A Convex Combination of Two Adaptive Filters as Applied to Economic Time Series

Adaptive Signal Processing
EEL 6502
1 May 2006

Casey T. Morrison

1 Introduction

While conventional adaptive filters allow for the dynamic adjustment of performance based on a sequence of inputs and error measurements, a recent publication [1] by Arenas-García, Figueiras-Vidal, and Sayed claims that improved performance may be achieved with the combination of multiple filters of this sort.

The objective of this experiment was to verify (or dispute) the claims of the authors and to evaluate the performance of a convex combination of two adaptive filters when applied to an economic time series. The problem of filtering economic data is complicated by the inherent non-stationarity of time series of this sort. It goes without saying that highly-accurate filters that can predict future values of such time series would be very valuable. As a result, this problem has been studied thoroughly and has proven to be a very difficult and complex one.

2 Convex Combination of Two Adaptive Filters

The authors of [1] propose an adaptive filter structure illustrated in Figure 2.1 below. As shown in the figure, updates are made to each component filter, as well as to the combination filter based on various error measurements and rules.

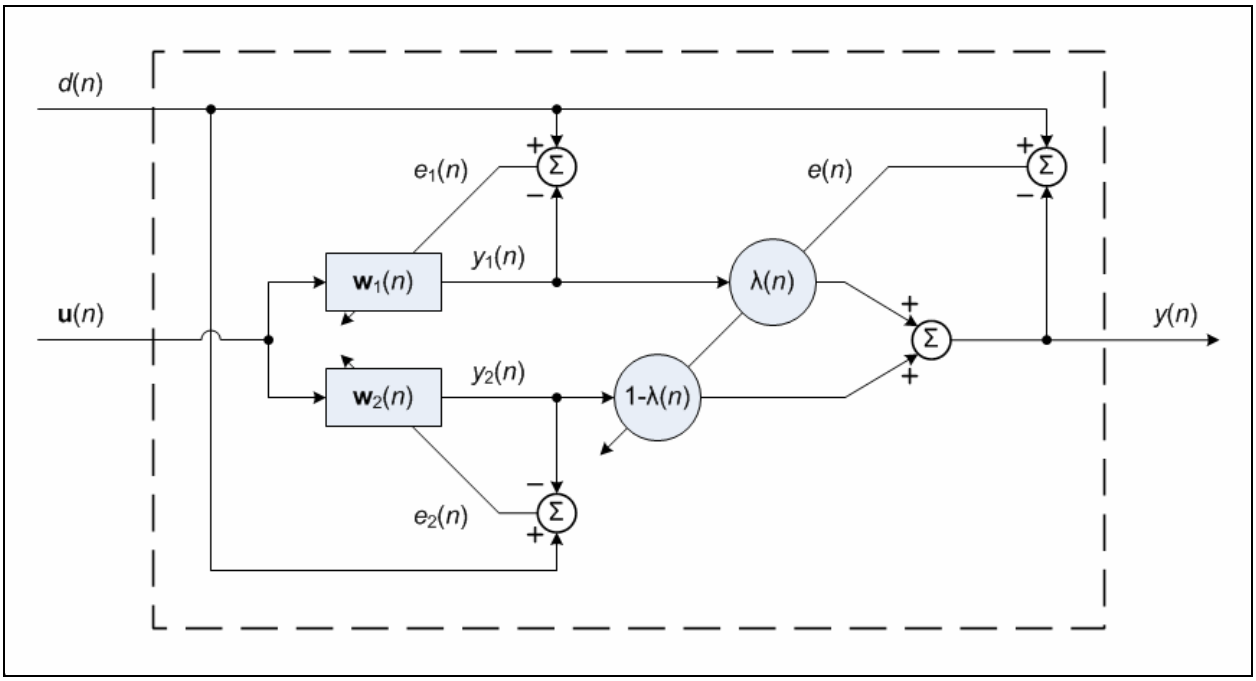

Figure 2.1: Adaptive convex combination of two transversal filters

The output of such a filter is defined by the following equation,

$$y(n) = \lambda(n)y_1(n) + [1-\lambda(n)]y_2(n),$$

where $y_1(n)$ and $y_2(n)$ are outputs of the two transversal filters, and $\lambda(n)$ is a mixing scalar parameter that lies between zero and one. With a properly chosen $\lambda(n)$, the combination filter can effectively extract the best properties of the individual filters $\vec{w}_1(n)$ and $\vec{w}_2(n)$.

A nonlinear adaptation of the mixing parameter is proposed according to the following sigmoidal function:

$$\lambda(n) = sgm[a(n)] = \frac{1}{1+e^{-a(n)}},$$

which appropriately ranges from zero to one in a manner depicted in Figure 2.2.

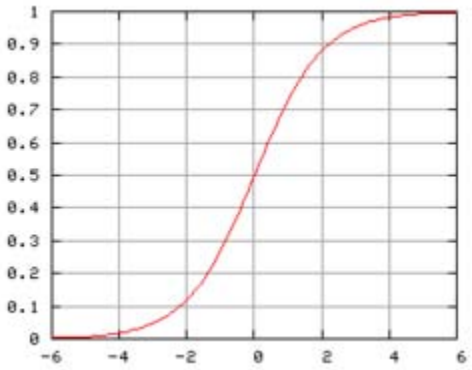

Figure 2.2: Sigmoid function

The variable $a(n)$ that defines $\lambda(n)$ is updated via the following recursive relation:

$$a(n+1) = a(n) + \mu_a e(n)[y_1(n) - y_2(n)]\lambda(n)[1-\lambda(n)].$$

An important result of this adaptation rule is that the stochastic gradient noise and the adaptation speed near $\lambda(n) = 1$ and $\lambda(n) = 0$ are reduced. This allows the combination filter to perform close to the individual filter corresponding to $\lambda(n) = 1$ or $\lambda(n) = 0$.

To avoid a situation where the adaptation of $a(n)$ ceases as $\lambda(n) \to 0$ or $\lambda(n) \to 1$, the authors propose a restriction on $a(n)$ such that $a(n) \in [-a^+, a^+]$ and $\lambda(n) \in [1-\lambda^+, \lambda^+]$, where $\lambda^+ = sgm[a^+]$.

A modification to the mixing parameter is proposed as follows:

$$y_u(n) = \lambda_u(n)y_1(n) + [1-\lambda_u(n)]y_2(n)$$

$$\lambda_u(n) = \begin{cases} 0, & a(n) \le -a^+ + \varepsilon \\ \lambda(n), & -a^+ + \varepsilon < a(n) < a^+ - \varepsilon \\ 1, & a(n) \ge a^+ - \varepsilon \end{cases}$$

where ε is a small positive constant.

With this structure in place, the authors of [1] assert that $y_u(n)$ is universal with respect to its components. This means that it performs, in steady-state, at least as well as the best component filter, and, in some cases, better than both component filters. Likewise, $y(n)$ is shown to be *nearly* universal, meaning it can perform as close as desired to the best component filter.

2.1 Notation

For purposes of performance analysis, the authors define some notation and variables. In addition, other performance metrics are defined for the purposes of evaluating the prediction accuracy of this architecture.

Table 2.1.1: Notation and variables

Variable description	Definition	Conditions
Weight error vector	$\vec{\varepsilon}_i(n) = \vec{w}_o - \vec{w}_i(n)$	For component filters $i = 1, 2$
	$\vec{\varepsilon}(n) = \vec{w}_o - \vec{w}(n)$	For combination filter
A priori errors	$e_{a,i}(n) = \vec{\varepsilon}_i^T(n)\vec{u}(n)$ $e_a(n) = \vec{\varepsilon}^T(n)\vec{u}(n)$	$i = 1, 2$
A posteriori errors	$e_{p,i}(n) = \vec{\varepsilon}_i^T(n+1)\vec{u}(n)$ $e_p(n) = \vec{\varepsilon}^T(n+1)\vec{u}(n)$	$i = 1, 2$
Excess mean-square error (EMSE)	$J_{ex,i}(\infty) = \lim_{n\to\infty} \mathrm{E}\{e_{a,i}^2(n)\}$	For individual filters $i = 1, 2$
	$J_{ex}(\infty) = \lim_{n\to\infty} \mathrm{E}\{e_a^2(n)\}$	For combination filter
Cross-EMSE of the component filters	$J_{ex,12}(\infty) = \lim_{n\to\infty} \mathrm{E}\{e_{a,1}(n)e_{a,2}(n)\}$	Steady-state correlation between the component filters
Percent error	$e_{\%} = \dfrac{\lvert d(n) - y(n)\rvert}{\lvert d(n)\rvert} \cdot 100\%$	Percent error in magnitude

2.2 Universality of the Combination Filter

The two combination schemes proposed earlier, $y(n)$ and $y_u(n)$, were examined by the authors in terms of their steady-state performance. In general, it is difficult to evaluate the EMSE for each scheme. However, there exist two cases in which this evaluation can be simplified.

1. If $\lim_{n\to\infty} \mathrm{E}\{a(n)\} = a^+$, then the EMSEs may be simplified to

$$J_{ex}(\infty) \approx J_{ex,1}(\infty)$$
$$J_{ex,u}(\infty) = J_{ex,1}(\infty),$$

where $J_{ex,u}(\infty)$ is the EMSE for the second combination scheme, $y_u(n)$.

2. If $\lim_{n\to\infty} \mathrm{E}\{a(n)\} = -a^+$, then the EMSEs may be simplified to

$$J_{ex}(\infty) \approx J_{ex,2}(\infty)$$
$$J_{ex,u}(\infty) = J_{ex,2}(\infty),$$

The authors introduce the differences

$$\Delta J_i = J_{ex,i}(\infty) - J_{ex,12}(\infty), \quad i = 1,2,$$

which measure the difference between the individual EMSEs and the cross-EMSE. The following three cases were examined in regard to ΔJ_i.

1. $\Delta J_1 \le 0$ and $\Delta J_2 \ge 0$ (i.e. $J_{ex,1}(\infty) \le J_{ex,12}(\infty) \le J_{ex,2}(\infty)$):
   It can be shown that in this case, both combination schemes perform like their best component filter. This can be stated as
$$J_{ex}(\infty) \approx J_{ex,1}(\infty)$$
$$J_{ex,u}(\infty) = J_{ex,1}(\infty).$$
2. $\Delta J_1 \ge 0$ and $\Delta J_2 \le 0$ (i.e. $J_{ex,1}(\infty) \ge J_{ex,12}(\infty) \ge J_{ex,2}(\infty)$):
   It can also be shown that in this case, the behavior of the overall filter is as good as its best element. This can be stated as
$$J_{ex}(\infty) \approx J_{ex,2}(\infty)$$
$$J_{ex,u}(\infty) = J_{ex,2}(\infty).$$
3. $\Delta J_1 > 0$ and $\Delta J_2 > 0$ (i.e. $J_{ex,12}(\infty) < J_{ex,1}(\infty)$ and $J_{ex,12}(\infty) < J_{ex,2}(\infty)$):
   It can be shown that in this case, both combination filters outperform the two component filters. This can be stated as
$$J_{ex}(\infty) < \min\{J_{ex,1}(\infty), J_{ex,2}(\infty)\}$$
$$J_{ex,u}(\infty) < \min\{J_{ex,1}(\infty), J_{ex,2}(\infty)\}.$$

This analysis leads to the conclusion that the first combination scheme, $y(n)$, is nearly universal, i.e., its steady-state performance is at least as good as that of the best component filter. It can also be concluded that the second combination scheme, $y_u(n)$, is universal in the sense that it performs at least as well as the best element in the mixture. In fact, both schemes are capable of outperforming their components in certain circumstances. These conclusions are summarized in Table 2.2.1.

Table 2.2.1: EMSEs of two combination schemes as a function of $\mathrm{E}\{a(\infty)\}$

	$\mathrm{E}\{a(\infty)\} = -a^+$	$-a^+ < \mathrm{E}\{a(\infty)\} \le -a^+ + \varepsilon$	$-a^+ + \varepsilon < \mathrm{E}\{a(\infty)\} < a^+ - \varepsilon$	$a^+ - \varepsilon \le \mathrm{E}\{a(\infty)\} < a^+$	$\mathrm{E}\{a(\infty)\} = a^+$
$J_{ex}(\infty)$	$\approx J_{ex,2}(\infty)$	$< \min\{J_{ex,1}(\infty), J_{ex,2}(\infty)\}$			$\approx J_{ex,1}(\infty)$
$J_{ex,u}(\infty)$	$= J_{ex,2}(\infty)$		$< \min\{J_{ex,1}(\infty), J_{ex,2}(\infty)\}$	$= J_{ex,1}(\infty)$	

3 Combination of Two NLMS Filters

The performance of an adaptive convex combination of two Normalized Least-Mean-Square (NLMS) filters that differ only in step-size was analyzed. The input data for this system consists of a semi-stationary economic time series. In particular, the closing stock price for Texas Instruments (TXN) from March 27, 2002 to March 16, 2006 was used as a data source.

3.1 Simulation Environment

A mathematical implementation of the proposed adaptive filter structure was constructed in Matlab (see Appendix A for the Matlab code). As previously illustrated in Figure 1 (page 2), the overall structure of the combination filter consists of two component filters that are adjusted by their respective prediction errors, and the combined prediction is formed by mixing the outputs of each component filter in a convex manner.

Each component filter is a Normalized Least-Mean-Square (NLMS) filter. The reasons behind this design decision are two-fold [2]:

1. The NLMS filter mitigates the gradient noise amplification problem, which can arise when the tap-input vector $\vec{u}(n)$ is large.
2. The rate of convergence of the NLMS filter is potentially faster than that of the conventional LMS filter for both uncorrelated and correlated data.

As previously mentioned, the combination filter draws its input data from 1000 samples of an economic time series (TXN closing price). This data spans from March 27, 2002 (sample 999) to March 16, 2006 (sample 0). Figure 3.1.1 below shows the input sequence used to excite the combination filter.

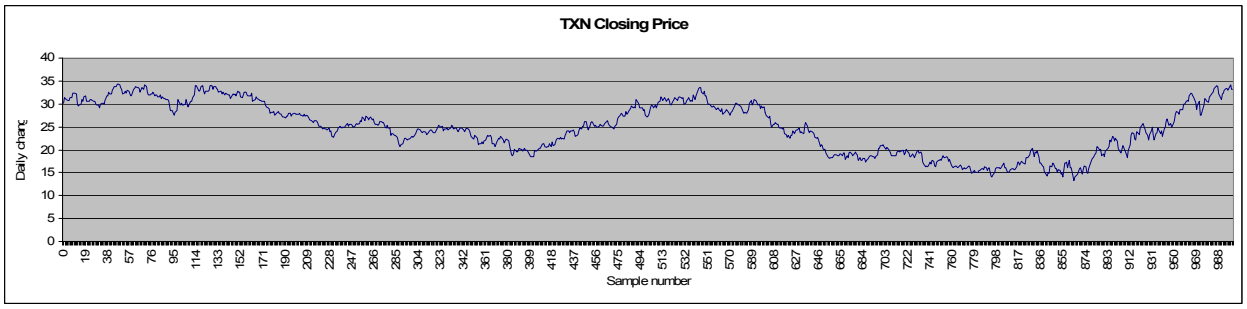

Figure 3.1: Input data sequence $\vec{u}(n)$

The NLMS filters used in this environment are each built around an M-tap transversal filter. The tap weights for each filter are updated according to the following recursive equation [2].

$$\vec{w}_i(n+1) = \vec{w}_i(n) + \frac{\tilde{\mu}_i}{\delta + \left\|\vec{u}(n)\right\|^2}\vec{u}(n)e_i^{\,*}(n), \quad i = 1,2$$

$$e_i(n) = d(n) - \vec{w}_i^{\,H}(n)\vec{u}(n),$$

where $\tilde{\mu}_i$ is the adaptation constant for the i^{th} filter, $\vec{u}(n)$ is the M-by-1 tap input vector at time step n, δ is a small positive constant offset, and $d(n)$ is the desired response at time step n. Figure 3.1.2 on the following page illustrates the structure of an M-tap NLMS transversal filter.

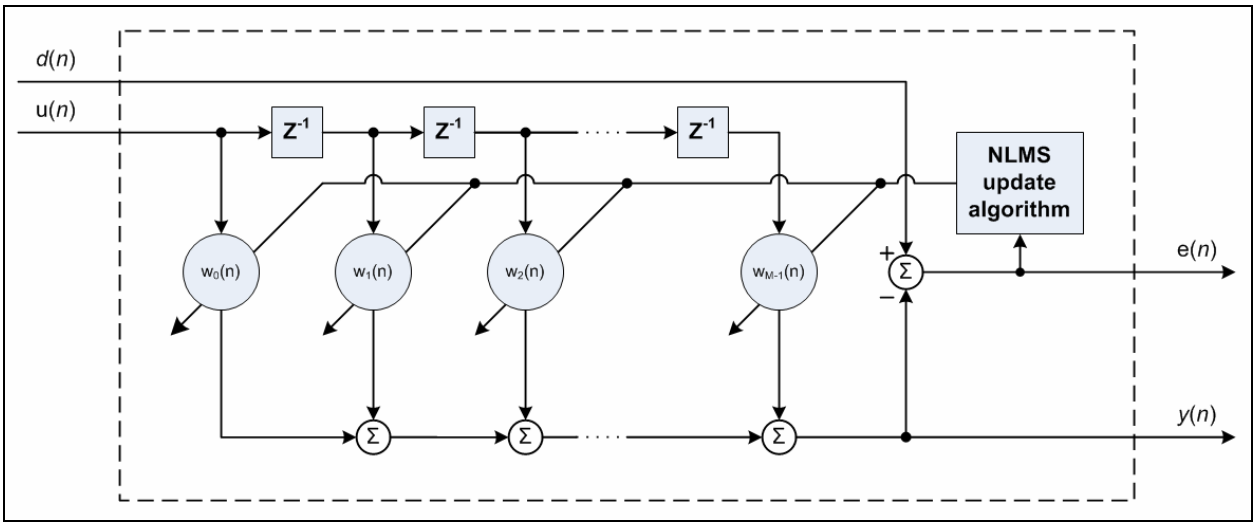

Figure 3.2: NLMS transversal filter structure

The combination of the two NLMS filters was accomplished by applying both the universal and the nearly-universal mixing schemes developed by the authors.

3.2 Experimental Procedures

Prior to the simulation of the combination filter, numerous individual NLMS filters with different adaptation constants were simulated. Figure 3.2.1 below shows the plot of the mean-square error (MSE) for each filter over time.

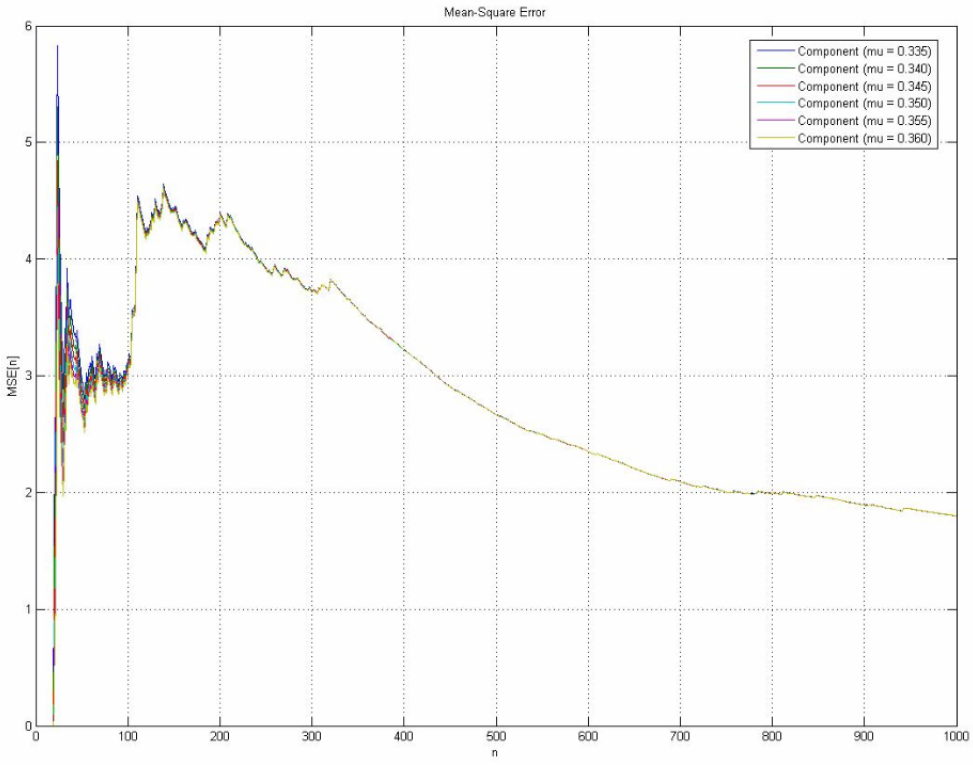

Figure 3.3: MSE for different values of $\widetilde{\mu}$

Several attempts were made to find the adaptation constant that produced the smallest MSE. The result of these trials indicated that the optimum adaptation constant was 0.345. Optimum, in this case, refers to the adaptation constant that achieves the minimum MSE over the duration of the simulation (particularly towards the end of the simulation).

Once this optimum adaptation constant, $\tilde{\mu}_{opt}$, was determined, several simulation cases were considered. In each case, the component NLMS filters utilized three taps. Adaptation constants were selected with the optimum value of 0.345 in mind. When such a combination filter is implemented in the real world, $\tilde{\mu}_{opt}$ will not be known. Therefore, the selection of $\tilde{\mu}_1$ and $\tilde{\mu}_2$ will be arbitrary to some degree. Categorizing the selection of $\tilde{\mu}_1$ and $\tilde{\mu}_2$ into the following cases will help understand the possible outcomes of a combination filter.

1. *Case 1*: $\tilde{\mu}_1 > \tilde{\mu}_{opt}$, $\tilde{\mu}_2 > \tilde{\mu}_{opt}$
2. *Case 2*: $\tilde{\mu}_1 < \tilde{\mu}_{opt}$, $\tilde{\mu}_2 < \tilde{\mu}_{opt}$
3. *Case 3.a*: $\tilde{\mu}_1 < \tilde{\mu}_{opt}$, $\tilde{\mu}_2 > \tilde{\mu}_{opt}$, $\left|\tilde{\mu}_{opt} - \tilde{\mu}_1\right| << \left|\tilde{\mu}_{opt} - \tilde{\mu}_2\right|$
4. *Case 3.b*: $\tilde{\mu}_1 < \tilde{\mu}_{opt}$, $\tilde{\mu}_2 > \tilde{\mu}_{opt}$, $\left|\tilde{\mu}_{opt} - \tilde{\mu}_1\right| \approx \left|\tilde{\mu}_{opt} - \tilde{\mu}_2\right|$

3.3 Experimental Results

Several trials were run with different adaptation constants for each NLMS filter. Trials in each of the aforementioned cases were conducted, and the results follow.

Case 1 – $\tilde{\mu}_1 > \tilde{\mu}_{opt}$, $\tilde{\mu}_2 > \tilde{\mu}_{opt}$

In this case, both component filters have an adaptation constant that is greater than the optimal value. Assume, without loss of generality, that $\tilde{\mu}_1 < \tilde{\mu}_2$. The adaptation constant of the first NLMS filter, therefore, is closest to the optimal value. As a result, the combination filter tended towards unity (i.e. $\lambda(\infty) = 1$), which means that the first component filter eventually became the only component of the combination. Furthermore, the performance of the combination approached that of the first component filter. In the long-run, the component filter would, at best, perform as good as the best component (filter one). Figure 3.3.1 shows the progression of the mixing parameter over time. Figure 3.3.2 shows the MSE for each component filter, as well as the combination.

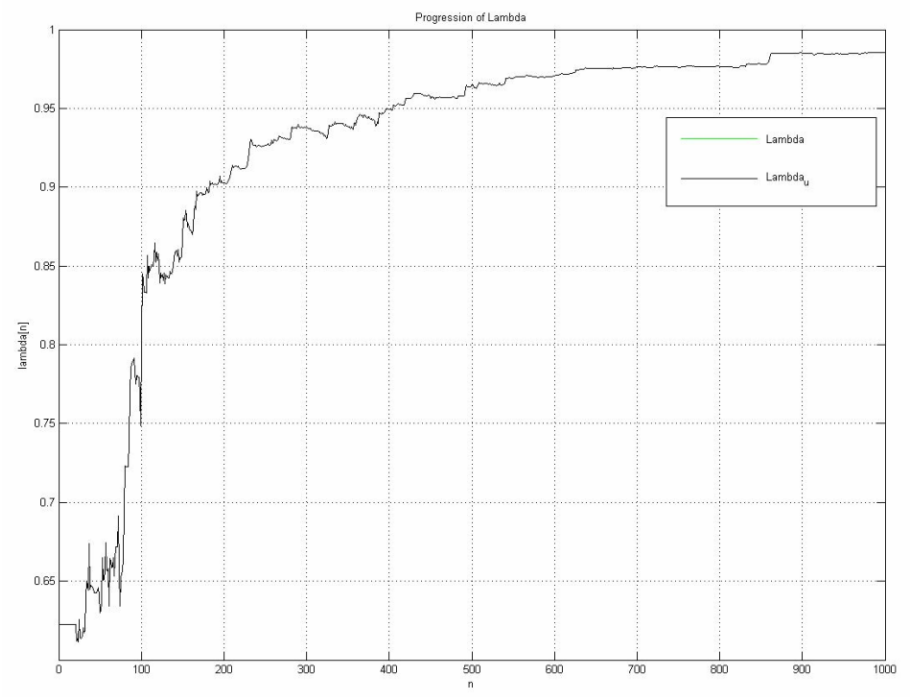

Figure 3.4: Progression of mixing parameter for Case 1

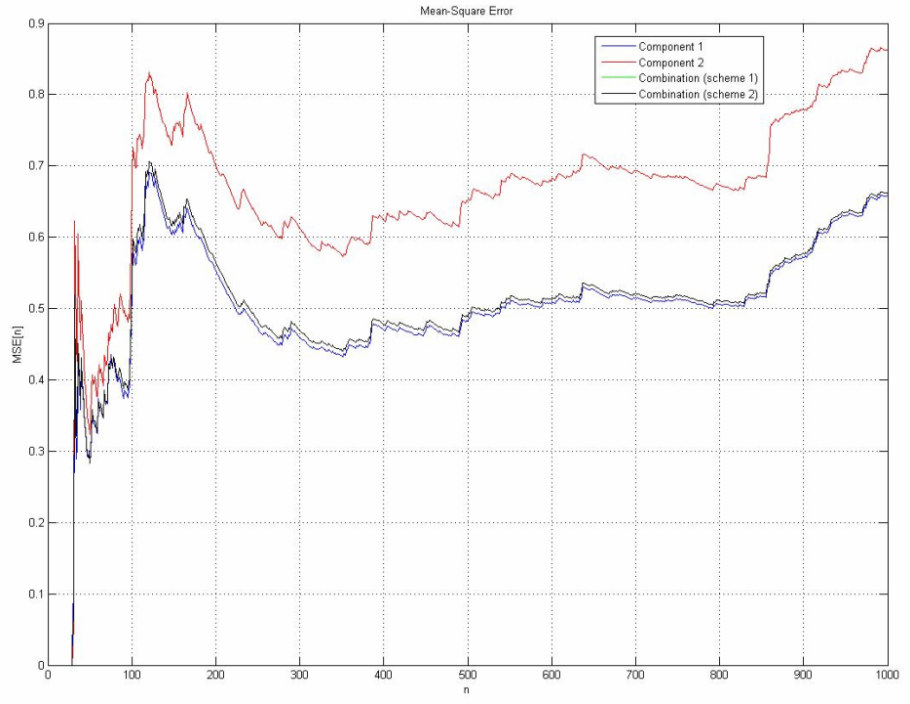

Figure 3.5: MSE for Case 1

Case 2 – $\widetilde{\mu}_1 < \widetilde{\mu}_{opt}$, $\widetilde{\mu}_2 < \widetilde{\mu}_{opt}$

In this case, both component filters have an adaptation constant that is less than the optimal value. Assume, without loss of generality, that $\widetilde{\mu}_1 > \widetilde{\mu}_2$. The adaptation constant of the first NLMS filter is again closest to the optimal value. As before, the combination filter tended

towards unity (i.e. $\lambda(\infty) = 1$). As a result, the performance of the combination approached that of the first component filter. Figure 3.3.3 below shows the progression of the mixing parameter over time. Figure 3.3.4 shows the MSE for each component filter, as well as the combination.

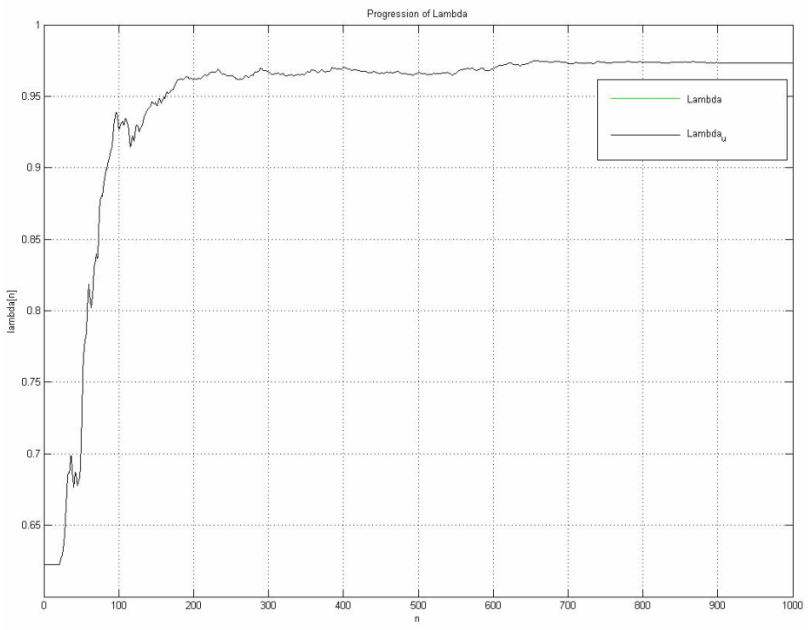

Figure 3.6: Progression of mixing parameter for Case 2

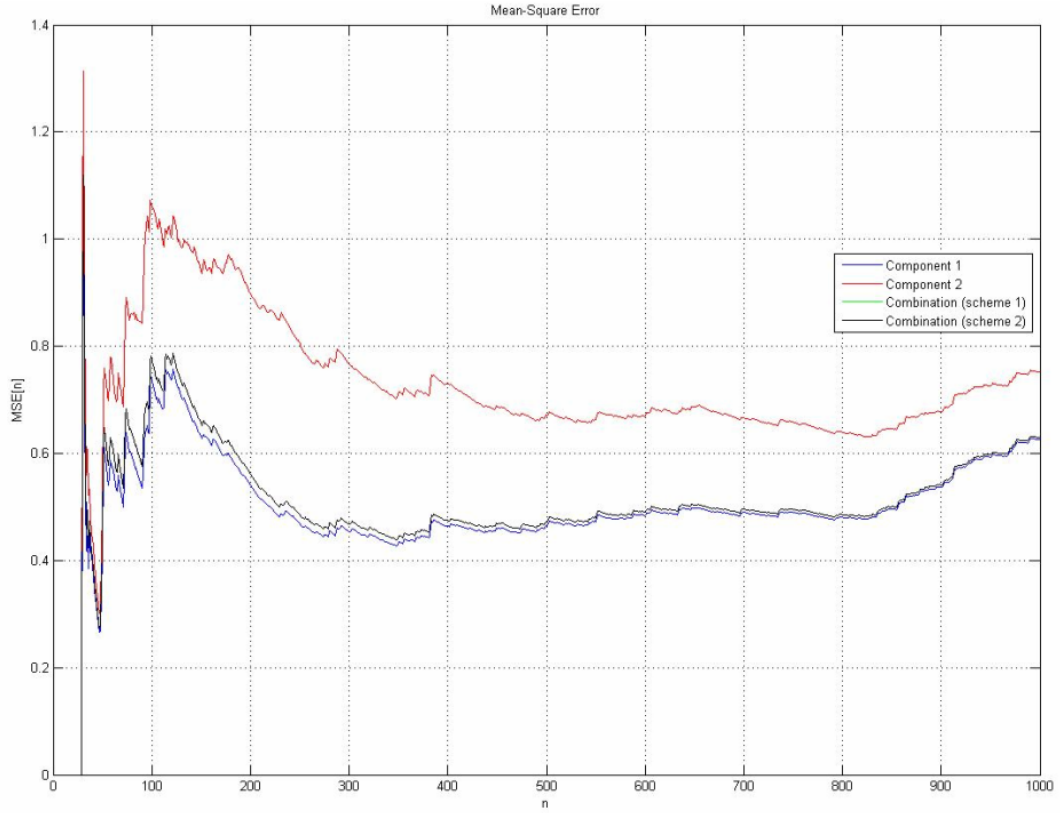

Figure 3.7: MSE for Case 2

Case 3.a – $\tilde{\mu}_1 < \tilde{\mu}_{opt}$, $\tilde{\mu}_2 > \tilde{\mu}_{opt}$, $\left|\tilde{\mu}_{opt} - \tilde{\mu}_1\right| << \left|\tilde{\mu}_{opt} - \tilde{\mu}_2\right|$

In this case, the component filter adaptation constants straddle the optimal value. Assume, without loss of generality, that $\left|\tilde{\mu}_{opt} - \tilde{\mu}_1\right| << \left|\tilde{\mu}_{opt} - \tilde{\mu}_2\right|$. The adaptation constant of the first NLMS filter is still closest to the optimal value. As before, the combination filter tended towards unity (i.e. $\lambda(\infty) = 1$), though in a less immediate manner; and the performance of the combination approached that of the first component filter. Figure 3.3.5 below shows the progression of the mixing parameter over time. Figure 3.3.6 shows the MSE for each component filter, as well as the combination.

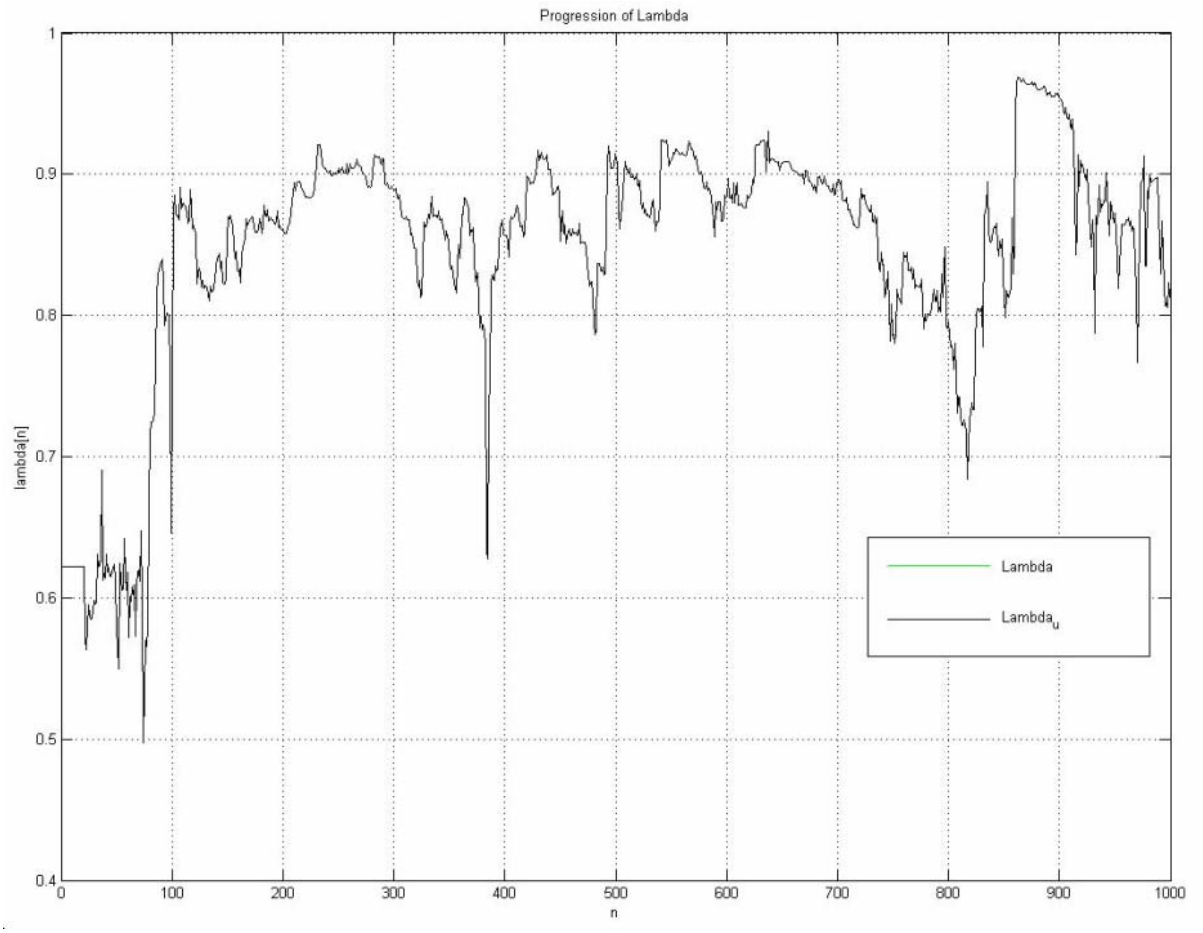

Figure 3.8: Progression of mixing parameter for Case 3.a

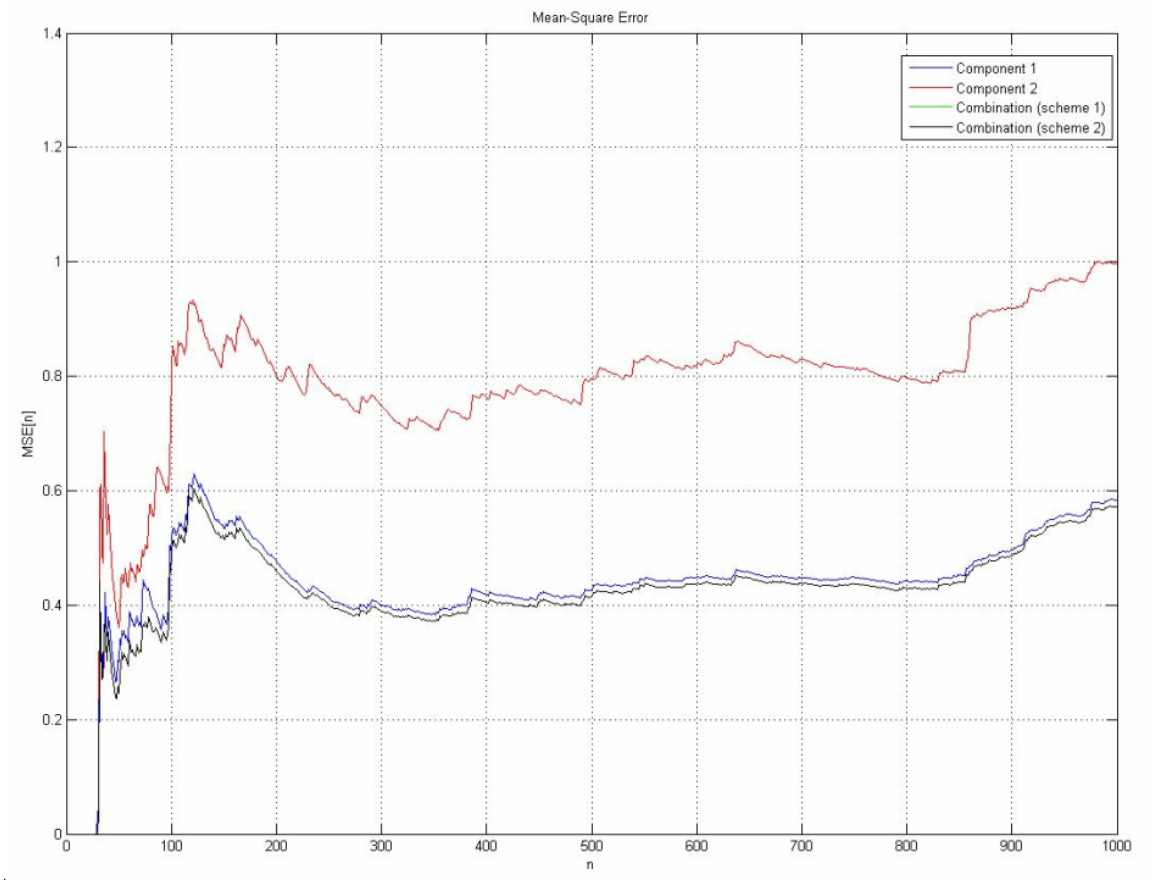

Figure 3.9: MSE for Case 3.a

Case 3.b – $\widetilde{\mu}_1 < \widetilde{\mu}_{opt}$, $\widetilde{\mu}_2 > \widetilde{\mu}_{opt}$, $\left|\widetilde{\mu}_{opt} - \widetilde{\mu}_1\right| \approx \left|\widetilde{\mu}_{opt} - \widetilde{\mu}_2\right|$

In this case, the component filter adaptation constants straddle the optimal value, and both are approximately the same "distance" away (in terms of their corresponding MSE performance). The adaptation constant of the first NLMS filter is no longer necessarily closer to the optimal value. Unlike the previous cases, the combination filter did not clearly favor one filter or the other. In fact, the mixing of the two components continued well into the simulation. This case meets the authors' criteria for universality. They assert that if the following is true,

$$-a^+ + \varepsilon < \mathrm{E}\{a(\infty)\} < a^+ - \varepsilon,$$

then both combination schemes yield a lower MSE than either of the components. Similarly, this experimental case shows that component mixing continues well into the simulation (and theoretically forever), and as a result, the combination outperforms the component filters. Figure 3.3.7 below shows the progression of the mixing parameter over time. Figure 3.3.8 shows the MSE for each component filter, as well as the combination.

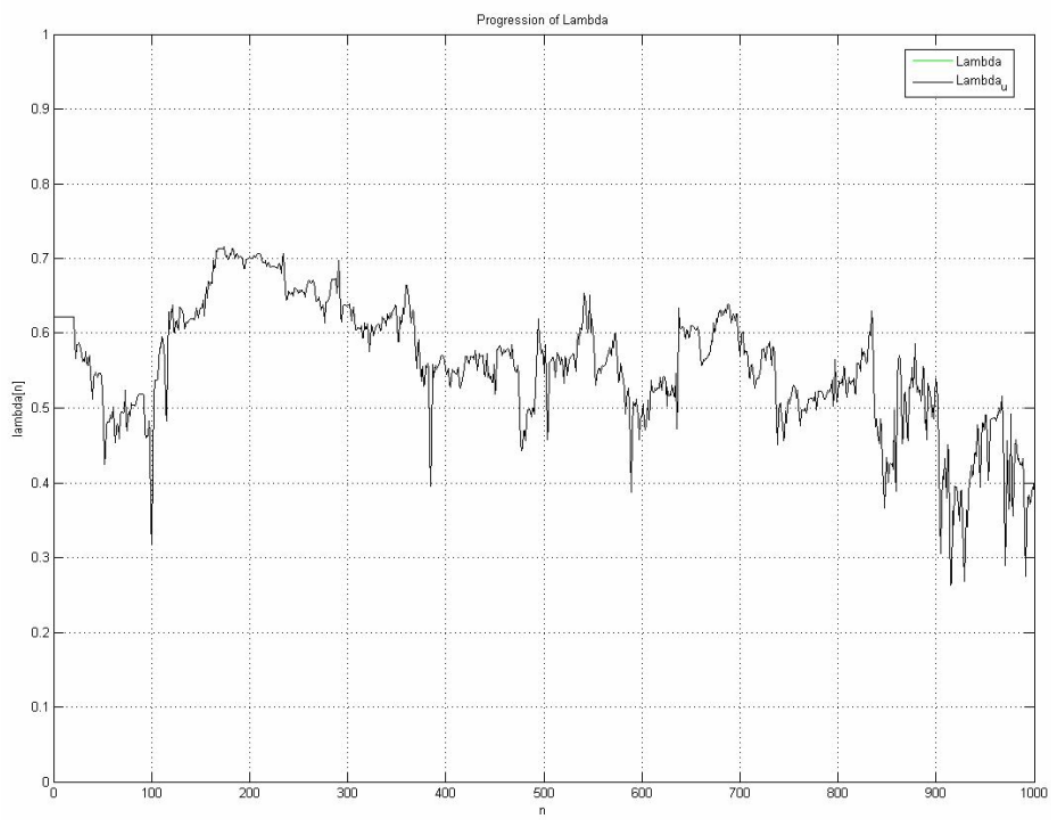

Figure 3.10: Progression of mixing parameter for Case 3.b

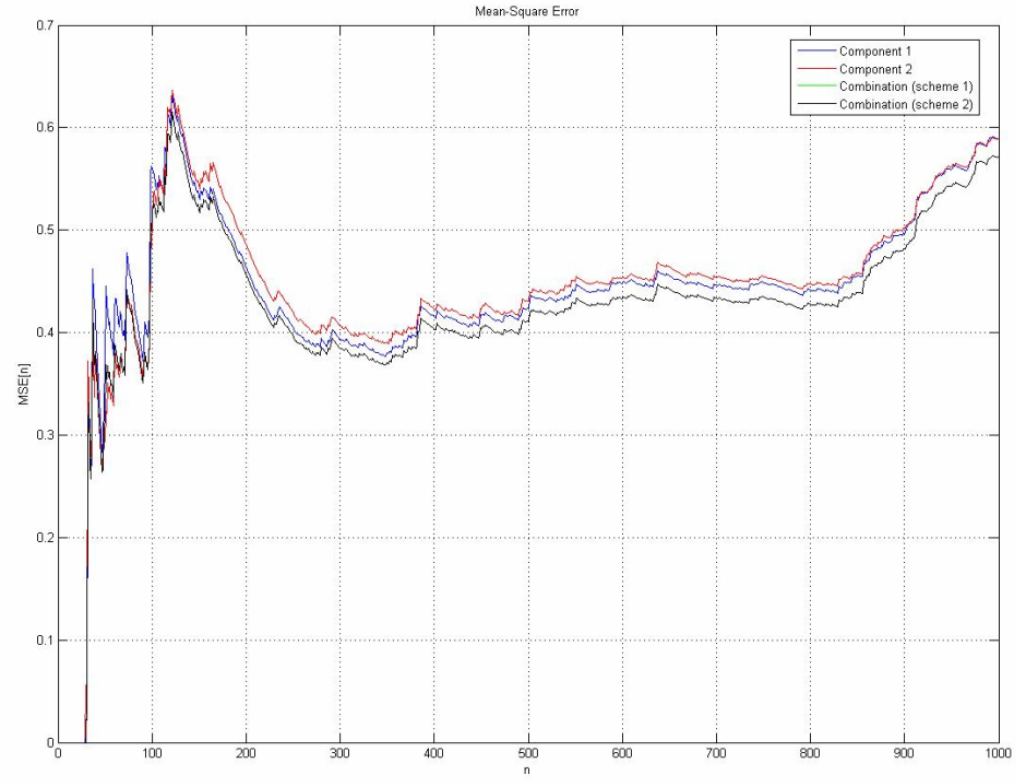

Figure 3.11: MSE for Case 3.b

4 Conclusions

After implementing the authors' proposed filter design with two NLMS filters, and using economic data as the input sequence, I was able to verify the most important claims of the authors. These are:

- If the mixing parameter approaches a steady-state value of either unity or zero, then the combination filter will perform equivalent to the corresponding component filter.
- If the mixing parameter does not approach a steady-state value of either unity or zero, then the combination filter outperforms *both* component filters in steady-state.

Furthermore, these conclusions offer some insight into the selection of adaptation constants for NLMS filters. If it is the case that the combination filter approaches one of the component filters, then this indicates that the corresponding component filter has a "more optimal" adaptation constant for the given input sequence. Likewise, if the combination filter does not tend towards one component or the other, then this indicates that the optimal adaptation constant lies somewhere in between the adaptation constants of the component filters.

After conducting exhaustive simulations with different parameters, I determined that this particular filter structure is not well-suited for the prediction of semi-stationary economic data. Although the MSE seemed to be fairly low, the percent error, as shown in Figure 4.1, was less than desirable. It is likely that a well-educated economist could form day-to-day guesses that result in a smaller percent error.

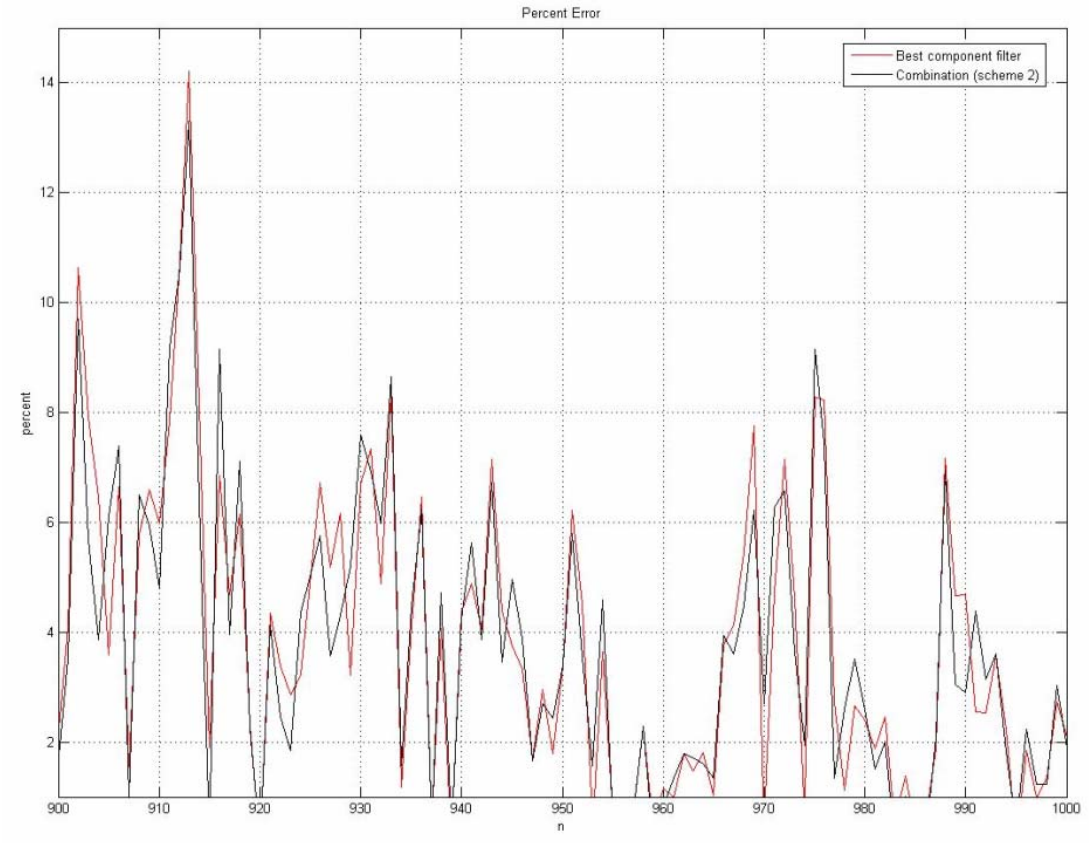

Figure 4.1: Percent error for combination and best component

5 References

[1] Arenas-García, J., Figueiras-Vidal, A., and Sayed, A., "Mean-Square Performance of a Convex Combination of Two Adaptive Filters," *IEEE Transactions on Signal Processing*, vol. 54, no. 3, pp. 1078-1090, March 2006.

[2] Haykin, Simon, *Adaptive Filter Theory*, 4th ed. Upper Saddle River, NJ: Pearson, 2002, pp. 320-343.

[3] Wikipedia contributors, "Least mean squares filter," *Wikipedia, The Free Encyclopedia,* http://en.wikipedia.org/w/index.php?title=Least_mean_squares_filter&oldid=49906745 (accessed April 27, 2006).

Appendix A: Matlab Code

main.m

```
function main(sim_type, I, O, k_1, k_2)

%-------------------------------------------------------------------------
% Function:    main()
% File name:   main.m
% Description: Main function that handles setup, simulation, and data
%              plotting.
%
% Notes:
%              w(n+1) = w(n) + mu(n)[p - R*w(n)]
%              J(n+1) = E{|e(n+1)|^2}
%              e(n+1) = d(n+1) - wH(n+1)*x(n+1)
%              w(n+1) = w(n) + (mu~/(a+mu(n)))x(n)e'(n)
%-------------------------------------------------------------------------

% Identify Global variables

% System-level parameters
global numIterations;
global filterOrder;
global closing_price;
global input;
global maxDataIndex;
global minDataIndex;
global predicted;
global desired;
global error;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global w1 w2;
global predicted1 predicted2;
global error1 error2;
global k1 k2;
global alpha;
global mu1 mu2;
global MSE1 MSE2;
global alpha;

% Other variables
global stationary nonstationary;

% Data boundary definitions
global stationary_min stationary_max;
global nonstat_min    nonstat_max;

% Initialize data boundaries
stationary_min = 1;
stationary_max = 1000;
nonstat_min    = 3001;
nonstat_max    = 4000;
```

```
% Set global variables
filterOrder   = O;
numIterations = I;
k1            = k_1;
k2            = k_2;
stationary    = 1;
nonstationary = 2;

if (sim_type == stationary)
    maxDataIndex  = stationary_min + filterOrder + numIterations - 1;
    minDataIndex  = stationary_min;
elseif (sim_type == nonstationary)
    maxDataIndex  = nonstat_min + filterOrder + numIterations - 1;
    minDataIndex  = nonstat_min;
else
    error('Error (main.m): Invalid simulation type');
end;

% Initialize environment
initialize_environment();

% Import price change data
temp_closing_price = xlsread('txn_data.xls');
%temp_closing_price = xlsread('AR3_data.xls');

% Resize data
closing_price = temp_closing_price(:,1);

% Plot input data
%plotInputData();

% Resize price change data
input = closing_price(minDataIndex:maxDataIndex);

% Desired(n) = input(n+1)
if (sim_type == stationary)
    desired = [input(2:(stationary_max-stationary_min+1)) ;
closing_price(maxDataIndex+1)];
elseif (sim_type == nonstationary)
    desired = [input(2:(nonstat_max-nonstat_min+1)) ;
closing_price(maxDataIndex+1)];
else
    error('Error (main.m): Invalid simulation type');
end;
closing_price(size(closing_price))];

% Simulate new method
simulate();

% Plot data
plotData();
```

initialize_environment.m

```
function initialize_environment()

%--------------------------------------------------------------------------
```

```
% Function:    initialize_environment()
% File name:   initialize_environment.m
% Description: Initialization function that zeros out all matrices/vectors
%-------------------------------------------------------------------------

% Identify Global variables

% System-level parameters
global numIterations;
global filterOrder;
global pc_data;
global input;
global maxDataIndex;
global minDataIndex;
global predicted_scheme1 predicted_scheme2;
global error_scheme1 error_scheme2;
global percent_e_scheme1 percent_e_scheme2;
global MSE_scheme1 MSE_scheme2;
global MSE_generic;
global desired;
global error;
global error_generic;
global w_generic;
global scheme1 scheme2;
global lambda lambda_u;
global a a_u;
global a_plus;
global epsilon;
global best_percent_error;
global best_MSE;
global mu_a;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global w1 w2;
global predicted1 predicted2;
global error1 error2;
global percent_error1 percent_error2;
global k1 k2;
global alpha;
global mu1 mu2;
global muc;
global MSE1 MSE2;
global alpha;

% Initialize constants
alpha   = 0.5;
scheme1 = 1;
scheme2 = 2;
%a_plus  = 1.5;
a_plus  = 5;
epsilon = 0.001;
mu_a    = 1.6;

% Initialize matrices/vectors
predicted      = zeros(numIterations);
predicted1     = zeros(numIterations);
```

```
predicted2     = zeros(numIterations);
w1             = zeros(filterOrder,numIterations);
w2             = zeros(filterOrder,numIterations);
w_generic      = zeros(10,filterOrder,1000);
error1         = zeros(numIterations);
error2         = zeros(numIterations);
percent_error1 = zeros(numIterations);
percent_error2 = zeros(numIterations);
error_generic  = zeros(10,1000);
MSE_generic    = zeros(10,1000);
mu1            = zeros(numIterations);
mu2            = zeros(numIterations);
error          = zeros(numIterations);
lambda         = zeros(numIterations);
lambda(1)      = 0.5;
lambda_u       = zeros(numIterations);
lambda_u(1)    = 0.5;
a              = zeros(numIterations);
a(1)           = 0.5;
a_u            = zeros(numIterations);
a_u(1)         = 0.5;
MSE_scheme1    = zeros(numIterations);
MSE_scheme2    = zeros(numIterations);
error_scheme1  = zeros(numIterations);
error_scheme2  = zeros(numIterations);
percent_e_scheme1  = zeros(numIterations);
percent_e_scheme2  = zeros(numIterations);
best_percent_error = zeros(numIterations);
best_MSE           = zeros(numIterations);
```

simulate.m

```
function simulate()

%-------------------------------------------------------------------------
% Function:    simulate()
% File name:   simulate.m
% Description: Simulation function that handles component filters,
%              combination filters, and performance metric gathering.
%
% Notes:
%-------------------------------------------------------------------------

% Identify Global variables

% System-level parameters
global numIterations;
global filterOrder;
global pc_data;
global input;
global maxDataIndex;
global minDataIndex;
global predicted;
global desired;
global error;
global scheme1 scheme2;
global percent_error1 percent_error2;
```

```
global best_percent_error;
global best_MSE;
global a a_u;
global mu_a;
global error_scheme1 error_scheme2;
global lambda lambda_u;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global w1 w2;
global predicted1 predicted2;
global error1 error2;
global k1 k2;
global alpha;
global mu1 mu2;
global muc;
global MSE1 MSE2;
global alpha;

% Data boundary definitions
global stationary_min stationary_max;
global nonstat_min    nonstat_max;

% Set global variable
maxN = numIterations;

% Simulate iteratively
for n = 1:maxN
    % Simulate component filter 1
    component_filter(1,n);

    % Simulate component filter 2
    component_filter(2,n);

    % Record best-performing component filter
    best_MSE(n)           = min(MSE1(n),MSE2(n));
    best_percent_error(n) = min(percent_error1(n),percent_error1(n));

    % Compute mixing parameter for combination scheme 1
    update_mixing(scheme1,n);

    % Compute mixing parameter for combination scheme 2
    update_mixing(scheme2,n);

    % Simulate combination filter
    combination_filter(n);

    % Update adaptation variable
    a(n+1)   = a(n)   + mu_a*error_scheme1(n)*(predicted1(n) -
predicted2(n))*lambda(n)*(1 - lambda(n));
    a_u(n+1) = a_u(n) + mu_a*error_scheme2(n)*(predicted1(n) -
predicted2(n))*lambda_u(n)*(1 - lambda_u(n));
end;
```

component_filter.m

```
function component_filter(filterNumber,n)
```

```
%-------------------------------------------------------------------------
% Function:    component_filter()
% File name:   component_filter.m
% Description: Simulation function that handles component filters.
%
% Notes:
%-------------------------------------------------------------------------

% Identify Global variables

% System-level parameters
global numIterations;
global filterOrder;
global pc_data;
global input;
global maxDataIndex;
global minDataIndex;
global predicted;
global desired;
global error;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global w1 w2;
global wo1 wo2;
global EMSE1 EMSE2;
global predicted1 predicted2;
global error1 error2;
global percent_error1 percent_error2;
global k1 k2;
global alpha;
global mu1 mu2;
global muc;
global MSE1 MSE2;

% Identify local variables
w         = zeros(filterOrder,1);      % Current tap weight vector
e         = 0;                         % Current error: desired - predicted
percent_e = 0;                         % Current percent error
MU        = 0.5;                         % Current step size

% Determine which filter to simulate
if (filterNumber == 1)
    % Set variables for component filter number 1
    w = w1(:,n);                % Use previously-calculated weight vector
    k = k1;                     % Use step-size parameter set at top-level
elseif (filterNumber == 2)
    % Set variables for component filter number 2
    w = w2(:,n);                % Use previously-calculated weight vector
    k = k2;                     % Use step-size parameter set at top-level
else
    error('Error (component_filter.m): filterNumber not in the expected range
[1,2].');
end;

% Simulate component filter
```

```
if (n < filterOrder)
    % Cannot simulate unles input is at least as large as the filter order
    y      = 0;
    wNext  = zeros(filterOrder,1);
    u      = zeros(filterOrder,1);
    wo     = zeros(filterOrder,1);
else
    % Define input sequence
    u = input(n:-1:n-filterOrder+1);

    % Calculate prediction
    y = w' * u;

    % Calculate errors in prediction
    e  = desired(n) - y;
    if (desired(n) == 0)
        percent_e = 0;
    else
        percent_e = 100*abs(desired(n) - y)/abs(desired(n));
    end;

    % Start with big mu for speeding the convergence then slow down to reach
the correct weights
    if n < 20
        MU = 0.5;
    else
        % Use adaptive step to reach the solution faster mu = k * 2/M*r(0)
        MU = k/(alpha+var(u));
    end

    % Calculate next tap weight vector
    wNext = w + ((MU * e)/(alpha + u'*u)) * u;
end;

% Update metrics/outputs for this iteration
if (filterNumber == 1)
    error1(n)         = e;
    percent_error1(n) = percent_e;
    predicted1(n)     = y;
    w1(:,n+1)         = wNext;
    mu1(n)            = MU;
    if (n < 30)
        MSE1(n)       = 0;
    else
        MSE1(n)       = mean((error1(30:n)).^2);
    end;
elseif (filterNumber == 2)
    error2(n)         = e;
    percent_error2(n) = percent_e;
    predicted2(n)     = y;
    w2(:,n+1)         = wNext;
    mu2(n)            = MU;
    if (n < 30)
        MSE2(n)       = 0;
    else
        MSE2(n)       = mean((error2(30:n)).^2);
    end;
```

```
else
    % Error case handles above
end;
```

update_mixing.m

```
function update_mixing(scheme,n)

%-------------------------------------------------------------------------
% Function:    update_mixing(scheme,n)
% File name:   update_mixing.m
% Description: This funtion updates the value of the mixing parameter
%              lambda.
%
% Notes:
%-------------------------------------------------------------------------

% Identify Global variables

% System-level parameters
global numIterations;
global filterOrder;
global closing_price;
global input;
global maxDataIndex;
global minDataIndex;
global predicted;
global desired;
global error;
global scheme1 scheme2;
global a a_u;
global a_plus;
global epsilon;
global mu_a;
global lambda lambda_u;
global error_scheme1 error_scheme2;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global w1 w2;
global predicted1 predicted2;
global error1 error2;
global k1 k2;
global alpha;
global mu1 mu2;
global muc;
global MSE1 MSE2;
global alpha;

% Identify local variables


if (scheme == scheme1)
    lambda(n) = 1/(1 + exp(-a(n)));
elseif  (scheme == scheme2)
    if (a_u(n) <= -1*a_plus + epsilon)
        lambda_u(n) = 0;
```

```
    elseif (a_u(n) >= a_plus - epsilon)
        lambda_u(n) = 1;
    else
        lambda_u(n) = lambda(n);
    end;
else
    error('Error (update_mixing): Invalid scheme selected (not in the set
{scheme1, scheme2}).');
end;
```

combination_filter.m

```
function combination_filter(n)

%-------------------------------------------------------------------------
% Function:    combination_filter()
% File name:   combination_filter.m
% Description: Simulation function that handles the combination filter.
%
% Notes:
%-------------------------------------------------------------------------

% Identify Global variables

% System-level parameters
global numIterations;
global filterOrder;
global pc_data;
global input;
global maxDataIndex;
global minDataIndex;
global predicted_scheme1 predicted_scheme2;
global desired;
global error_scheme1 error_scheme2;
global percent_e_scheme1 percent_e_scheme2;
global scheme1 scheme2;
global lambda lambda_u;
global a;
global a_plus;
global epsilon;
global mu_a;
global MSE_scheme1 MSE_scheme2;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global wo w1 w2;
global EMSE;
global predicted1 predicted2;
global error1 error2;
global k1 k2;
global alpha;
global mu1 mu2;
global muc;
global MSE1 MSE2;
global alpha;

% Compute the predictions at time n
```

```
predicted_scheme1(n) = lambda(n)*predicted1(n)   + (1 - lambda(n)
)*predicted2(n);
predicted_scheme2(n) = lambda_u(n)*predicted1(n) + (1 -
lambda_u(n))*predicted2(n);

% Compute the errors at time n
error_scheme1(n) = desired(n) - predicted_scheme1(n);
error_scheme2(n) = desired(n) - predicted_scheme2(n);

% Comute the percent error at time n
if (desired(n) == 0)
    percent_e_scheme1(n) = 0;
    percent_e_scheme2(n) = 0;
else
    percent_e_scheme1(n) = 100*abs(desired(n) -
predicted_scheme1(n))/abs(desired(n));
    percent_e_scheme2(n) = 100*abs(desired(n) -
predicted_scheme2(n))/abs(desired(n));
end;

% Update metrics for this iteration
if (n < 30)
    MSE_scheme1(n) = 0;
    MSE_scheme2(n) = 0;
else
    MSE_scheme1(n) = mean((error_scheme1(30:n)).^2);
    MSE_scheme2(n) = mean((error_scheme2(30:n)).^2);
end;
```

plot_data.m

```
function plotData(null)

%--------------------------------------------------------------------------
% Function:    plotData()
% File name:   plotData.m
% Description:
%
% Notes:
%--------------------------------------------------------------------------

% System-level parameters
global numIterations;
global filterOrder;
global pc_data;
global input;
global maxDataIndex;
global minDataIndex;
global predicted_scheme1 predicted_scheme2;
global desired;
global error_scheme1 error_scheme2;
global percent_e_scheme1 percent_e_scheme2;
global MSE_scheme1 MSE_scheme2;
global generic_MSE;
global generic_error;
global best_percent_error;
global best_MSE;
```

```
global lambda lambda_u;

% Filter-level parameters
global mu_tilde1 mu_tilde2;
global w1 w2;
global wo1 wo2;
global EMSE1 EMSE2;
global predicted1 predicted2;
global error1 error2;
global percent_error1 percent_error2;
global k1 k2;
global alpha;
global mu1 mu2;
global muc;
global MSE1 MSE2;

% Data plots
%n = minDataIndex:maxDataIndex;
n = 1:numIterations;
%{
% Input sequence
subplot(2,1,1);
plot(n,input(n)); grid on;
xlabel('n');
ylabel('u[n]');
axis([1 numIterations 0 1]); axis 'auto y';
title('Input Sequence');
%}
%{
% Output sequence
subplot(1,1,1);
plot(n,desired(n),'m-',n,predicted_scheme1(n),'g-',n,predicted_scheme2(n),'k-
.'); grid on;
legend('desired','Combination (scheme 1)','Combination (scheme 2)')
xlabel('n');
ylabel('d[n], y1[n], y2[n]');
axis([1 numIterations 0 1]); axis 'auto y';
title('Desired and Predicted Output Sequences');
%}

% Mean-square error
%subplot(3,1,1);
figure(1);
plot(n,MSE1(n),'b-',n,MSE2(n),'r-',n,MSE_scheme1(n),'g-',n,MSE_scheme2(n),'k-
'); grid on;
legend('Component 1','Component 2','Combination (scheme 1)','Combination
(scheme 2)')
xlabel('n');
ylabel('MSE[n]');
title('Mean-Square Error');
%{
% Filter taps
%subplot(3,1,2);
figure(2);
if (filterOrder == 3)
    plot3(squeeze(w1(1,n)),squeeze(w1(2,n)),squeeze(w1(3,n)),'b-'); grid on;
hold on;
```

```
    plot3(squeeze(w2(1,n)),squeeze(w2(2,n)),squeeze(w2(3,n)),'r-'); hold off;
    legend('Component 1','Component 2')
    xlabel('w1');
    ylabel('w2');
    zlabel('w3');
elseif (filterOrder == 2)
    plot(squeeze(w1(1,n)),squeeze(w1(2,n)),'b-'); grid on; hold on;
    plot(squeeze(w2(1,n)),squeeze(w2(2,n)),'r-'); hold off;
    legend('Component 1','Component 2')
    xlabel('w1');
    ylabel('w2');
else
    plot(n,w1(1,n),'b-'); grid on; hold on;
    plot(n,w2(1,n),'r-'); hold off;
    legend('Component 1','Component 2')
    xlabel('n');
    ylabel('w1(n)');
end;
title('Progression of tap weights');

% MSE for component filters
figure(3);
plot(n,MSE1(n),'b-',n,MSE2(n),'r-'); grid on;
legend('Component 1','Component 2')
xlabel('n');
ylabel('MSE[n]');
%axis([0 1000 1 100]);
title('Mean-Square Error');

% MSE for nearly-universal combination filter and best component filter
figure(4);
plot(n,best_MSE(n),'m-',n,MSE_scheme1(n),'g-'); grid on;
legend('Best component filter','Combination (scheme 1)')
xlabel('n');
ylabel('MSE[n]');
%axis([0 1000 1 100]);
title('Mean-Square Error');

% MSE for universal combination filter and best component filter
figure(5);
plot(n,best_MSE(n),'m-',n,MSE_scheme2(n),'k-'); grid on;
legend('Best component filter','Combination (scheme 2)')
xlabel('n');
ylabel('MSE[n]');
%axis([0 1000 1 100]);
title('Mean-Square Error');
%}
% Lambda
figure(6);
plot(n,lambda(n),'g-',n,lambda_u(n),'k-'); grid on;
legend('Lambda','Lambda_u')
xlabel('n');
ylabel('lambda[n]');
axis([0 1000 0 1]);
title('Progression of Lambda');

p = 900:1000;
```

```
% Percent error for component filters
figure(3);
plot(p,percent_error1(p),'b-',p,percent_error2(p),'g-'); grid on;
legend('Component 1','Component 2')
xlabel('n');
ylabel('percent');
axis([900 1000 1 15]);
title('Percent Error');

% Percent error for nearly-universal combination filter and best component
filter
figure(4);
plot(p,best_percent_error(p),'r-',p,percent_e_scheme1(p),'k-'); grid on;
legend('Best component filter','Combination (scheme 1)')
xlabel('n');
ylabel('percent');
axis([900 1000 1 15]);
title('Percent Error');

% Percent error for universal combination filter and best component filter
figure(5);
plot(p,best_percent_error(p),'r-',p,percent_e_scheme2(p),'k-'); grid on;
legend('Best component filter','Combination (scheme 2)')
xlabel('n');
ylabel('percent');
axis([900 1000 1 15]);
title('Percent Error');

%{
% Progression of mu
%subplot(3,1,3);
figure(3);
plot(n,mu1(n),'b-'); grid on; hold on;
plot(n,mu2(n),'r-'); hold off;
legend('Component 1','Component 2')
xlabel('n');
ylabel('mu(n)');
title('Progression of mu1 and mu2');
%}
```