 **]**
b2f = **[** 1.4346 **]**

Completed Ext Kalman Alg, Starting Steepest Descent BackPropagation
Running SDBackPropagation
W1_0 = **[** 0.1377 0.4577 -0.2593 0.1761 **]** ^{T}
b1_0 = **[** -0.2109 0.1718 0.1951 -0.4320 **]** ^{T}
W2_0 = **[** -0.2452 -0.2760 0.1678 0.3444 **]**
b2_0 = **[** -0.1555 **]**

Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9962e-006
SumSq_Err = 9.9962e-006

Results for@1-4-1-Wgt_2;Bias_2-SDBP-Alpha=0.04
Iter = 5430
AvgNormGrad = 0.0129
Average Sum Squared Error value = 0.0103
Average Validation Error value = 0.0094
Average Testing Sum Sq Error value
Test_Err = 9.3417e-006
Final Weights and biases
W1_0 = **[** 1.4511 2.8490 -1.4885 1.7484 **]** ^{T}
b1_0 = **[** -3.2790 -5.9602 -1.9583 -0.8072 **]** ^{T}
W2_0 = **[** -3.2038 -5.1226 2.6743 5.9175 **]**
b2_0 = **[** -2.0299 **]**

Completed Steepest Descent BackPropagation
Completed EKF vs SDBP Run
Starting Ext Kalman Alg
Running Ext Kalman Alg

W1_0 = **[** 0.2805 0.1753 -0.4933 0.1022 -0.1132 0.4160 -0.4988 -0.0376 -0.0757 **]** ^{T}
b1_0 = **[** -0.0391 0.2702 -0.1775 0.2847 -0.0286 -0.4642 -0.3241 0.2218 -0.0265 **]** ^{T}
W2_0 = **[** -0.3473 -0.1589 0.1074 -0.3083 0.2384 -0.2572 0.4174 -0.2309 0.2655 **]**
b2_0 = **[** -0.3113 **]**

Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9684e-006
SumSq_Err = 9.9684e-006

Results for@1-9-1-Wgt_1;Bias_1-EKF

```
Iter =   184
AvgNormGrad =    2.2397
Average Sum Squared Error value =    0.0078
Average Validation Error value =    0.0080

Average Testing Sum Sq Error value
Test_Err =  6.0158e-006
Final Weights and biases
W1f =  [  -1.9964    2.9479    2.3014   -3.3175   -2.7922    1.2925    0.6180    0.6984    1.1305 ] T
b1f =   [ -2.7069   -6.0940   -1.1200    6.4807   -0.5339   -1.4004   -2.9776    2.0999   -1.3040 ] T
W2f = [    1.2275   -5.4741    3.1164    0.6295   -0.5690   -0.1423    2.6540   -1.8818    0.7301 ]
b2f =   [ 0.1693 ]

Completed Ext Kalman Alg, Starting Steepest Descent BackPropagation
Running SDBackPropagation
W1_0 = [    0.2805    0.1753   -0.4933    0.1022   -0.1132    0.4160   -0.4988   -0.0376   -0.0757 ] T
b1_0 =  [ -0.0391    0.2702   -0.1775    0.2847   -0.0286   -0.4642   -0.3241    0.2218   -0.0265 ] T
W2_0 = [  -0.3473   -0.1589    0.1074   -0.3083    0.2384   -0.2572    0.4174   -0.2309    0.2655 ]
b2_0 =  [ -0.3113 ]

Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9965e-006
SumSq_Err =  9.9965e-006
```

Results for@1-9-1-Wgt_1;Bias_1-SDBP-Alpha=0.04

```
Iter =        9124
AvgNormGrad =    0.0176
Average Sum Squared Error value =    0.0085
Average Validation Error value =    0.0075
Average Testing Sum Sq Error value
Test_Err =  6.8095e-006
Final Weights and biases
W1_0 =  [  -0.4829   -2.0660   -1.5382   -2.5287   -0.8495    0.2842   -0.3976   -2.0062    1.5869 ] T
b1_0 =   [ -0.6023    4.5190    0.8179    5.3859    2.0691   -0.6593   -0.3709    4.4046    1.9024 ] T
W2_0 = [    0.8140    2.7253   -7.5006    3.4791    0.9257   -0.5555    0.9254    2.6303   -2.3958 ]
b2_0 =  [ -2.7685 ]

Completed Steepest Descent BackPropagation
Completed EKF vs SDBP Run
Starting Ext Kalman Alg
Running Ext Kalman Alg
W1_0 =   [  -0.2125   -0.4089    0.0762    0.1834    0.0466   -0.0743    0.1444    0.1476    0.1790 ] T
b1_0 =    [ 0.1358    0.4452   -0.2911    0.2093   -0.2638   -0.3806    0.1073   -0.0499   -0.0413 ] T
W2_0 =  [  0.1619    0.2703   -0.1498    0.1620   -0.0838    0.3419    0.3329   -0.2436    0.1135]
b2_0 =   [ 0.0822 ]

Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=9.9458e-006
SumSq_Err =  9.9458e-006
```

Results for@1-9-1-Wgt_2;Bias_2-EKF

```
Iter =   332
AvgNormGrad =    2.5258
Average Sum Squared Error value =    0.0173
```

Average Validation Error value = 0.0157
Average Testing Sum Sq Error value
Test_Err = 7.6300e-006
Final Weights and biases
W1f = **[** -1.4269 -3.3774 -2.9763 0.4666 0.8995 2.6871 -2.0534 0.1295 0.4133 **]** ^{T}
b1f = **[** -1.6393 2.7341 6.2836 -2.4040 1.3439 -5.3423 0.3825 1.3808 -0.6840 **]** ^{T}
W2f = **[** 1.9701 -0.6782 5.3556 2.1637 -1.0801 -0.7365 -4.0724 -2.2020 -0.0586 **]**
b2f = **[**-0.1475 **]**

Completed Ext Kalman Alg, Starting Steepest Descent BackPropagation
Running SDBackPropagation
W1_0 = **[** -0.2125 -0.4089 0.0762 0.1834 0.0466 -0.0743 0.1444 0.1476 0.1790 **]** ^{T}
b1_0 = **[** 0.1358 0.4452 -0.2911 0.2093 -0.2638 -0.3806 0.1073 -0.0499 -0.0413 **]** ^{T}
W2_0 = **[** 0.1619 0.2703 -0.1498 0.1620 -0.0838 0.3419 0.3329 -0.2436 0.1135
b2_0 = **[** 0.0822

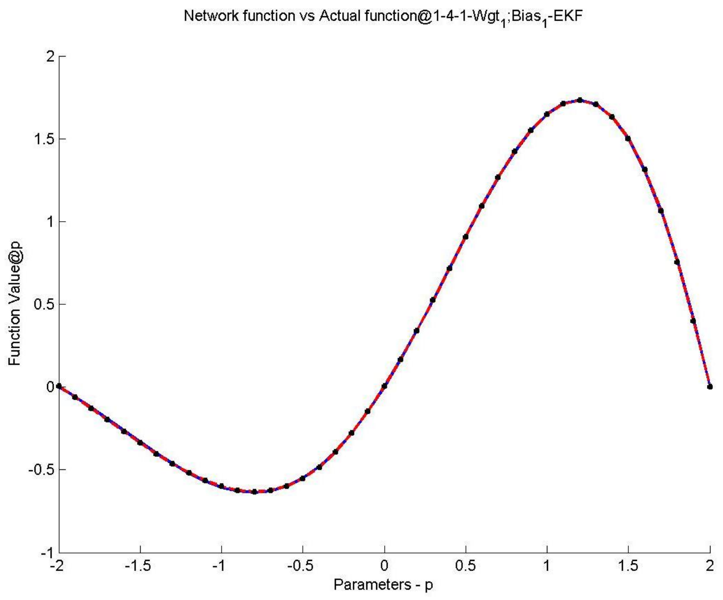
Solution Converged - Max Iterations-60000

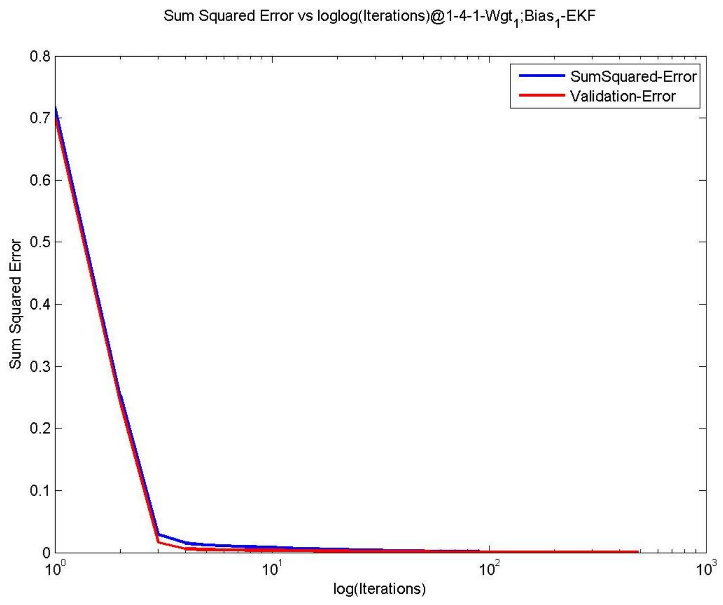
<u>Results for@1-9-1-Wgt_2;Bias_2-SDBP-Alpha=0.04</u>
Iter = 60000
AvgNormGrad = 0.0213
Average Sum Squared Error value = 0.0011
Average Validation Error value = 0.0010
Average Testing Sum Sq Error value
Test_Err = 1.6589e-005
Final Weights and biases
W1_0 = **[** -0.4763 -1.3394 -0.6704 2.9325 -0.6100 -1.6436 1.7485 -0.6201 0.3271 **]** ^{T}
b1_0 = **[**-0.2066 2.6557 -0.7718 -6.1136 -0.7213 -1.9533 -0.7296 -0.6310 -0.2504 **]** ^{T}
W2_0 = **[** 0.6528 1.2315 0.4584 -5.5355 0.4018 1.5196 6.3416 0.4513 -2.1991**]**
b2_0 = **[**-3.1560**]**

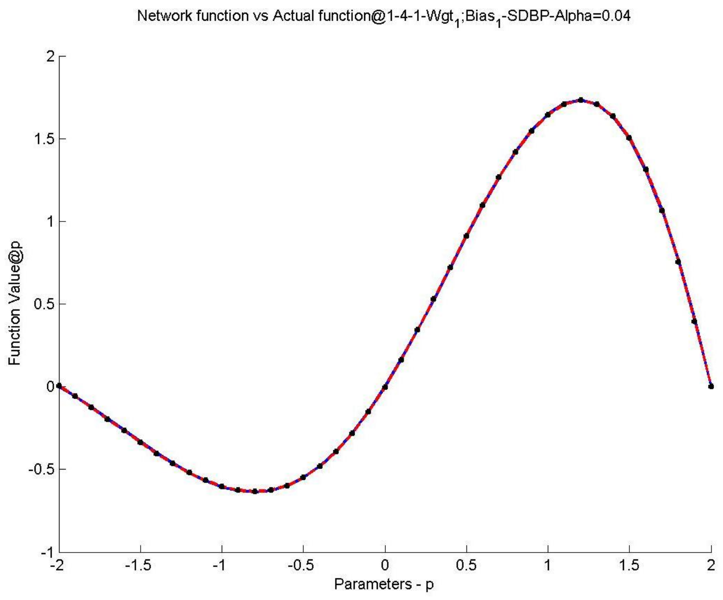
Completed Steepest Descent BackPropagation
Completed EKF vs SDBP Run

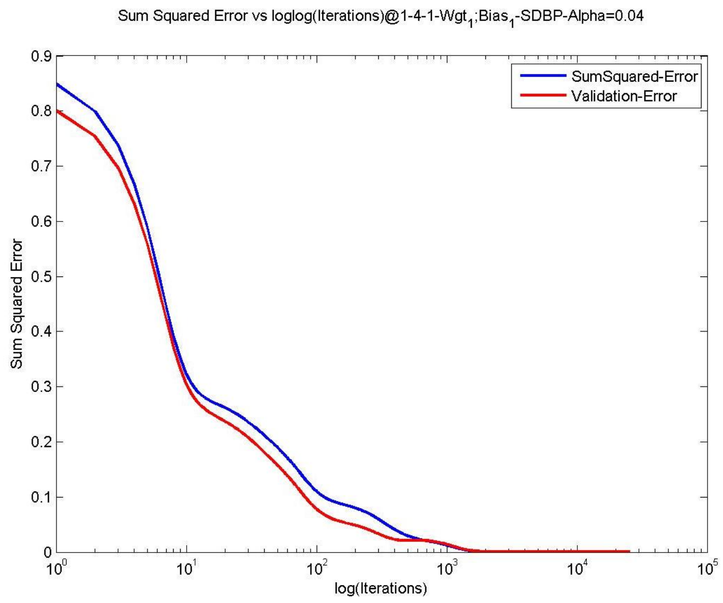
Plots: Network Function vs Actual Function, Sum Squared Error, Regression for EKF and BPSD

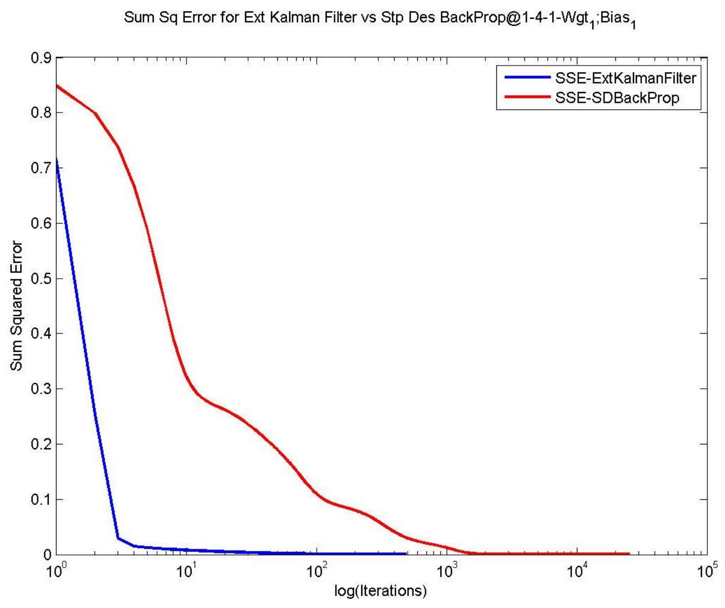
Case 1 @1-4-1 Weight_1 and Bias_1



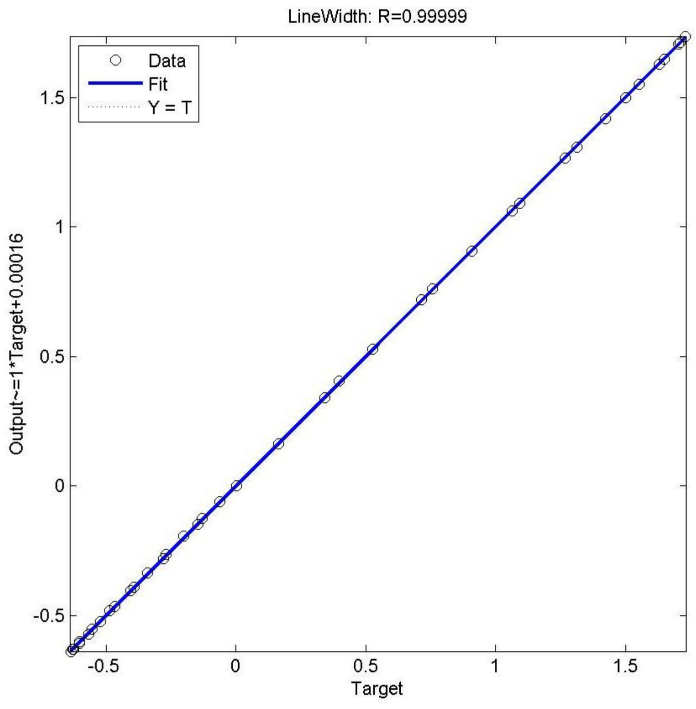




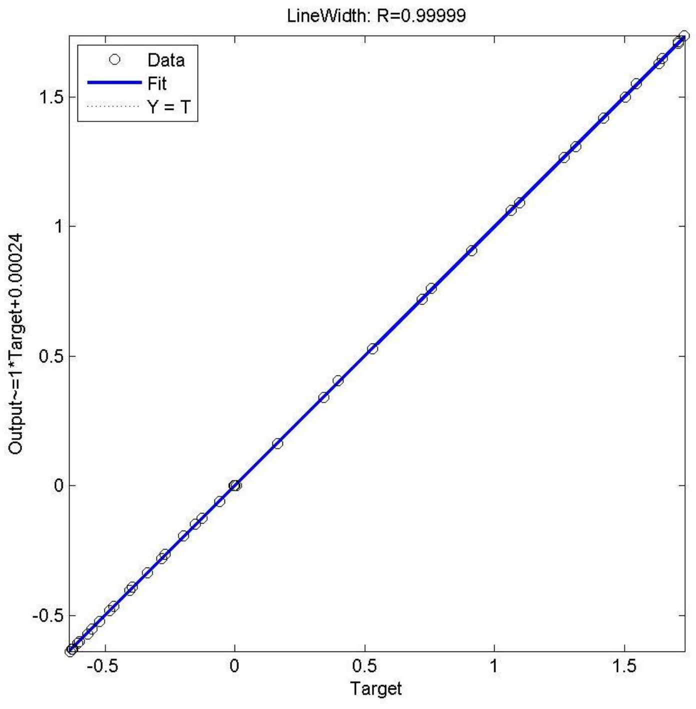




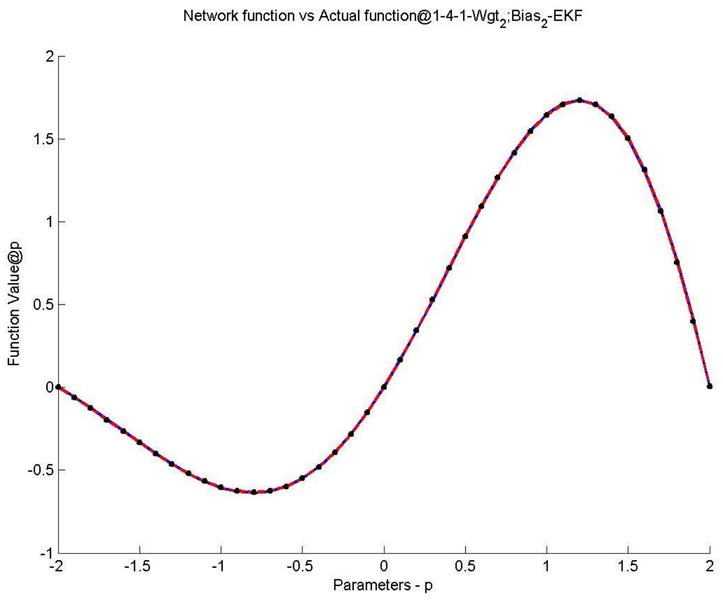
Regression@1-4-1-Wgt_1;Bias_1-EKF

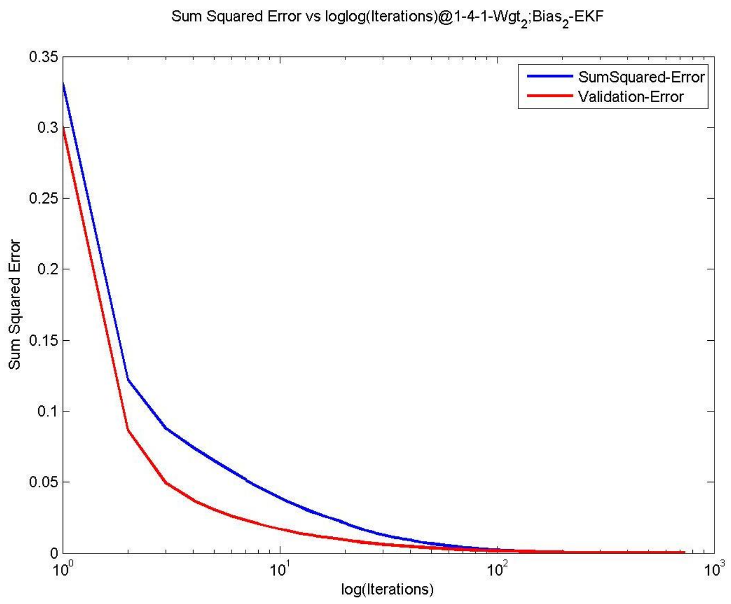


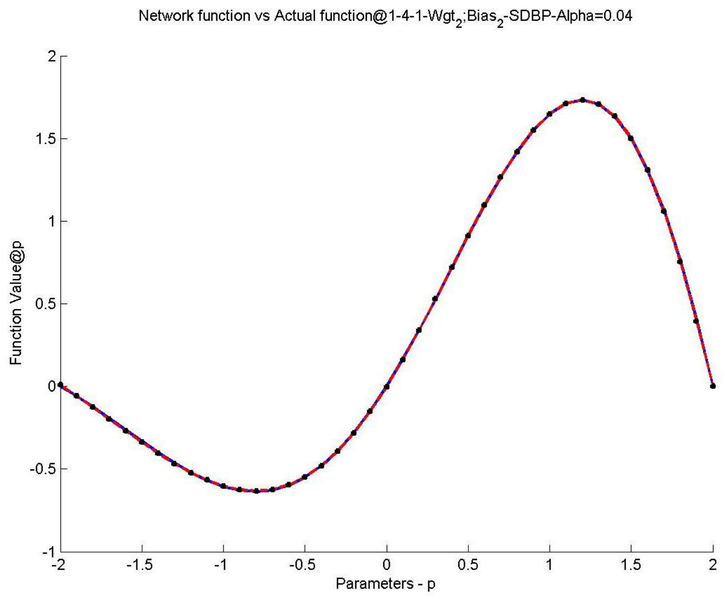
Regression@1-4-1-Wgt_1;Bias_1-SDBP-Alpha=0.04

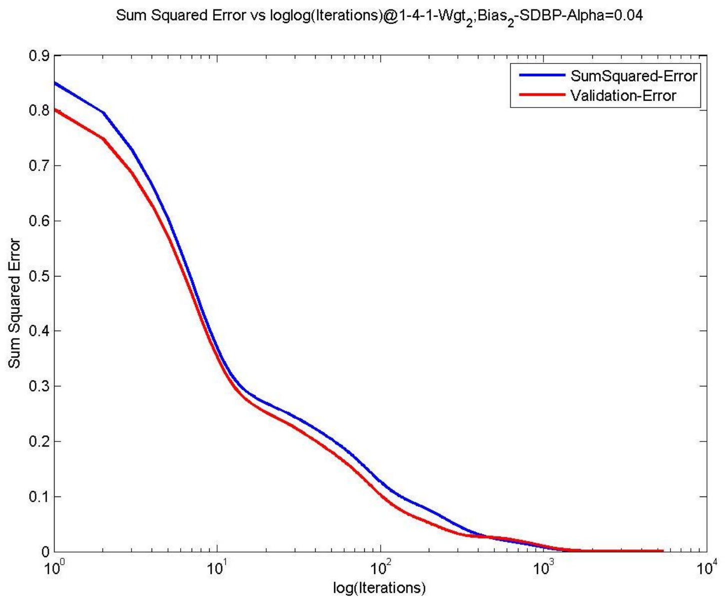


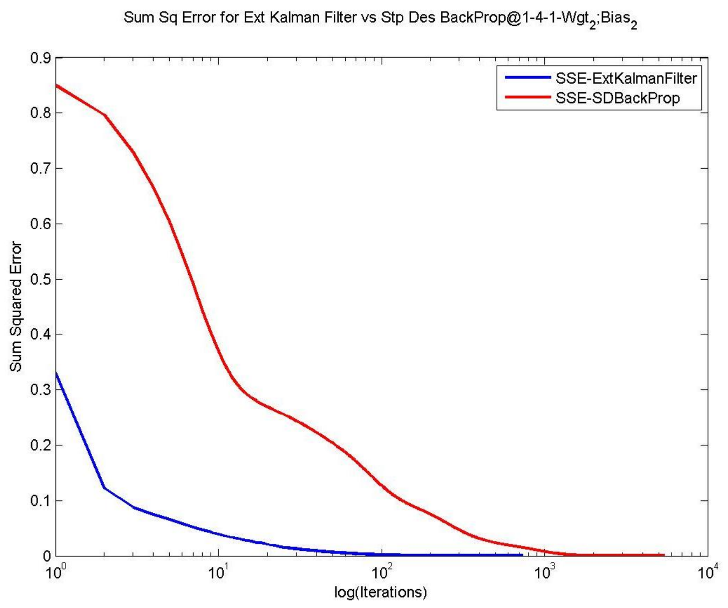
Case 2 @1-4-1 Weight_2 and Bias_2



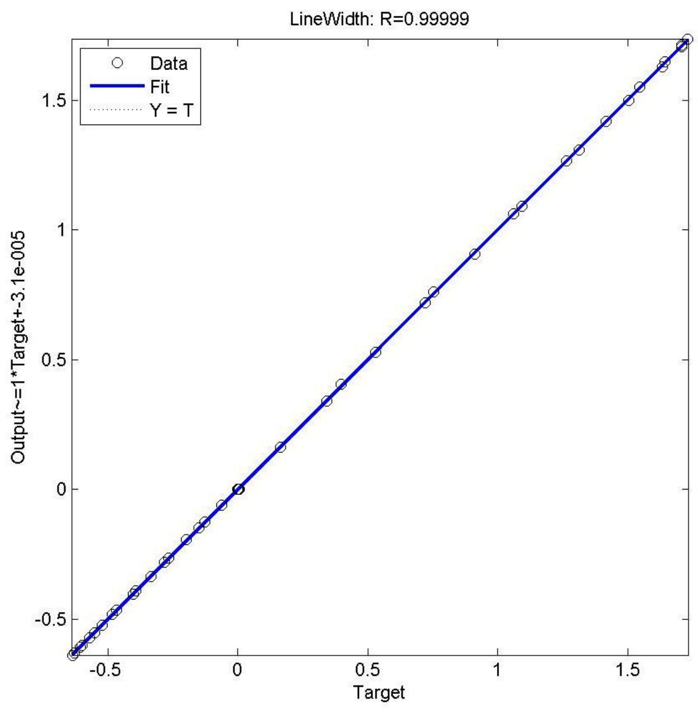




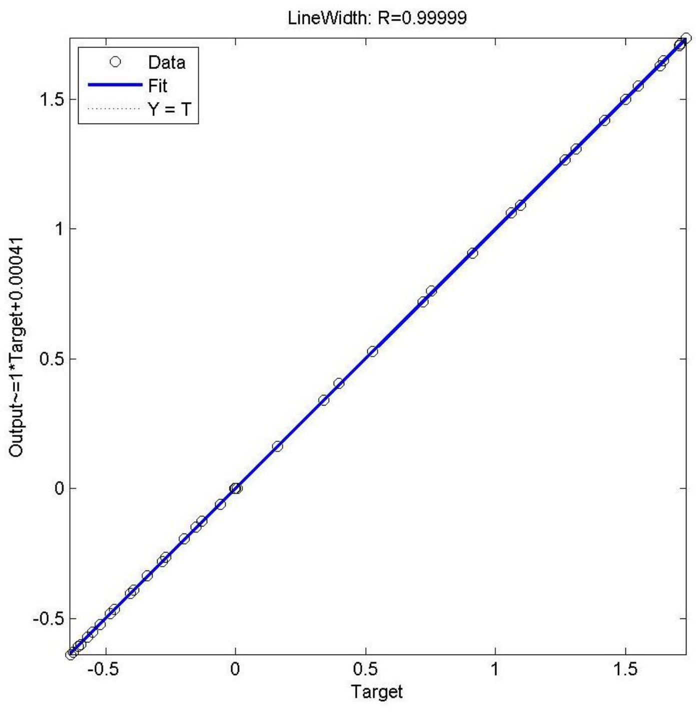




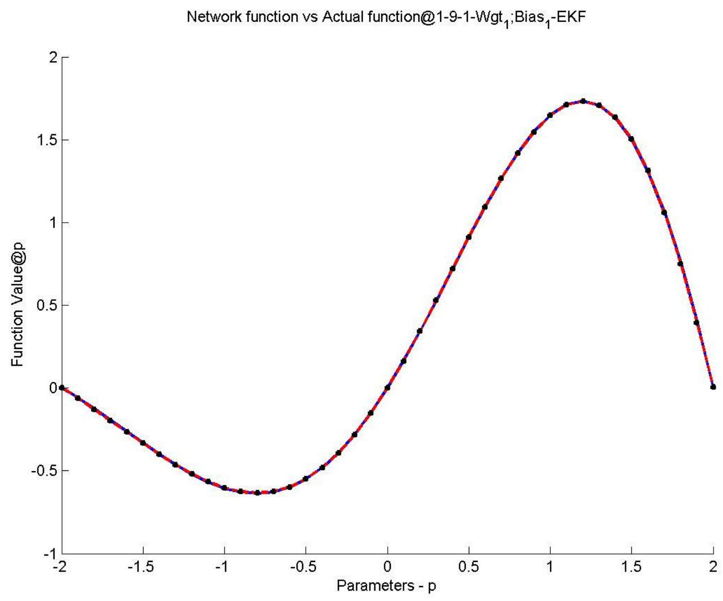
Regression@1-4-1-Wgt 2;Bias 2-EKF

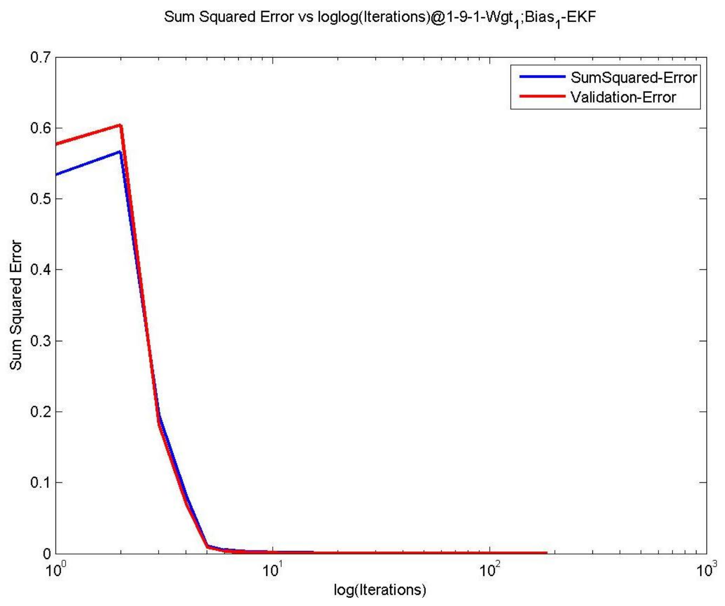


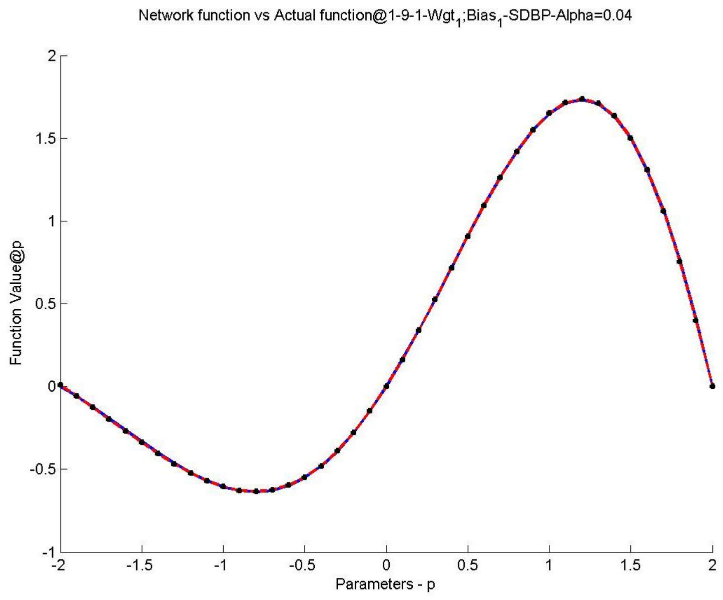
Regression@1-4-1-Wgt 2;Bias 2-SDBP-Alpha=0.04

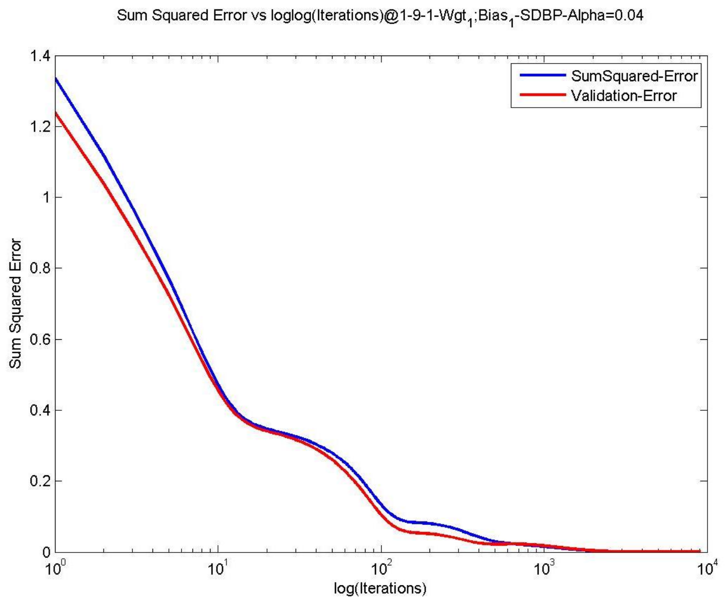


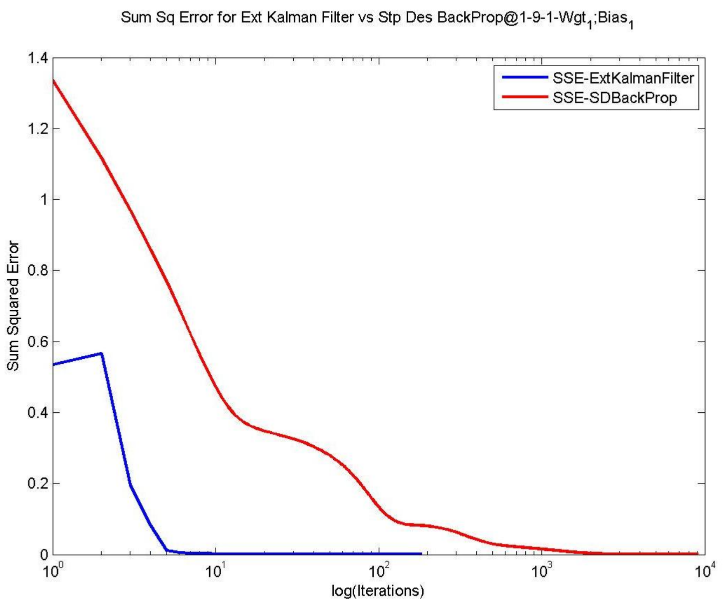
Case 3 @1-9-1 Weight_1 and Bias_1



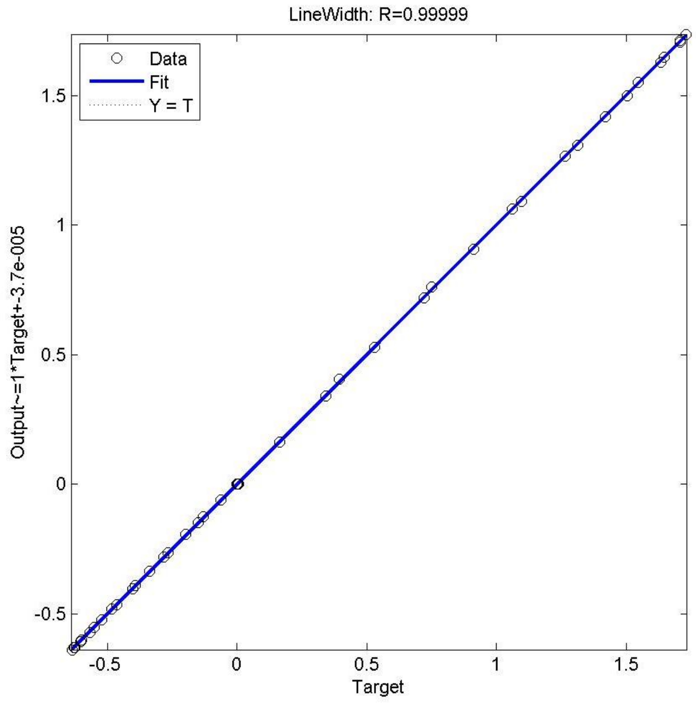




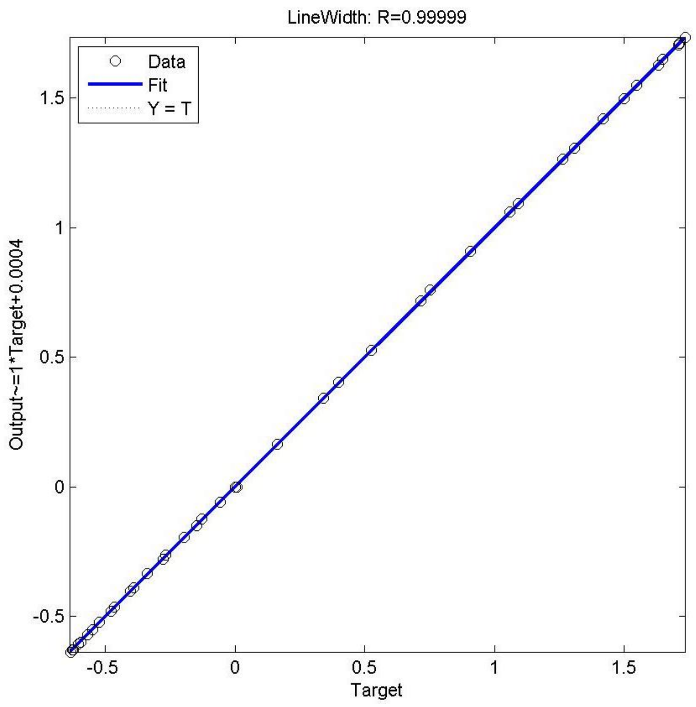




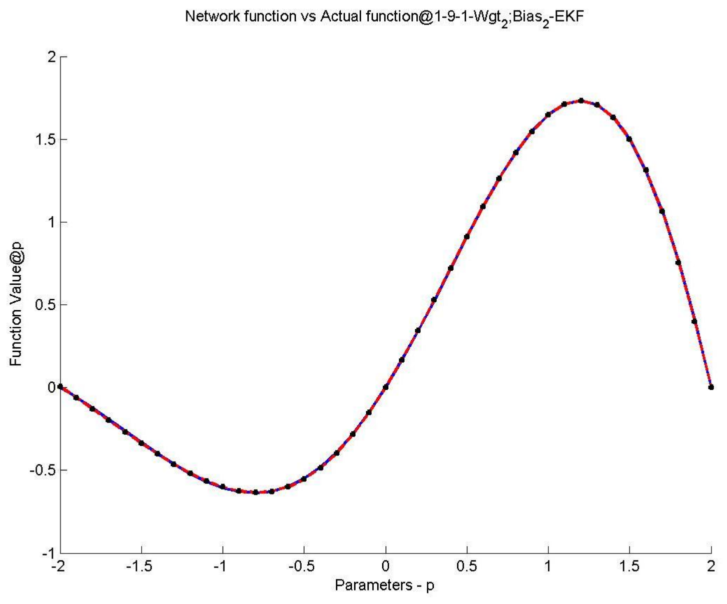
Regression@1-9-1-Wgt 1;Bias 1-EKF

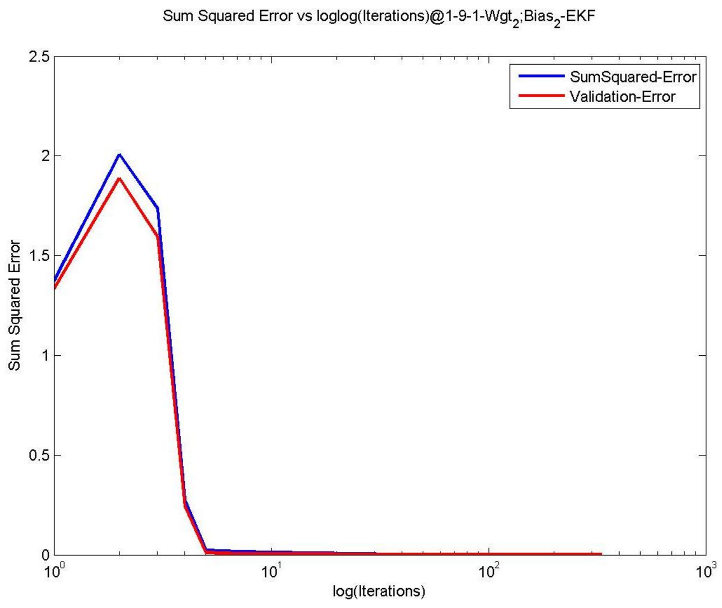


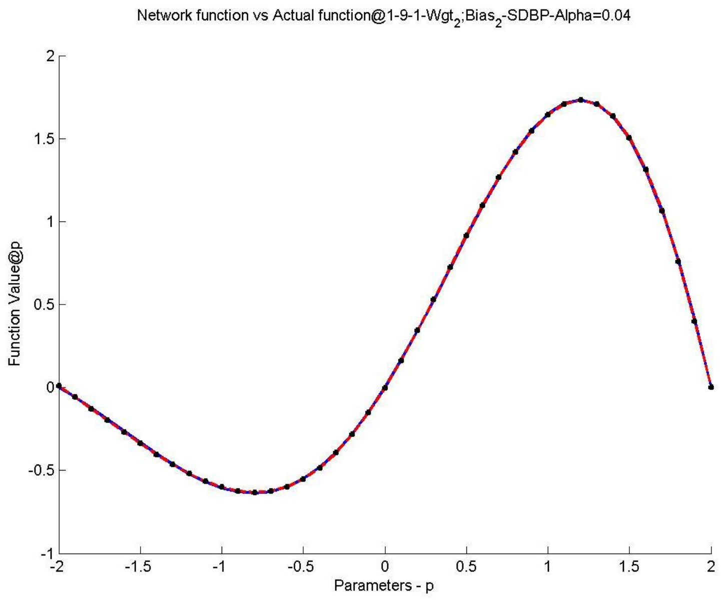
Regression@1-9-1-Wgt 1;Bias 1-SDBP-Alpha=0.04

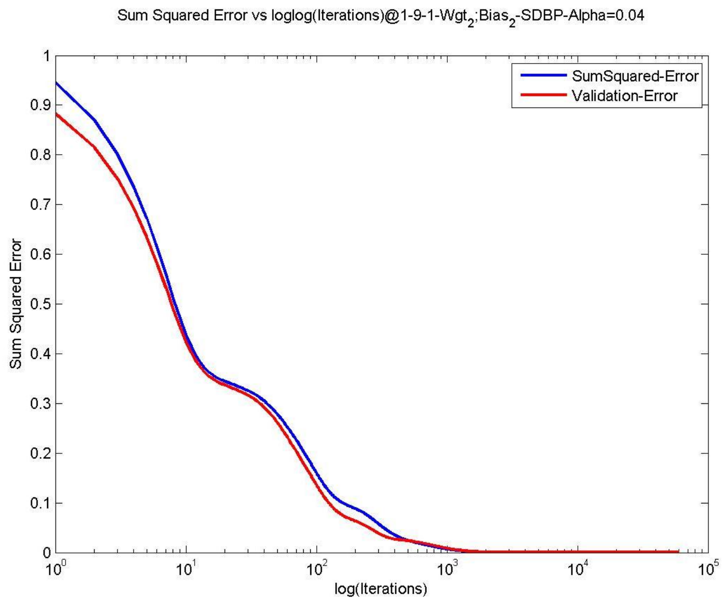


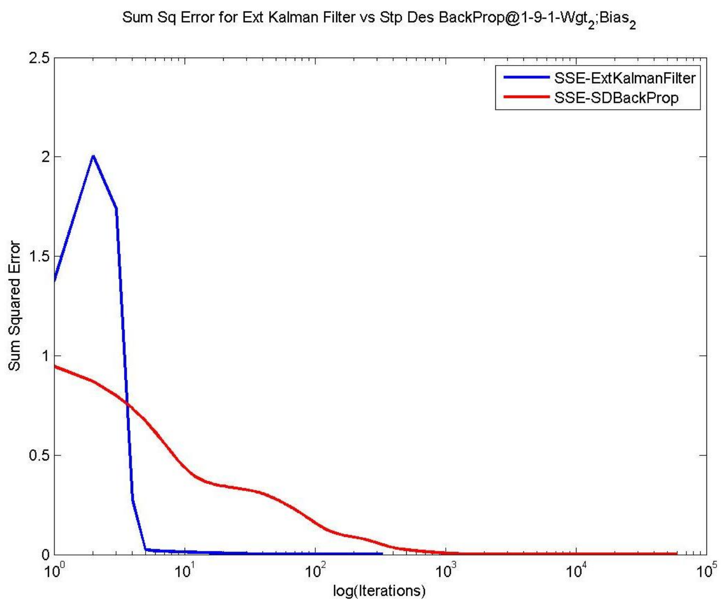
Case 4 @1-9-1 Weight_2 and Bias_2



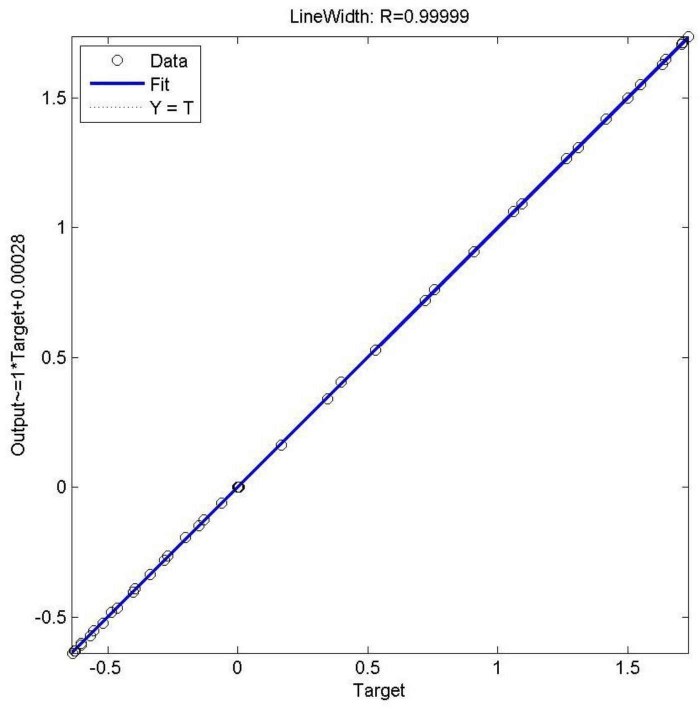




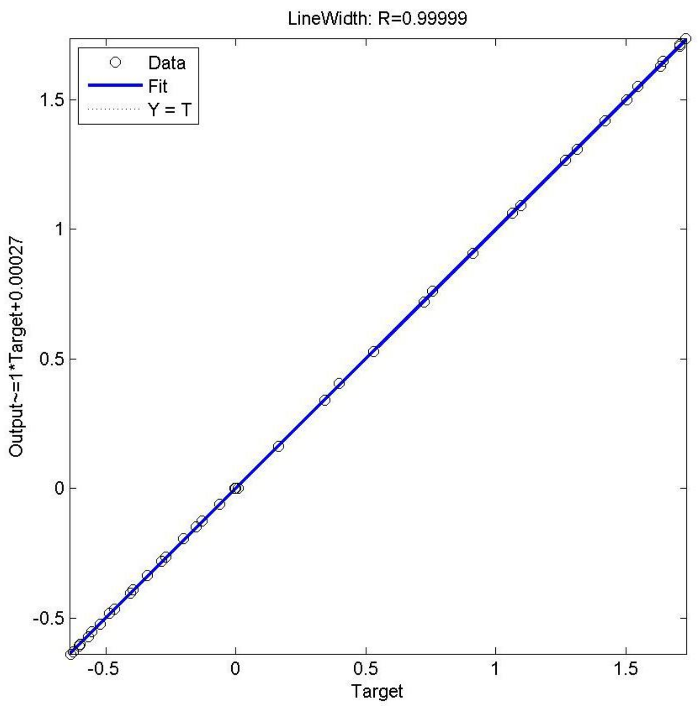




Regression@1-9-1-Wgt_2;Bias_2-EKF



Regression@1-9-1-Wgt_2;Bias_2-SDBP-Alpha=0.04



Conclusions

Advantages and Disadvantages

This is not a complex problem that tests the limits of the BPSD algorithm but definitely gives an understanding of the best among the two in terms of quickness and accuracy. The BPSD Algorithm does not follow the target curve properly for smaller number of neurons unless it's run for enough iteration's. EKF Algorithm overall converges quickly and the Network function follows the input function better.

BPSD converges with a smaller Average Norm of the Gradient than EKF. It is guaranteed to converge and it is computationally expensive in the sense that it solves a smaller problem a lot of times as it takes more iteration's. Because of this, it is better to monitor the validation error as there is a possibility of over fitting.

EKF converges quicker with less Neurons but it is computationally expensive as it involves matrix multiplication for all three updates of the parameters, Kalman Gain, estimated covariance (P), and inverse while computing the Kalman gain. When the complexity of the problem increases in terms of size i.e., more layers, more neurons more parameters etc, these matrices will grow bigger and bigger and efficient matrix inversion operations like Cholesky factorization have to be employed. If the speed of convergence is the primary criteria EKF is a good option over BPSD.

The detailed comparisons are given in the following tables in the Results section.

Comparison of the Extended Kalman and Back Propagation Algorithms
Comparison of the Avg Norm of the Gradient and Sum Squared Error

Ways to improve the Network

The Validation and test set can be selected in a better way to represent the input sine function better.

Plotting the error surface for smaller networks can give some insights.

Determining the Estimated number of parameters for the network's so that a better network size can be used for this problem size and scope.

Appendix

Abbreviations

EKF – Extended Kalman Filter

BPSD – Back Propagation with Steepest Descent

Max_Iter – maximum number of iterations

EpsGrad – Epsilon convergence factor for the Average Norm of the Gradient

EpsSSE – Epsilon convergence factor for the Sum Squared Error

SumSq_Err – Sum Squared Error

SSE – Sum Squared Error

References

1. Sharad Singhal and Lance Wu, "Training Feed-Forward Networks with the Extended Kalman Algorithm"
2. Martin T. Hagan, Howard B. Demuth and Mark Beale, "Neural Network Design"

Matlab Programs

RunProject.m

```
clc;clear;pwd
Alpha = 0.04;
W1_0=[]; b1_0=[]; W2_0=[]; b2_0=[]; S1=0; S2=0;
%Taking a set of 10 points for Validation and Testing
% Validation = [-1.95 -1.23 -0.99 -0.51 -0.03  0.45  0.93  1.05  1.65  1.85];
% Testing    = [-1.83 -1.49 -0.78 -0.27 -0.13  0.21  0.55  1.23  1.57  1.96];
Validation = [-1.99 -1.23 -0.99 -0.03  0.45  1.65  1.95];
Testing    = [-1.97 -1.49 -0.27 -0.13  0.21  1.57  1.97];

for idx_1 = 1:2

    if(idx_1 == 1)
        caseName_1 = '@1-4-1';
        S1 = 4; % Neurons in first layer
        S2 = 1; % Neurons in second layer
    else
        caseName_1 = '@1-9-1';
        S1 = 9; % Neurons in first layer
        S2 = 1; % Neurons in second layer
    end

    for idx_2 = 1:2
        if(idx_2 ==1)
            caseName_2 = strcat(caseName_1 , '-Wgt_1;Bias_1');
            W1_0 = rand (S1, 1)-0.5;
            b1_0 = rand (S1, 1)-0.5;
            W2_0 = rand (S2, S1)-0.5;
            b2_0 = rand (S2, 1)-0.5;
        else
            caseName_2 = strcat(caseName_1 , '-Wgt_2;Bias_2');
            W1_0 = rand (S1, 1)-0.5;
            b1_0 = rand (S1, 1)-0.5;
            W2_0 = rand (S2, S1)-0.5;
            b2_0 = rand (S2, 1)-0.5;
        end
        disp('Starting Ext Kalman Alg')
        % W1_0  b1_0 W2_0  b2_0
        Iter = 0;
        caseName_3 = strcat(caseName_2 , '-EKF');
        [SSE_k, Iter] = ExtKalmanFilter(W1_0 , b1_0, W2_0, b2_0, S1, S2, caseName_3,
Validation, Testing);

        disp('Completed Ext Kalman Alg, Starting Steepest Descent BackPropagation')
        caseName_3 = strcat(caseName_2 , strcat('-SDBP-Alpha=',num2str(Alpha)));
        [SSE_b, Iter] = SDBackPropagation(W1_0 , b1_0, W2_0, b2_0, S1, S2, Alpha,
caseName_3, Validation, Testing);
        disp('Completed Steepest Descent BackPropagation')

        FileNameSSE = strcat(caseName_2 , '-SSE_Overlay.jpg');
        Titl = 'Sum Sq Error for Ext Kalman Filter vs Stp Des BackProp';
        %xlbl = 'log(Iterations)';
        %ylbl = 'Sum Squared Error';
        plt1 = 'SSE-ExtKalmanFilter';
        plt2 = 'SSE-SDBackProp';
        PlotSSE(SSE_k, SSE_b, FileNameSSE, caseName_2, Titl, plt1, plt2 );

        disp('Completed EKF vs SDBP Comparison Run')
    end %for idx_2 = 1:2
end %for idx_1 = 1:2
```

sineFunction.m

```
function Y = sineFunction(p)

    Y = exp(p/2).*sin(pi*p/2);

end
```

ExtKalmanFilter.m

```
function [SSE_k, Iter] = ExtKalmanFilter(W1_0 , b1_0, W2_0, b2_0, S1, S2, caseName_3,
Validation, Testing)
    %From the paper - Training Feed Forward Networks with the Extended Kalman Algorithm
    %ExtKalmanFilter implementation to optimize the weights and bias of a 1-N-1 network

    disp('Running Ext Kalman Algorithm')
    W1_0
    b1_0
    W2_0
    b2_0
    fh_WgtVectorToArray = @(xWgtBias, S1, S2)WgtVectorToArray(xWgtBias, S1, S2);
    fh_ComputeGradNtwkOutput = @(W1_0, b1_0, W2_0, b2_0, a_0, S1)ComputeGradNtwkOutput(W1_0,
b1_0, W2_0, b2_0, a_0, S1);
```

```
    caseName_2 = ' Extended Kalman Filter';
    p = -2:0.2:2;
    TrainPtsSz = size(p); Max_TrainPts = TrainPtsSz(2);
    SumSq_Err = 0; SumSq_ErrMat = [];
    %Stopping Criteria
    EpsGrad = 1*10^-2; EpsSSE = 1*10^-5;
    Gradient = 0; Norm_Grad = 0; CumuNormGrad = 0; AvgNormGrad = 0;
    %-------------
    Max_Iter = 8000; nSweep = 0; Const = 1; x_k0 = []; P_k0 = [];
    SumSq_ErrMat = []; Valid_ErrMat = [];
    %----------------------
    Status = 'First_Iter'; Iter = 0;
    for Iter = 1:Max_Iter

        Cum_Err = 0; Gradient = 0; Norm_Grad = 0; CumuNormGrad = 0; AvgNormGrad = 0;
        nSweep = mod(Iter,20); % For every 20 sweeps update R_k
        if(nSweep == 0)
            Const = exp(-1*Iter/50.0);
        end
        for p = -2:0.2:2

            if(strcmp(Status, 'First_Iter'))
                x_k0 = [W1_0; b1_0; W2_0'; b2_0 ]; %This is for any 1-N-1 network
            else
                [W1_0, b1_0, W2_0, b2_0] = fh_WgtVectorToArray(x_k0, S1, S2);
            end
            [H_k, a_2] = fh_ComputeGradNtwkOutput(W1_0, b1_0, W2_0, b2_0, p, S1); %Gradient
Matrix
            [m n] = size(H_k);
            if(strcmp(Status, 'First_Iter'))
                P_k0  = diag(500*ones(1,m)); %Final Covariance
            end

            R_k     = 1;                        % Sigma in x[w1 w2..] Std dev in measurements
            R_k     = R_k*Const;                % Windowing - smooths out the Network fun
quicker
            d_k     = sineFunction(p); %exp(p/2).*sin(pi*p/2);   % target a_2 + V_k %
Network output + noise
            d_k_hat = a_2;                      % Network output

            %Optimal Kalman Gain
            K_k = (P_k0*H_k) / ((H_k'*P_k0*H_k)+R_k);
            x_k1 = x_k0 + (K_k*(d_k - d_k_hat));
            % Updated estimate a posteriori covariance
            P_k1 = P_k0 - (K_k*H_k'*P_k0); %P_k * ( I - K_k * H_k')

            x_k0 = x_k1;
            P_k0 = P_k1;
            %-----------
            Norm_Grad = norm(H_k);
            CumuNormGrad = CumuNormGrad + Norm_Grad;
            Status = 'After_First_Iter';

        end % for p=-2:0.2:2
        %--------------------
        AvgNormGrad = (CumuNormGrad/Max_TrainPts);
        [W1_cse, b1_cse, W2_cse, b2_cse] = fh_WgtVectorToArray(x_k0, S1, S2);%
WgtVectorToArray(x_k0, S1, S2);
        pp = -2:0.2:2;
        CumuSumSqErrr = GetCumuSumSqErr(W2_cse, b2_cse, W1_cse, b1_cse, pp);
        %Avg Sum Sq error
        SumSq_Err = (CumuSumSqErrr/Max_TrainPts);
        if(Iter == 1)
            SumSq_ErrMat = [SumSq_Err];
        else
            SumSq_ErrMat = [SumSq_ErrMat SumSq_Err];
        end
        %----------------------
        ValidErr = 0;
        CumuValidErr = GetCumuSumSqErr(W2_cse, b2_cse, W1_cse, b1_cse, Validation);
        %Avg Validation error
        ValidErr = (CumuValidErr/length(Validation));
        if(Iter == 1)
            Valid_ErrMat = [ValidErr];
        else
            Valid_ErrMat = [Valid_ErrMat ValidErr];
        end
        %-----------------------
        %Check for convergence
        if(AvgNormGrad <= EpsGrad)
            disp(strcat('Solution Converged - AvgNormGrad <= EpsGrad, AvgNormGrad=',
num2str(AvgNormGrad)))
            AvgNormGrad
            break;
        end
        if(SumSq_Err <= EpsSSE)
            disp(strcat('Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=',
num2str(SumSq_Err)))
            SumSq_Err
            break;
        end
    end %for Iter = 1:Max_Iter
    %-------------------------
    if(Iter == Max_Iter)
        disp(strcat('EKF Solution Converged - Max Iterations= ', int2str(Iter)))
    end

    SSE_k = SumSq_ErrMat;
    FileNameSSE = strcat(caseName_3, '-SSE.jpg');
    FileNameNAF = strcat(caseName_3, '-NAF.jpg');
    disp(strcat('Results for ', caseName_3));
    Titl = 'Sum Squared Error vs loglog(Iterations)';
    plt1 = 'SumSquared-Error';
    plt2 = 'Validation-Error';
    PlotSSE(SumSq_ErrMat, Valid_ErrMat, FileNameSSE, caseName_3, Titl, plt1, plt2 );
    Iter
    AvgNormGrad
    disp('Average Sum Squared Error value')
    mean(SumSq_ErrMat)
    disp('Average Validation Error value')
    mean(Valid_ErrMat)


    [W1f, b1f, W2f, b2f] = fh_WgtVectorToArray(x_k0, S1, S2); %WgtVectorToArray(x_k0, S1,
S2)
```

```
    %Plot Network function vs Actual Function with final weights and biases
    PlotNetworkFunction(W2f, b2f, W1f, b1f, caseName_3, FileNameNAF);
    %-------------------------------
    disp('Average Testing Sum Sq Error value')
    CumuTestErr = GetCumuSumSqErr(W2f, b2f, W1f, b1f, Testing);
    Test_Err = (CumuTestErr/length(Testing)) %only 10 points

    disp(strcat('Final Weights and biases'))
    [W1f, b1f, W2f, b2f] = fh_WgtVectorToArray(x_k0, S1, S2)

end %function EKT


function [W1, b1, W2, b2] = WgtVectorToArray(xWgtBias, S1, S2)
    %This function is good only for a 1-N-1 layer network
    W1 = xWgtBias(1:S1);
    b1 = xWgtBias(S1+1: (S1*2) );
    W2 = xWgtBias((S1*2)+1 : (S1*3))';
    b2 = xWgtBias((S1*3)+1: end );
end

function [H, a_2] = ComputeGradNtwkOutput(W1_0, b1_0, W2_0, b2_0, a_0, S1)
% - calc Grad and output for each individual 'p' --using this now
% 1-N-1 Network; hidden layer - logsig, output layer - purelin

    a_1 = logsig (W1_0 * a_0 + b1_0);
    a_2 = purelin(W2_0 * a_1 + b2_0);

    %Backpropagation
    s_2 = 1;   %s_2 = 1 only for purelin function in 2nd layer
    OnesMat = ones(S1, 1);
    s_1 = (OnesMat-a_1).*a_1.* W2_0'* s_2; % s_m = F_m(n_m)*(W_m+1)'*s_m+1
    %For Tansig
    %s_1 = (1-(a_1.*a_1)).* W2_0'* s_2

    Wgt__L2_Grad = (s_2 * a_1')'; Bias_L2_Grad = s_2;
    Wgt__L1_Grad = (s_1 * a_0');  Bias_L1_Grad = s_1;

    H = [Wgt__L1_Grad; Bias_L1_Grad; Wgt__L2_Grad; Bias_L2_Grad ];

end

%--------------------------------
% Kalman Filter
% %Predictor - apriori state est
% x_k+1_k = Phi_k+1_k * x_k_k + Psi_k+1_k*u_k
% %Predictor apriori estimated covariance
% p_k+1_k = Phi_k+1_k *p_k_k * Phi_k+1_k' + Tau_k+1_k * Phi_k * Tau_k+1_k'
% %Optimal Kalman Gain
% k_k+1 = Phi_k+1_k *H_k+1' * Inv( H_k+1' * p_k+1_k *H_k+1  + R_k+1 )
% %y_k - innovation or measurement residual
% x_k+1_k+1 = x_k+1_k + K_k+1 * ( Z_k+1 - H_k+1 * x_k+1_k)
% % Updated estimate a posteriori covariance
% P_k+1_k+1 = P_k+1_k ( I - K_k+1 * H_k+1)
%--------------------------------
```

SDBackPropagation.m

```
function [SSE_b, Iter] = SDBackPropagation(W1_0 , b1_0, W2_0, b2_0, S1, S2, Alpha,
caseName_3, Validation, Testing)

    disp('Running SDBackPropagation')
    W1_0
    b1_0
    W2_0
    b2_0
    %Works for any number of neurons in hidden layer in a 2 layer network
    p=-2:.2:2; %p(1)...p(Max_train)
    TrainPtsSz  = size(p);  Max_TrainPts = TrainPtsSz(2);
    SumSq_Err = 0;  SumSq_ErrMat = []; Valid_ErrMat = [];
    %Stopping Criteria
    Max_Iter = 60000; Iter = 0; EpsGrad = 1*10^-2; EpsSSE = 1*10^-5; %Eps = 5*10^-2;

    Gradient = 0; Norm_Grad = 0; CumuNormGrad = 0; AvgNormGrad = 0;
    caseName_1= ''; caseName_2= ''; %caseName_3= 'SDBP';

    for Iter = 1:Max_Iter
        Cum_Err = 0;
        Gradient = 0; Norm_Grad = 0; CumuNormGrad = 0; AvgNormGrad = 0;
        for p = -2:0.2:2
            %Forward Propagation
            a_0 = p;
            a_1 = logsig (W1_0 * a_0 + b1_0);
            a_2 = purelin(W2_0 * a_1 + b2_0);
            a = a_2;

            t = sineFunction(a_0);
            e = t - a;
            Cum_Err = Cum_Err + e;
            %Backpropagation
            s_2 = -2 * 1 * e;   %s_M = -2*F(n)(t-a)
            OnesMat = ones(S1, 1); %Neu_Lyr_1
            s_1 = (OnesMat-a_1).*a_1.* W2_0'* s_2;% s_m = F_m(n_m)*(W_m+1)'
            %Weight Update Final Layer
            W2_1 = W2_0 - (Alpha * s_2 * a_1'); %W_m(k+1) = W_m(k) - Alpha * s_m * (a_m-1)'
            b2_1 = b2_0 - (Alpha * s_2);        %b_m(k+1) = b_m(k) - Alpha * s_m

            Gradient = [(s_2 * a_1')'; s_2; (s_1 * a_0'); s_1];
            %Weight Update Hidden Layer
            W1_1 = W1_0 - (Alpha * s_1 * a_0');
            b1_1 = b1_0 - (Alpha * s_1);

            W2_0 = W2_1;
            b2_0 = b2_1;
            W1_0 = W1_1;
            b1_0 = b1_1;

            Norm_Grad = norm(Gradient);
            CumuNormGrad = CumuNormGrad + Norm_Grad;
        end % for p = -2:0.2:2

        %Avg norm(Gradient)
        AvgNormGrad = (CumuNormGrad/Max_TrainPts);
        pp = -2:0.2:2;
        W2_cse=W2_0; b2_cse=b2_0; W1_cse=W1_0; b1_cse=b1_0;
        CumuSumSqErrr = GetCumuSumSqErr(W2_cse, b2_cse, W1_cse, b1_cse, pp);
        %Avg Sum Sq error
        SumSq_Err = (CumuSumSqErrr/Max_TrainPts);
        if(Iter == 1)
            SumSq_ErrMat = [SumSq_Err];
        else
            SumSq_ErrMat = [SumSq_ErrMat SumSq_Err];
        end
        %----------------------
        ValidErr = 0;
        CumuValidErr = GetCumuSumSqErr(W2_cse, b2_cse, W1_cse, b1_cse, Validation);
        %Avg Validation error
        ValidErr = (CumuValidErr/length(Validation));
        if(Iter == 1)
            Valid_ErrMat = [ValidErr];
        else
            Valid_ErrMat = [Valid_ErrMat ValidErr];
        end
        %-----------------------
        %Check for convergence
        if(AvgNormGrad <= EpsGrad)
            disp(strcat('Solution Converged - AvgNormGrad <= EpsGrad, AvgNormGrad=',
num2str(AvgNormGrad)))
            AvgNormGrad
            break;
        end
        if(SumSq_Err <= EpsSSE)
            disp(strcat('Solution Converged - SumSq_Err <= EpsSSE, SumSq_Err=',
num2str(SumSq_Err)))
            SumSq_Err
            break;
        end
    end % for Iter = 1:Max_Iter

    if(Iter == Max_Iter)
        disp(strcat('Solution Converged - Max Iterations- ', int2str(Iter)))
    end
    SSE_b = SumSq_ErrMat;
    FileNameSSE = strcat(caseName_3, '-SSE.jpg');
    FileNameNAF = strcat(caseName_3, '-NAF.jpg');
    disp(strcat('Results for ', caseName_3));
    Titl = 'Sum Squared Error vs loglog(Iterations)';
    plt1 = 'SumSquared-Error';
    plt2 = 'Validation-Error';
    PlotSSE(SumSq_ErrMat, Valid_ErrMat, FileNameSSE, caseName_3, Titl, plt1, plt2 );

    Iter
    AvgNormGrad
    disp('Average Sum Squared Error value')
    mean(SumSq_ErrMat)
    disp('Average Validation Error value')
    mean(Valid_ErrMat)

    disp('Average Testing Sum Sq Error value')
    CumuTestErr = GetCumuSumSqErr(W2_0, b2_0, W1_0, b1_0, Testing);
    Test_Err = (CumuTestErr/length(Testing)) %only 10 points

    %Plot Network function vs Actual Function with final weights and biases
    PlotNetworkFunction(W2_0, b2_0, W1_0, b1_0, caseName_3, FileNameNAF)
    disp(strcat('Final Weights and biases'))
    W1_0
    b1_0
    W2_0
    b2_0

end %SDBP
```

GetCumuSumSqErr.m

```
function CumuSumSqErrr=GetCumuSumSqErr(W2_0, b2_0, W1_0, b1_0, p)

    CumuSumSqErrr = 0;
    for i = 1:length(p) %p = -2:0.2:2
        a_0 = p(i);
        a_1 = logsig (W1_0 * a_0 + b1_0);
        a_2 = purelin(W2_0 * a_1 + b2_0);
        a = a_2;

        t = exp(a_0/2).*sin(pi*a_0/2);
        e = t - a;
        CumuSumSqErrr = CumuSumSqErrr + e^2;
    end
```

PlotNetworkFunction.m

```
function PlotNetworkFunction(W2_0, b2_0, W1_0, b1_0, caseName, FileNameNAF)
%Overlay Plot of Network Function on the Actual Function
    a_ArrMat = [];
    for p = -2:0.1:2
        a_0 = p;
        a_1 = logsig (W1_0 * a_0 + b1_0);
        a_2 = purelin(W2_0 * a_1 + b2_0);
        a = a_2;

        if(p == -2)
            a_ArrMat = [a];
        else
            a_ArrMat = [a_ArrMat a];
        end
    end

    figure
    PlotTitle = strcat('Network function vs Actual function',caseName);
    title({' ',PlotTitle,' '})
    xlabel('Parameters - p')
    ylabel('Function Value@p')

    p=-2:0.1:2;
    Y = sineFunction(p);  %exp(p/2).*sin(pi*p/2);

    hold on
    hh=plot(p,Y,'LineWidth',8);
    set(hh(1),'linewidth',2);
    set(hh(1),'Color','b');
    hold on
    plot(p,a_ArrMat,'--rs','LineWidth',2,...
                    'MarkerEdgeColor','k',...
                    'MarkerFaceColor','g',...
                    'MarkerSize',2)
    saveas(hh, FileNameNAF);
    RegPlot=plotregression(a_ArrMat,Y,'LineWidth',3);
    RegplotName = strcat('Regression',caseName);
    RegplotName = strcat(RegplotName, '.jpg');
    saveas(RegPlot, RegplotName);

end
```

PlotSSE.m

```
function PlotSSE(SSE_1, SSE_2, FileNameSSE, caseName_3, Titl, plt1, plt2 )

    xlbl = 'log(Iterations)';
    ylbl = 'Sum Squared Error';
    Titl = strcat(Titl, caseName_3);
    figure
    h=semilogx(1:length(SSE_1), SSE_1,'LineWidth',2, 'Color','b');
    hold on
    h=semilogx(1:length(SSE_2), SSE_2,'LineWidth',2, 'Color','r');
    title({' ',Titl,' '});
    xlabel(xlbl);
    ylabel(ylbl);
    hleg1 = legend(plt1,plt2);
    saveas(h, FileNameSSE);

end
```

From Paper Review section part II – Paper notation to textbook notation

Paper – Training Feed Forward Networks with the Extended Kalman Algorithm