# MODEL IDENTIFICATION AND ADAPTIVE SYSTEMS PROJECT REPORT

## Nonlinear Model Identification with FROE and Feed-Forward Neural Networks

10513874- Mustafa Anıl Tuncel

10476432 – Ruifeng Ma

Politecnico di Milano, Polo Territoriale di Como

# Table of Contents

## 1. Introduction

The work is consisting of performing a nonlinear model identification by using the Forward Regression Orthogonal Estimates Method with polynomial NARX models and the feed-forward neural networks and the comparison of two approaches under different model complexity assumptions. The dataset is divided in identification and validation. Identification is used in estimating the model and the validation is used for the evaluation of the model in terms of both prediction and simulation accuracy. Before the identification, the data is partition into training and test datasets in order to prevent overfitting. Overfitting occurs when a model is excessively complex, such as having too many regressors relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data [2].

### 1.1.Dataset

This dataset is used in training a feed-forward neural network and a FROE with polynomial NARX model to predict the pH value of a solution in a tank from acid and base solution flow. The simulation data is of a pH neutralization process in a constant tank volume of 1100 liters. The acid solution concentration is (HAC) 0.0032 Mol/l. The base solution concentration is (NaOH) 0,05 Mol/l. The dataset contains 2001 number of samples with a sampling frequency of 10 seconds. The input data are twofold: Acid solution flow in liters (u1) and Base solution flow in liters (u2). The output (y1) is the pH of the solution in the tank. Our system is a multiple inputs single output (MISO) system.

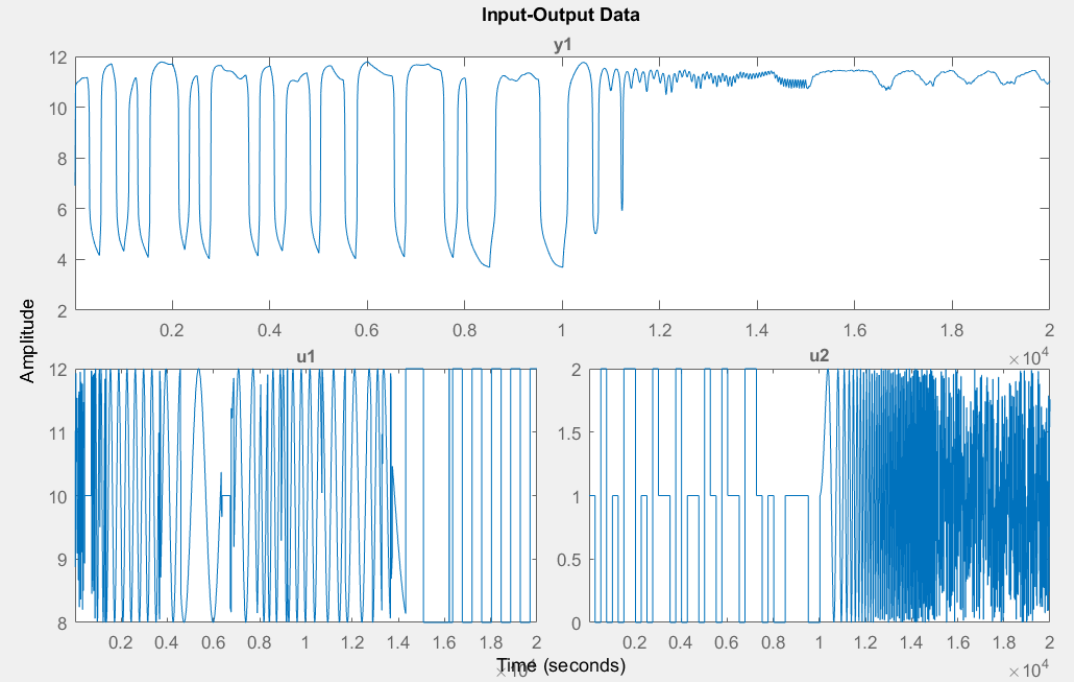



*Figure 1.* Depicts the amplitude of inputs u1, u2 and the output y1 as a function of time.

## 2. Identification of Polynomial NARX Models using FROE

A nonlinear autoregressive exogenous (NARX) model relates the output by the previous values of the output signal and previous values of an independent (exogenous) input signal. The generic NARX model is formulated as follows:

S: $y(t) = f(y(t-1), ..., y(t-n_y), u(t-1), ..., u(t-n_u))$

The parameter estimation problem can simply be solved by using Least Squares. In fact, the complication is to select the correct parameters of polynomial expansion to include in the model rather than estimating them. The number of possible polynomial terms increases exponentially with respect to the model order k. To evaluate a model, one has to estimate its parameters and measure its goodness of fit. However testing all the possible model structures is not feasible. Therefore many structure selection methods employ the forward regression approach. Which starts

with and empty model and adds one regressor at a time until an accuracy is satisfied. However, to add the correct regressor at a time, all of the regressor must be tested one by one. Forward selection can be speeded up even further using a technique called orthogonal least squares [3]. This is a Gram-Schmidt orthogonalization process [4] which ensures that each new column added to the design matrix of the growing subset is orthogonal to all previous columns.

We implemented the Forward Regression Orthogonal Estimates algorithm (Billings et al. 1989) in MATLAB under different model complexity assumptions. Our implementation begins with the generation of the candidate regressors that can be added to the model and creation of an empty model. We then computed the phi matrix for the candidate regressors, subsequently. Later we calculated the Error Reduction Ratio (ERR) for each column of the phi matrix. Then we selected the regressor with the highest ERR value to be included in the model as the first regressor. Afterwards, we iteratively selected the further regressors to be included into the model until a predefined value of threshold is satisfied. Having chosen the regressors, we used the nlarx method of System Identification Framework to estimate parameters of nonlinear ARX model.

The results are evaluated using MATLABs compare method in terms of the normalized root mean square (NRMSE).

### 2.1. Implementation of NARX Identification in PEM and SEM Frameworks

First of all we created an empty model with the model orders and delays specified below. For a model with ny output channels and nu input channels:

- na is an ny-by-ny matrix, where na(i,j) specifies the number of regressors from the jth output used to predict the ith output.
- nb is an ny-by-nu matrix, where nb(i,j) specifies the number of regressors from the jth input used to predict the ith output.

- nk is an ny-by-nu matrix, where nk(i,j) specifies the lag in the jth input used to predict the ith output.

Afterwards we created the empty system using the *idnlarx* method. *idnlarx* represents a nonlinear ARX model, which is an extension of the linear ARX structure and contains linear and nonlinear functions.

```
na = 0;
nb = [0 0];
nk = [0 0];
empty_orders = [na nb nk];
system = idnlarx(empty_orders,[]);
```

*Figure 2.* The empty model with its specified orders and delays

Then, we included each regressor that are selected by the FROE algorithm to the empty system one by one.

```
for ind = used_reg_ind
   system = addreg(system,candidateRegs(ind));
end
```

*Figure 3.* Inclusion of the regressors selected by the FROE algorithm

We used the idnlarx constructor to create the nonlinear ARX model and then estimate the model parameters using nlarx. MATLAB nlarx command computes the regressors from the current and past input values and the past output data. However it also allows us to specify custom regressors which are nonlinear functions of delayed inputs and outputs. Thus, all of the regressors selected by the FROE algorithm are included into the system as ‘custom regressors’ and their parameters are identified by the nlarx command. By default, nlarx uses numerical optimization to

minimize the *cost function*, a weighted norm of the prediction error. Moreover, nlarx command allows us to change the minimization objective. If the minimization objective is set to 'simulation' then the nlarx command minimizes the norm of the simulation error which is defined as the difference between the measured output and the simulated response of the model. The method for the estimation of the model parameters used by the nlarx command is nonlinear least squares. The figure below displays the model parameter estimation process with respect to the simulation error minimization.

Estimation Progress

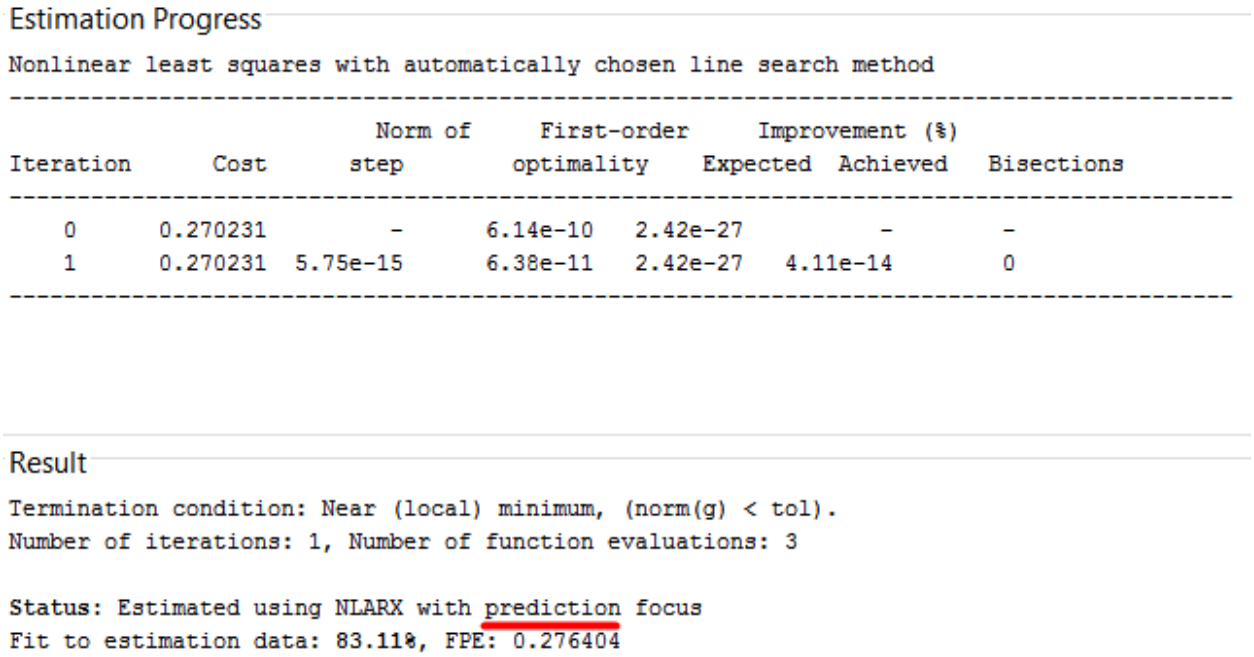
```
Initialization by one-step prediction error minimization...
Nonlinear least squares with automatically chosen line search method
-------------------------------------------------------------------------------------------
                          Norm of     First-order     Improvement (%)
Iteration      Cost       step        optimality    Expected  Achieved   Bisections
-------------------------------------------------------------------------------------------
    0      0.270231         -      6.14e-10   2.42e-27          -         -
    1      0.270231  5.75e-15      6.38e-11   2.42e-27   4.11e-14         0
-------------------------------------------------------------------------------------------
done.
Estimated by simulation error minimization...
Nonlinear least squares with automatically chosen line search method
-------------------------------------------------------------------------------------------
                          Norm of     First-order     Improvement (%)
Iteration      Cost       step        optimality    Expected  Achieved   Bisections
-------------------------------------------------------------------------------------------
    0      0.885251         -      2.87e+04       11.3          -         -
    1      0.782255      1.43      1.39e+04       11.3       11.6         0
    2      0.775525     0.115           295       3.51       0.86         1
    3      0.771601     0.278           105      0.452      0.506         0
    4      0.771403    0.0698          91.5     0.0203     0.0257         0
    5      0.771385    0.0204          20.7    0.00172    0.00223         0
-------------------------------------------------------------------------------------------
done.
```

Result

```
Termination condition: Near (local) minimum, (norm(g) < tol).
Number of iterations: 5, Number of function evaluations: 12

Status: Estimated using NLARX with simulation focus
Fit to estimation data: 71.42%, FPE: 0.323492
```

*Figure 4.* Estimation of the model parameters w.r.t. simulation error minimization



Estimation Progress

```
Nonlinear least squares with automatically chosen line search method
-------------------------------------------------------------------------------------
                          Norm of     First-order     Improvement (%)
Iteration      Cost       step         optimality    Expected  Achieved   Bisections
-------------------------------------------------------------------------------------
    0      0.270231         -       6.14e-10   2.42e-27          -         -
    1      0.270231  5.75e-15       6.38e-11   2.42e-27   4.11e-14         0
-------------------------------------------------------------------------------------
```

Result

```
Termination condition: Near (local) minimum, (norm(g) < tol).
Number of iterations: 1, Number of function evaluations: 3

Status: Estimated using NLARX with prediction focus
Fit to estimation data: 83.11%, FPE: 0.276404
```

*Figure 5.* Estimation of the model parameters w.r.t. prediction error minimization

### 2.2. FROE using Prediction Error Minimization

PEM minimizes the norm of the prediction error, which is defined as the difference between the measured output and the one-step ahead predicted response of the model. The first regressor chosen by FROE algorithm is y1(t-1). This is an expected result since the output at the previous time sample is a good regressor to explain the output.

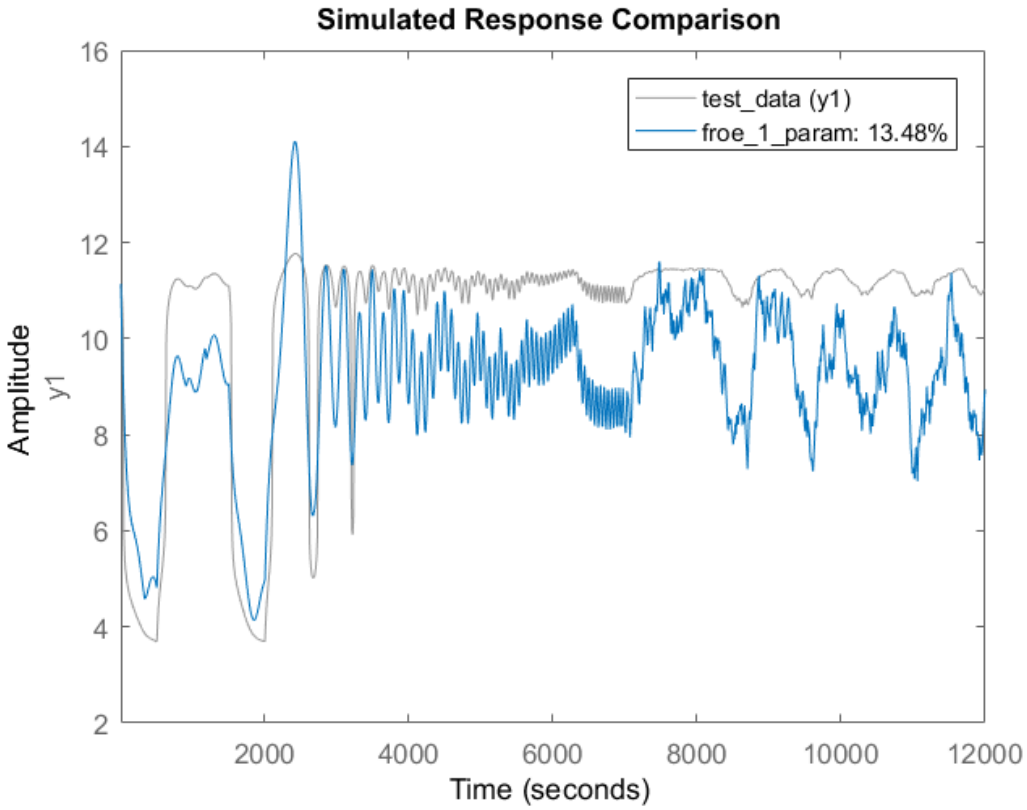



*Figure 6.* FROE model with y1(t-1) regressor w.r.t. the test data.

The second and third regressors chosen by the FROE algorithm are u2(t-1) and y1(t-1).*u2(t-1), correspondingly. The third regressor is the first nonlinear term in the model.

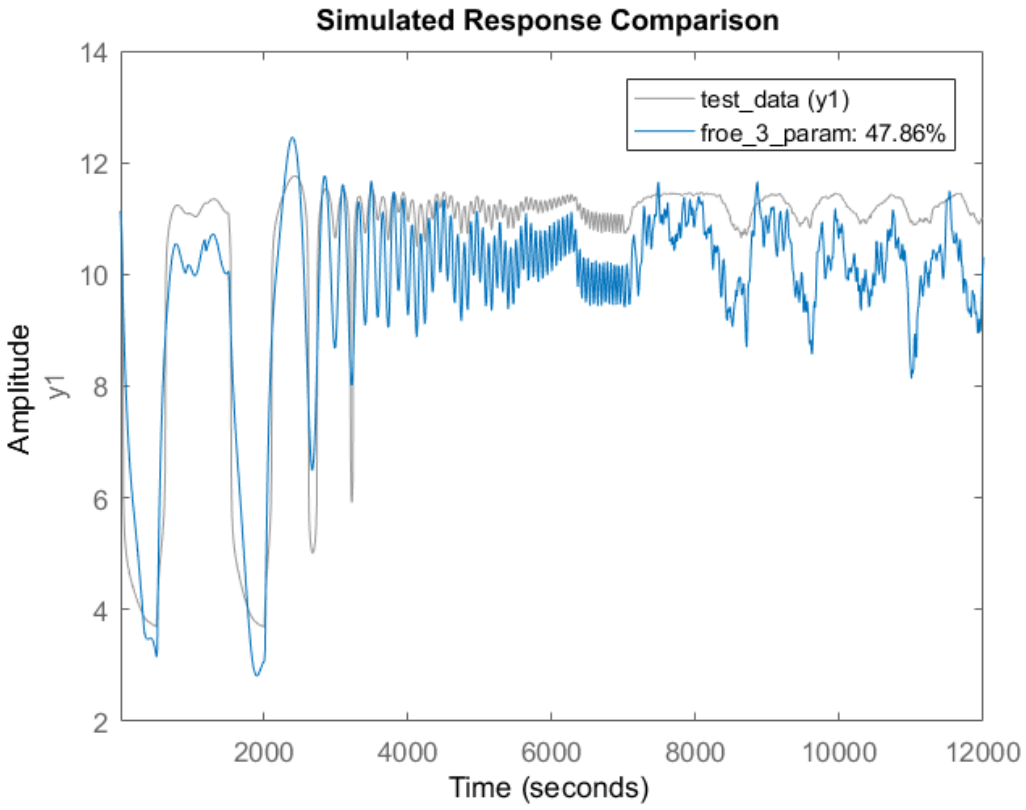



*Figure 7.* FROE model with regressors: y1(t-1), u2(t-1), y1(t-1).*u2(t-1)

As seen from the figure above by selecting two more regressors increases the fit to 47.86%.

After trying various threshold values, the best fit we get using FROE with PEM approach is the model having 8 regressors with the fit value of 60.43%. Those regressors are:

y1(t-1), u2(t-1), y1(t-1).*u2(t-1), u2(t-3), y1(t-1).*u2(t-3), y1(t-1).*u1(t-1), u1(t-3), u2(t-1).*u2(t-3).

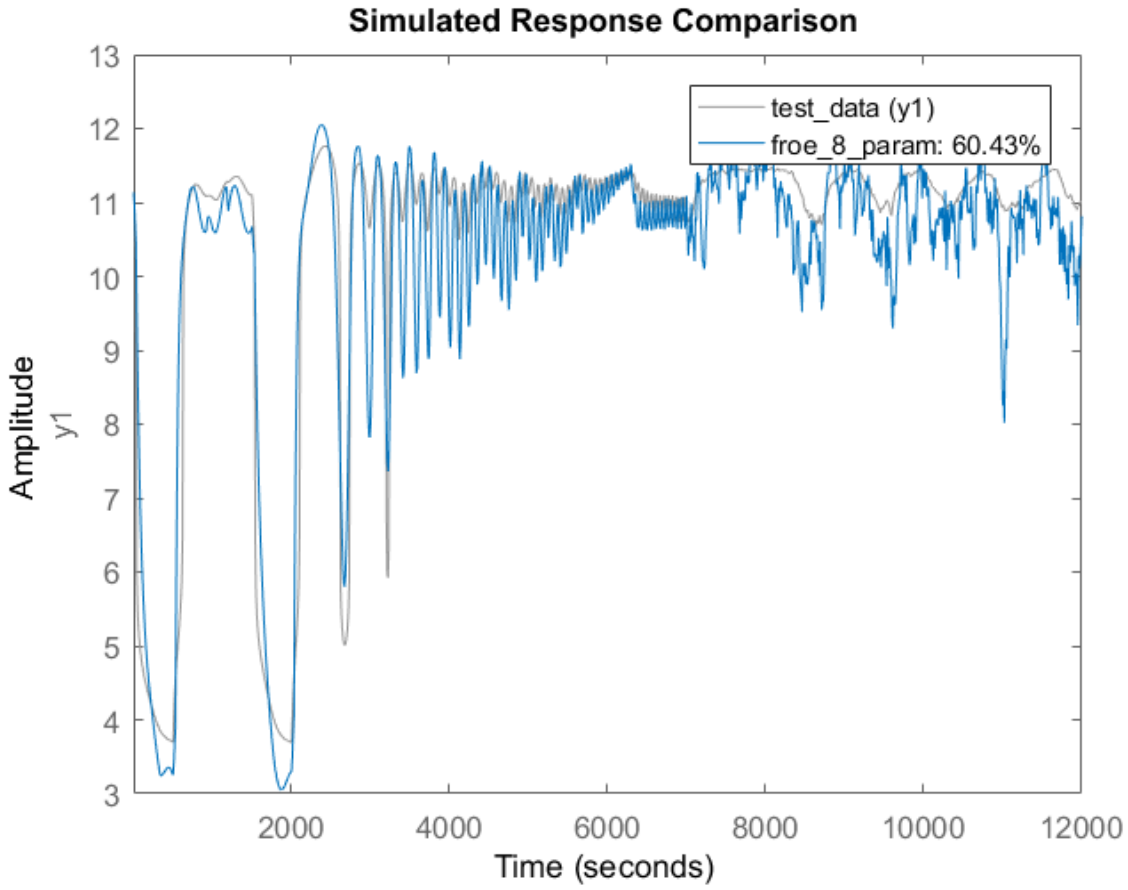



*Figure 8.* FROE model with eight regressors w.r.t. The test data.

### 2.3. FROE using Simulation Error Minimization

SEM minimizes the norm of the simulation error, which is defined as the difference between the measured output and simulated response of the model. The results we get by trying the FROE algorithm with SEM are as follows. Unlike the PEM approach, using only one regressor yields 42.06% fit to the test data.

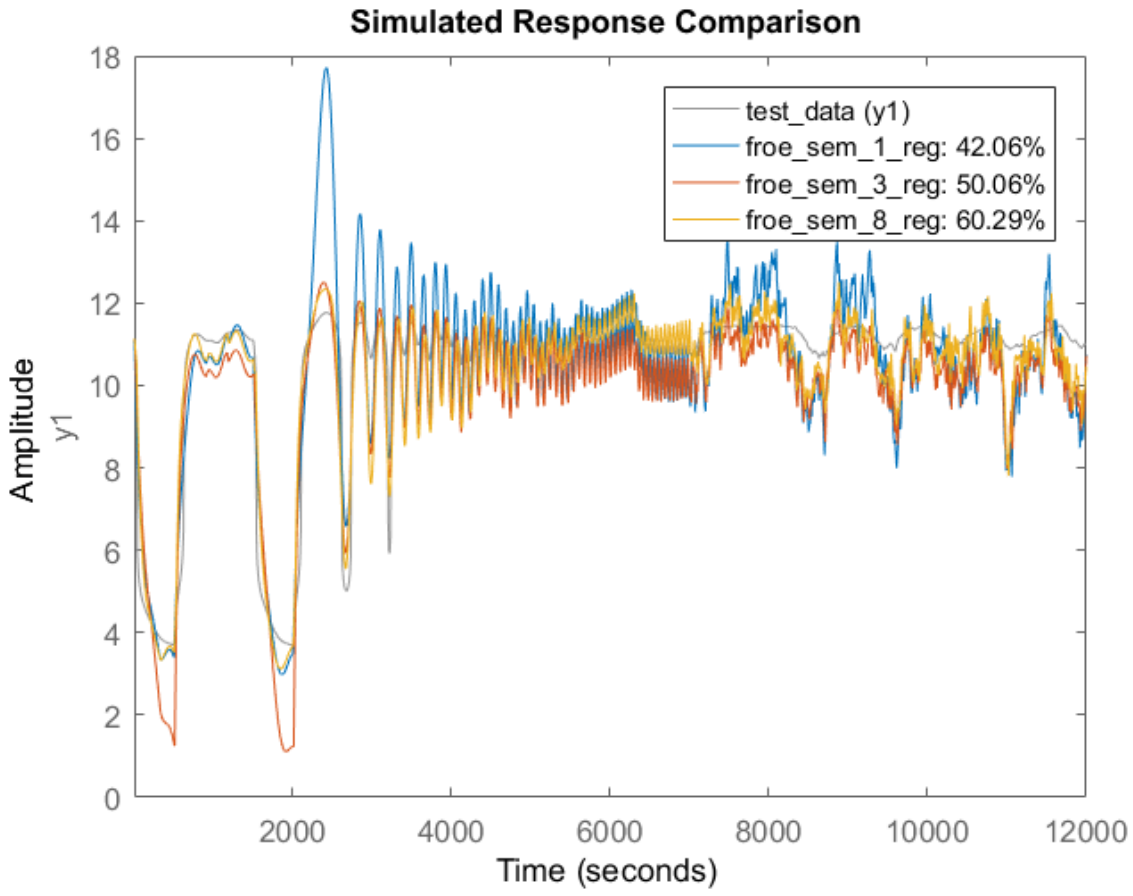



*Figure 9.* FROE using SEM approach, with 1, 3 and 8 regressors w.r.t. the test data.

### 2.4. Comparison of FROE Models

| Model | Regressors and Coefficients | NMSE |
|---|---|---|
| ARX (PEM) | 0.9045 * y(t-1) - 0.0466 * u1(t) + 0.4817 * u2 (t) | 13.48% |
| ARX (SEM) | 0.907 * y(t-1) - 0.03592 * u1(t) + 0.7165 * u2 (t) | 42.06% |
| FROE (PEM) | 0.9445 * y(t-1) - 0.03768 * u1(t) + 0.207 * u2(t) + 1.1941 * u2(t-1) - 0.0917 * y1(t-1)*u2(t-1) | 47.86% |
| FROE (SEM) | 0.9463 * y(t-1) - 0.0385 * u1(t) + 0.0368 * u2(t) + 1.8425 * u2(t-1) - 0.127 * y1(t-1)*u2(t-1) | 50.06% |
| FROE (PEM) | 0.9346 * y(t-1) -0.008 * u1(t) + 0.1949 * u2(t) + 0.048 * u2(y-1) + 0.0198 * y1(t-1) * u2(t-1) + 2.7421 * u2(t-3) - 0.1985 * y1 (t-1) * u2 (t-3) - 0.0041 * y1(t-1) * u1(t-1) + 0.0009 * u1(t-3) - 0.2046 * u2(t-1) * u2 (t-3) | 60.43% |
| FROE (SEM) | 0.9106 * y(t-1) -0.02574 * u1(t) + 0.277 * u2(t) + 0.0636 * u2(t-1) - 0.03596 * y1(t-1) * u2(t-1) + 1.184894 * u2(t-3) - 0.3166 * y1 (t-1) * u2 (t-3) - 0.00182 * y1(t-1) * u1(t-1) + 0.0011 * u1(t-3) - 0.3585 * u2(t-1) * u2 (t-3) | 60.29% |
| FROE (PEM) | 0.909 * y(y-1) + 1.2903 * u2 (t-1) - 0.1396 * y1(t-1) * u2(t-1) + 0.7843 * u2(t-3) | 43.18% |
| FROE (PEM) | 0.975* y(t-1) + 0.4781 * u2(t-1) - 0.03165 * y1(t-1) * u2(t-1) + 2.344 * u2 (t-3) -0.1761 * y1(t-1) * u2(t-3) - 0.005 * y1(t-1) * u1(t-1) | 54.31% |
| FROE (SEM) | 0.919 * y(y-1) + 1.7364 * u2 (t-1) - 0.1466 * y1(t-1) * u2(t-1) + 0.4586 * u2(t-3) | 48.18% |
| FROE (SEM) | 0.9666* y(t-1) + 0.9672 * u2(t-1) - 0.07295 * y1(t-1) * u2(t-1) + 1.6406 * u2 (t-3) -0.103 * y1(t-1) * u2(t-3) - 0.0066 * y1(t-1) * u1(t-1) | 49.06% |

*Table 1.* NMSE values of the FROE Models

## 3. Feed-forward Neural Networks

Artificial neural networks are functional approximators and thus can be used in black-box identification of the nonlinear systems. The most common neural network structure is the feedforward multilayer network. A feedforward neural network consists of a series of layers. The first layer has the connection between the input and the network. Each subsequent layer however, has a connection from a previous layer. The output is produced by the final layer of the network. In other words, the information moves only in one direction which is forward starting from the input nodes and through the hidden nodes. There are no loops or cycles in a feedforward neural network. We evaluated numerous neural network architectures with respect to the number of neurons contained into the hidden layer and the learning algorithm employed.

### 3.1. Neural Network Learning Algorithms

Training a neural network means selecting one model from a set of allowed models that minimizes the cost function. There are various algorithms available for training the neural network models.

Levenberg-Marquardt is a neural network training function that updates the weight and bias values according to the Levenberg-Marquardt optimization. *Trainlm* method is referred as the fastest backpropogation algorithm in the neural network toolbox therefore it is recommended as the default neural network learning algorithm.

Bayesian regularization propagation is another neural network training function that again updates the weight and bias values according to the Levenberg-Marquardt optimization however, it minimizes a combination of squared error and weights and determines the correct combination so as to produce a network that generalizes well.

Another learning algorithm is the Gradient descent with momentum backpropagation. This training algorithm updates the bias and weight values according to the gradient descent with momentum. The other learning algorithms available in MATLAB are: Conjugate gradient backpropagation with Polak-Ribiére updates (cgp), Conjugate gradient backpropagation with Fletcher-Reeves updates (cgf), Conjugate gradient backpropagation with Powell-Beale restarts (cgb), BFGS quasi-Newton backpropagation (bgf), Gradient descent with adaptive learning rate backpropagation, Gradient descent with momentum and adaptive learning rate backpropagation (gda), One-step secant backpropagation (oss), Resilient backpropagation (rp) and Scaled conjugate gradient backpropagation (scg).

| Acronym | Algorithm | Description |
|---|---|---|
| LM | `trainlm` | Levenberg-Marquardt |
| BFG | `trainbfg` | BFGS Quasi-Newton |
| RP | `trainrp` | Resilient Backpropagation |
| SCG | `trainscg` | Scaled Conjugate Gradient |
| CGB | `traincgb` | Conjugate Gradient with Powell/Beale Restarts |
| CGF | `traincgf` | Fletcher-Powell Conjugate Gradient |
| CGP | `traincgp` | Polak-Ribiére Conjugate Gradient |
| OSS | `trainoss` | One Step Secant |
| GDX | `traingdx` | Variable Learning Rate Backpropagation |

*Table 2.* Neural network training algorithms - acronyms

### 3.2. Evaluating the Neural Networks

As stated before, we tried various different neural network architectures trained with numerous learning algorithms. Our neural networks has two layers: one nonlinear hidden layer and an output layer which is linear as shown in the figure below. We conducted our tests by changing the number of neurons in the hidden layer and the network training algorithm employed. This is achieved using the command below.

```
feedforwardnet(hiddenSizes,trainFcn)
```

Where, hiddenSizes parameter specifies the size of the hidden layer and the trainFcn specifies the training algorithm employed.

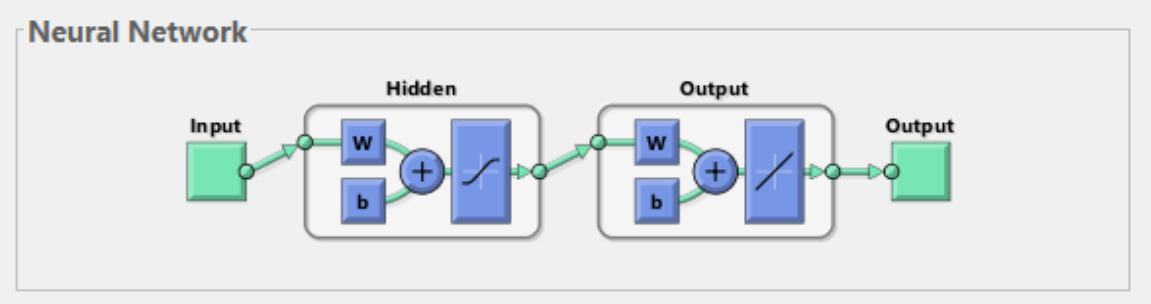



*Figure 10.* Neural network architecture with one non-linear hidden and one linear output layer

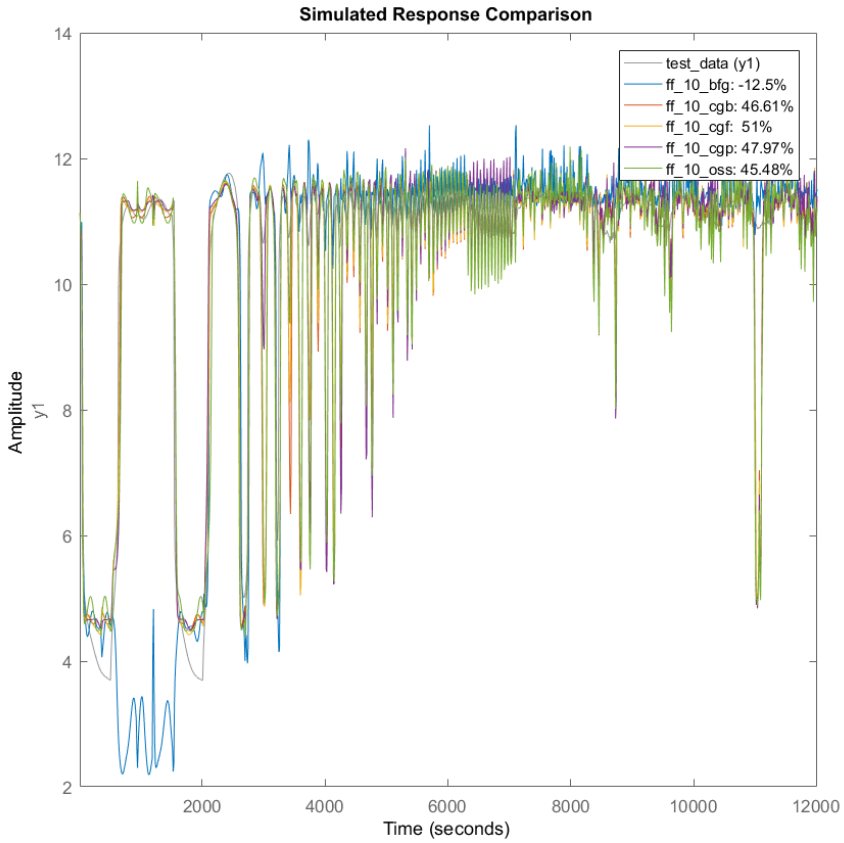



*Figure 11.* Comparison of learning algorithms with 10 neurons in the hidden layer

The figure above depicts the performances comparison of five different learning algorithms trained on a neural network architecture having 10 neurons in the hidden layer. As the plot shows, BFGS quasi-Newton backpropagation yielded poor results whereas the other four algorithms shown similar performance. The best fit performance is 51% by the Conjugate gradient backpropagation with Fletcher-Reeves updates algorithm. In fact this is a lower result than the FROE method.

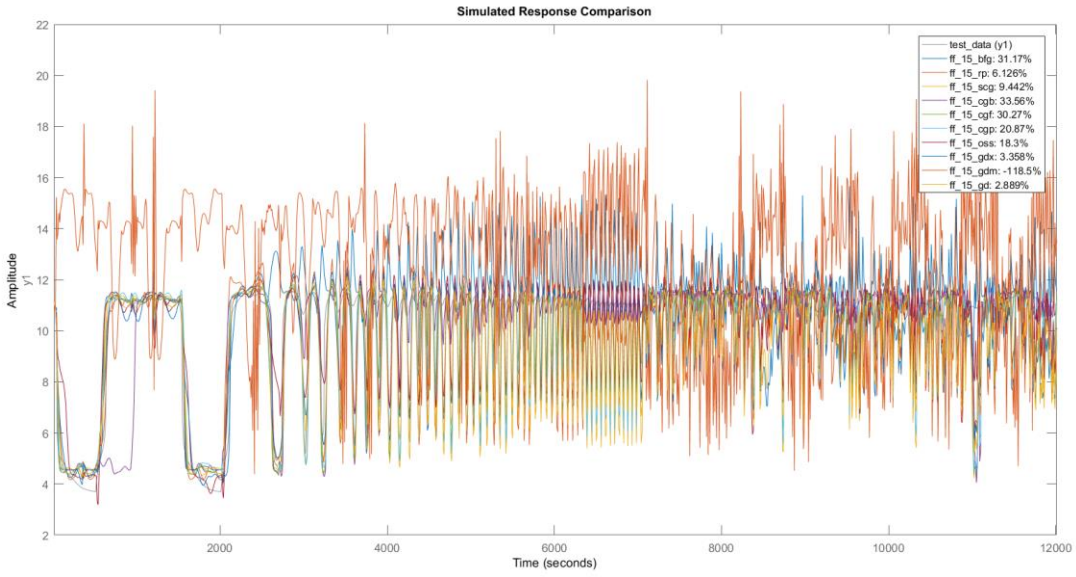



*Figure 12.* Comparison of learning algorithms with 15 neurons in the hidden layer

The figure above shows the evaluation of the MATLAB learning algorithms with the hidden layer size equal to 15.

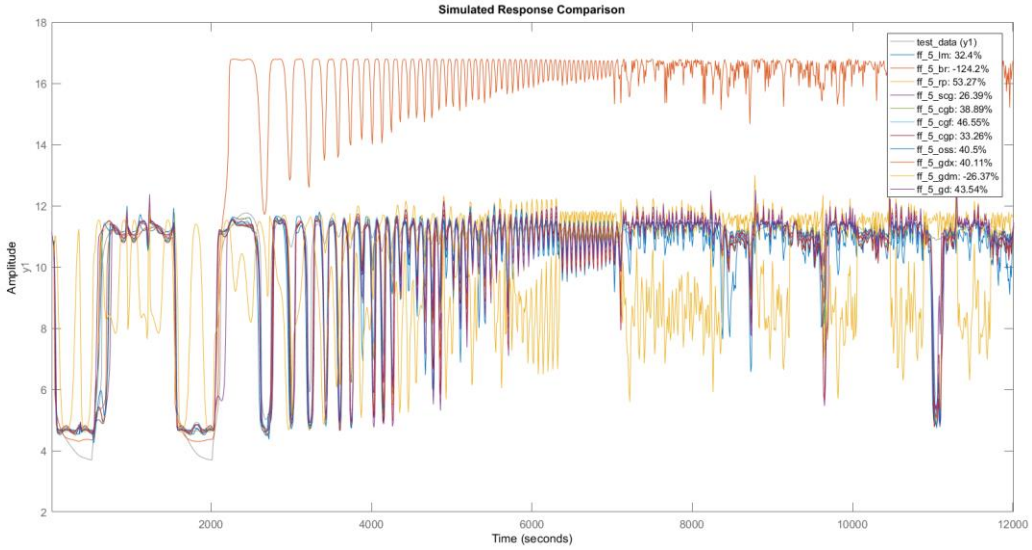



*Figure 13.* Comparison of learning algorithms with 5 neurons in the hidden layer

In this experiment, the highest fit value we could get is the Resilient backpropagation algorithm with 53.27%. Still the FROE algorithm outperforms the neural network models.

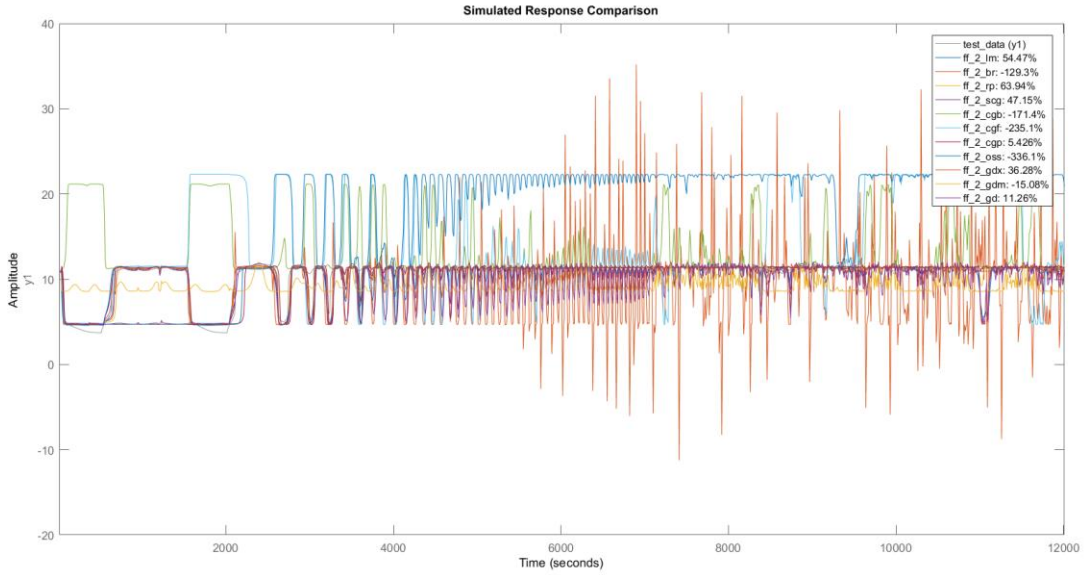



*Figure 14.* Comparison of learning algorithms with 2 neurons in the hidden layer

With this feedforward neural network architecture having two neuron in the hidden layer, we achieved a slightly better performance than the FROE algorithm. After conducting further tests, we achieved a 73.5% fit with Levenberg-Marquardt backpropagation algorithm.

| Training Algorithm | Hidden Layer Size | NMSE |
|---|---|---|
| Levenberg-Marquardt | 2 | 54.47% |
| Bayesian Regularization | 2 | -129.30% |
| Resilient Backpropagation | 2 | 63.94% |
| Scaled conjugate gradient backpropagation | 2 | 47.15% |
| Conjugate gradient backpropagation Polak-Ribiere | 2 | 5.43% |
| Conjugate gradient backpropagation Fletcher-Reeves | 2 | -235.10% |
| Conjugate gradient backpropagation Power-Beale | 2 | -171.40% |
| One step secant backpropagation | 2 | -336.10% |
| Levenberg-Marquardt | 20 | 43.32% |
| Levenberg-Marquardt | 15 | 51.45% |
| Levenberg-Marquardt | 25 | 48.40% |
| Resilient Backpropagation | 15 | 6.13% |
| Scaled conjugate gradient backpropagation | 15 | 9.44% |
| Conjugate gradient backpropagation Power-Beale | 15 | 33.56% |
| Conjugate gradient backpropagation Fletcher-Reeves | 15 | 30.27% |
| Conjugate gradient backpropagation Polak-Ribiere | 15 | 20.87% |
| One step secant backpropagation | 15 | 18.30% |
| Conjugate gradient backpropagation Power-Beale | 10 | 47.97% |
| Conjugate gradient backpropagation Fletcher-Reeves | 10 | 51.00% |
| Conjugate gradient backpropagation Polak-Ribiere | 10 | 46.61% |
| One step secant backpropagation | 10 | 45.48% |
| Levenberg-Marquardt | 16 | 47.92% |
| Levenberg-Marquardt | 17 | 73.50% |
| Levenberg-Marquardt | 18 | 47.72% |
| Levenberg-Marquardt | 19 | 40.83% |
| Levenberg-Marquardt | 5 | 32.40% |
| Bayesian Regularization | 5 | -124.20% |
| Resilient Backpropagation | 5 | 53.27% |
| Scaled conjugate gradient backpropagation | 5 | 26.39% |
| Conjugate gradient backpropagation Power-Beale | 5 | 38.89% |
| Conjugate gradient backpropagation Fletcher-Reeves | 5 | 46.55% |
| Conjugate gradient backpropagation Polak-Ribiere | 5 | 33.26% |
| One step secant backpropagation | 5 | 40.50% |

*Table 3.* Neural network comparison table

## 4. Discussion

In terms of the computational expense, FROE algorithm requires much less time than the traditional forward regression methods since FROE always select the orthogonal regressor among all of the candidate regressors. The time complexity of neural networks, on the other hand depends on the architecture of the network. Another variable that has significant importance on the computational expense of neural networks is the training algorithm employed. Levenberg-Marquardt algorithm is known for its fast convergence and stability properties. Therefore it yields the results within less time than the other neural network training algorithms. Even though both FROE and the neural networks require more or less the same time, it is correct to consider the FROE implementation as still the fastest solution due to the fact that the MATLAB Neural Network Toolkit uses parallelism with multicore CPU computing. In fact, even the simplest neural network requires 20 iterations to achieve stable results. The upper bound of the number of iterations is 1000 Whereas the FROE implementation using either PEM or SEM models are able to achieve %60 fit on the test data by a mere eight iterations.

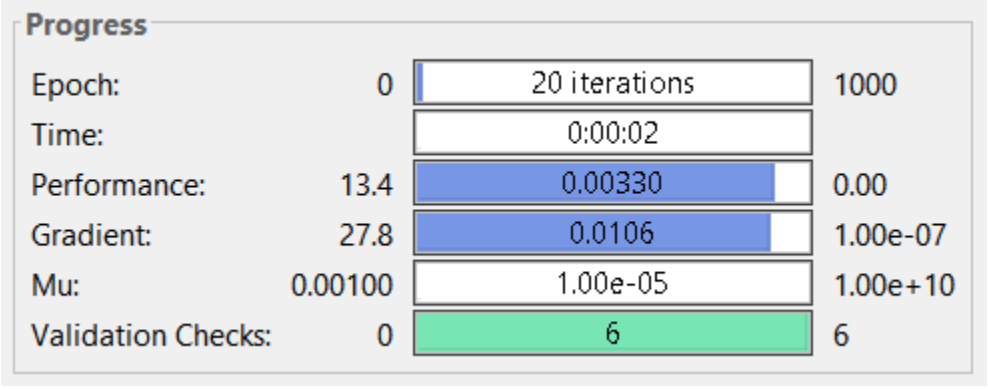



*Figure 15.* The minimum number of iterations needed for the two layered neural network architecture

One disadvantage of the FROE is its being initialization dependent since the order in which the regressors are included influences the model selection process. Therefore the first chosen term is most likely to be y(t-1) that is the previously sampled output. The final figure plots the highest NMSE values we achieved using both FROE and neural networks.

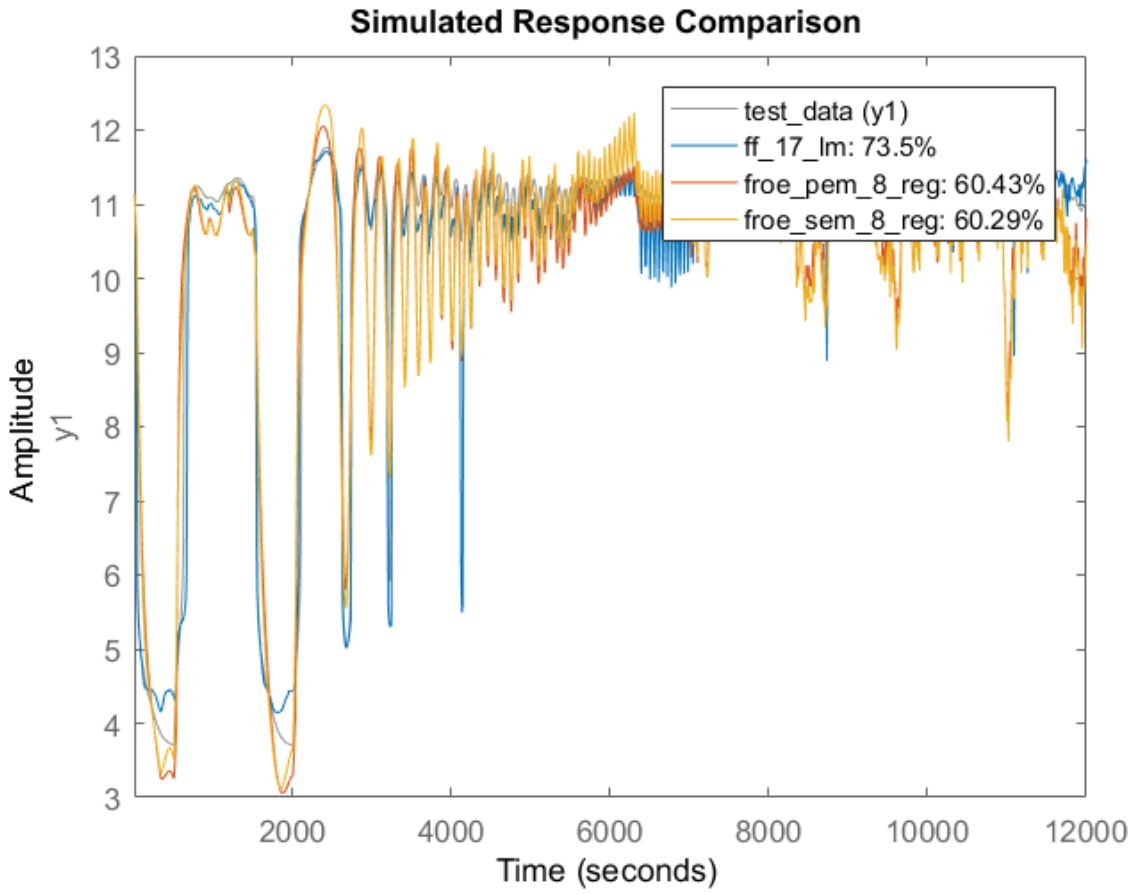



*Figure 16.* The plot of the highest NMSE values achieved

## References


1) T.J. Mc Avoy, E.Hsu and S.Lowenthal, Dynamics of pH in controlled stirred tank reactor, Ind.Eng.Chem.Process Des.Develop.11(1972)
2) Martha K. Smith (2014-06-13). "Overfitting". University of Texas at Austin. Retrieved 2016-07-31.
3) S. Chen, C.F.N. Cowan, and P.M. Grant. Orthogonal least squares learning for radial basis function networks. IEEE Transactions on Neural Networks, 2(2):302-309, 1991.
4) R.A. Horn and C.R. Johnson. Matrix Analysis. Cambridge University Press, Cambridge, UK, 1985.
5) MATLAB System Identification Toolbox, https://uk.mathworks.com/help/ident
6) MATLAB Neural Network Toolbox, https://uk.mathworks.com/help/nnet/
7) MIAS Course, Prof. Luigi Piroddi , http://home.deib.polimi.it/piroddi/mias.html