Adaptive Filters

Adaptive Filters

A Tutorial for the Course *Computational Intelligence*

http://www.igi.tugraz.at/lehre/CI

Christian Feldbauer, Franz Pernkopf, and Erhard Rank
Signal Processing and Speech Communication Laboratory
Inffeldgasse 16c

Abstract

This tutorial introduces the LMS (least mean squares) and the RLS (recursive least-squares) algorithm for the design of adaptive transversal filters. These algorithms are applied for identification of an unknown system.

Usage

To make full use of this tutorial you have to

1. Download the file `AdaptiveFilter.zip` which contains this tutorial and the accompanying Matlab program(s).
2. Unzip `AdaptiveFilter.zip` which will generate a subdirectory `AdaptiveFilter/matlab` where you can find the Matlab program(s).
3. Add the path `AdaptiveFilter/matlab` to the matlab search path with a command like `addpath('C:\Work\Ada` if you are using a Windows machine or `addpath('/home/jack/AdaptiveFilter/matlab')` if you are using a Unix/Linux machine.

1 Introduction

Discrete-time (or digital) filters are ubiquitous in todays signal processing applications. Filters are used to achieve desired spectral characteristics of a signal, to reject unwanted signals, like noise or interferers, to reduce the bit rate in signal transmission, etc.

The notion of making filters adaptive, i.e., to alter parameters (coefficients) of a filter according to some algorithm, tackles the problems that we might not in advance know, e.g., the characteristics of the signal, or of the unwanted signal, or of a systems influence on the signal that we like to compensate. Adaptive filters can adjust to unknown environment, and even track signal or system characteristics varying over time.

2 Adaptive Transversal Filters

In a transversal filter of length N, as depicted in fig. 1, at each time n the output sample $y[n]$ is computed by a weighted sum of the current and delayed input samples $x[n], x[n-1], \ldots$

$$y[n] = \sum_{k=0}^{N-1} c_k^*[n] x[n-k]. \tag{1}$$

Here, the $c_k[n]$ are time dependent filter coefficients (we use the complex conjugated coefficients $c_k^*[n]$ so that the derivation of the adaption algorithm is valid for complex signals, too). This equation re-written

in vector form, using $\mathbf{x}[n] = \big[x[n], x[n-1], \ldots, x[n-N+1]\big]^{\mathsf{T}}$, the tap-input vector at time n, and $\mathbf{c}[n] = \big[c_0[n], c_1[n], \ldots, c_{N-1}[n]\big]^{\mathsf{T}}$, the coefficient vector at time n, is

$$y[n] = \mathbf{c}^{\mathsf{H}}[n]\mathbf{x}[n]. \tag{2}$$

Both $\mathbf{x}[n]$ and $\mathbf{c}[n]$ are column vectors of length N, $\mathbf{c}^{\mathsf{H}}[n] = (\mathbf{c}^*)^{\mathsf{T}}[n]$ is the hermitian of vector $\mathbf{c}[n]$ (each element is conjugated *, and the column vector is transposed $^{\mathsf{T}}$ into a row vector).

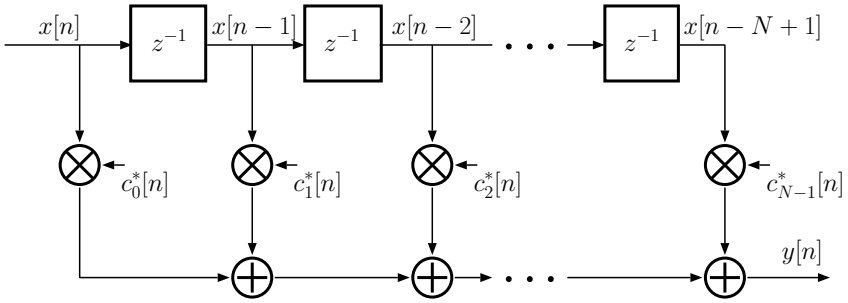

Figure 1: Transversal filter with time dependent coefficients

In the special case of the coefficients $\mathbf{c}[n]$ not depending on time n: $\mathbf{c}[n] = \mathbf{c}$ the transversal filter structure is an FIR[1] filter of length N. Here, we will, however, focus on the case that the filter coefficients are variable, and are adapted by an adaptation algorithm.

3 The LMS Adaptation Algorithm

The LMS (least mean squares) algorithm is an approximation of the steepest descent algorithm which uses an instantaneous estimate of the gradient vector of a cost function. The estimate of the gradient is based on sample values of the tap-input vector and an error signal. The algorithm iterates over each coefficient in the filter, moving it in the direction of the approximated gradient [1].

For the LMS algorithm it is necessary to have a reference signal $d[n]$ representing the desired filter output. The difference between the reference signal and the actual output of the transversal filter (eq. 2) is the error signal

$$e[n] = d[n] - \mathbf{c}^{\mathsf{H}}[n]\mathbf{x}[n]. \tag{3}$$

A schematic of the learning setup is depicted in fig. 2.

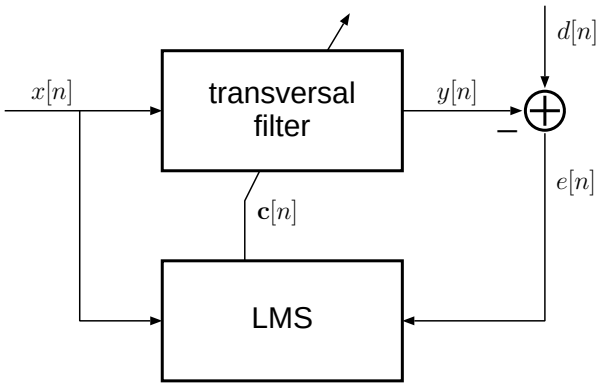

Figure 2: Adaptive transversal filter learning

The task of the LMS algorithm is to find a set of filter coefficients $\mathbf{c}$ that minimize the expected value of the quadratic error signal, i.e., to achieve the least mean squared error (thus the name). The squared

[1] finite impulse response

error and its expected value are (for simplicity of notation and perception we drop the dependence of all variables on time n in eqs. 4 to 7)

$$e^2 = \left(d - \mathbf{c}^{\mathsf{H}}\mathbf{x}\right)^2 = d^2 - 2d\,\mathbf{c}^{\mathsf{H}}\mathbf{x} + \mathbf{c}^{\mathsf{H}}\mathbf{x}\,\mathbf{x}^{\mathsf{H}}\mathbf{c}, \tag{4}$$

$$\begin{aligned} \mathrm{E}(e^2) &= \mathrm{E}(d^2) - \mathrm{E}(2d\,\mathbf{c}^{\mathsf{H}}\mathbf{x}) + \mathrm{E}(\mathbf{c}^{\mathsf{H}}\mathbf{x}\,\mathbf{x}^{\mathsf{H}}\mathbf{c}) \\ &= \mathrm{E}(d^2) - \mathbf{c}^{\mathsf{H}}\,2\,\mathrm{E}(d\mathbf{x}) + \mathbf{c}^{\mathsf{H}}\;\mathrm{E}(\mathbf{x}\,\mathbf{x}^{\mathsf{H}})\,\mathbf{c} \end{aligned} \tag{5}$$

Note, that the squared error e^2 is a quadratic function of the coefficient vector $\mathbf{c}$, and thus has only one (global) minimum (and no other (local) minima), that theoretically could be found if the correct expected values in eq. 5 were known.

The gradient descent approach demands that the position on the error surface according to the current coefficients should be moved into the direction of the 'steepest descent', i.e., in the direction of the negative gradient of the *cost function* $J = \mathrm{E}(e^2)$ with respect to the coefficient vector[2]

$$-\nabla_{\mathbf{c}} J = 2\,\mathrm{E}(d\,\mathbf{x}) - 2\,\mathrm{E}(\mathbf{x}\,\mathbf{x}^{\mathsf{H}})\mathbf{c}. \tag{6}$$

The expected values in this equation, $\mathrm{E}(d\,\mathbf{x}) = \mathbf{p}$, the cross-correlation vector between the desired output signal and the tap-input vector, and $\mathrm{E}(\mathbf{x}\,\mathbf{x}^{\mathsf{H}}) = \mathbf{R}$, the auto-correlation matrix of the tap-input vector, would usually be estimated using a large number of samples from d and $\mathbf{x}$. In the LMS algorithm, however, a very short-term estimate is used by only taking into account the current samples: $\mathrm{E}(d\,\mathbf{x}) \approx d\,\mathbf{x}$, and $\mathrm{E}(\mathbf{x}\,\mathbf{x}^{\mathsf{H}}) \approx \mathbf{x}\,\mathbf{x}^{\mathsf{H}}$, leading to an update equation for the filter coefficients

$$\begin{aligned} \mathbf{c}^{\mathrm{new}} &= \mathbf{c}^{\mathrm{old}} + \mu/2\;\;(-\nabla_{\mathbf{c}} J(\mathbf{c})) \\ &= \mathbf{c}^{\mathrm{old}} + \mu\,\mathbf{x}\,(d - \mathbf{x}^{\mathsf{H}}\mathbf{c}) \\ &= \mathbf{c}^{\mathrm{old}} + \mu\,\mathbf{x}\,e^*. \end{aligned} \tag{7}$$

Here, we introduced the 'step-size' parameter μ, which controls the distance we move along the error surface. In the LMS algorithm the update of the coefficients, eq. 7, is performed at every time instant n,

$$\mathbf{c}[n+1] = \mathbf{c}[n] + \mu\,e^*[n]\,\mathbf{x}[n]. \tag{8}$$

3.1 Choice of step-size

The 'step-size' parameter μ introduced in eq. 7 and 8 controls how far we move along the error function surface at each update step. μ certainly has to be chosen $\mu > 0$ (otherwise we would move the coefficient vector in a direction towards larger squared error). Also, μ should not be too large, since in the LMS algorithm we use a local approximation of $\mathbf{p}$ and $\mathbf{R}$ in the computation of the gradient of the cost function, and thus the cost function at each time instant may differ from an accurate global cost function.

Furthermore, too large a step-size causes the LMS algorithm to be instable, i.e., the coefficients do not converge to fixed values but oscillate. Closer analysis [1] reveals, that the upper bound for μ for stable behavior of the LMS algorithm depends on the largest eigenvalue $\lambda_{\max}$ of the tap-input auto-correlation matrix $\mathbf{R}$ and thus on the input signal. For stable adaptation behavior the step-size has to be

$$\mu < \frac{2}{\lambda_{\max}}. \tag{9}$$

Since we still do not want to compute an estimate of $\mathbf{R}$ and its eigenvalues, we first approximate $\lambda_{\max} \approx \mathrm{tr}(\mathbf{R})$ ($\mathrm{tr}(\mathbf{R})$ is the *trace* of matrix $\mathbf{R}$, i.e., the sum of the elements on its diagonal), and then – in the same way as we approximated the expected values in the cost function – $\mathrm{tr}(\mathbf{R}) \approx \|\mathbf{x}[n]\|^2$, the tap-input power at the current time n. Hence, the upper bound for μ for stable behavior depends on the signal power[3].

[2] $\nabla_{\mathbf{c}}$ is the 'nabla' differential operator with respect to the vector $\mathbf{c}$: $\nabla_{\mathbf{c}} = \left[\frac{\partial}{\partial c_1}, \frac{\partial}{\partial c_2}, \ldots, \frac{\partial}{\partial c_d}\right]^{\mathsf{T}}$.

[3] The dependence of the stability bound on the signal power is exploited in the *normalized LMS algorithm* by normalizing the step-size according to the signal power: $\mathbf{c}[n+1] = \mathbf{c}[n] + \frac{\mu}{\epsilon + \|\mathbf{x}[n]\|^2}\,e^*[n]\,\mathbf{x}[n]$, with a small positive constant ϵ.

3.2 Summary of the LMS algorithm

1. Filter operation:
$$y[n] = \mathbf{c}^{\mathsf{H}}[n]\mathbf{x}[n]$$
2. Error calculation:
$$e[n] = d[n] - y[n]$$
where $d[n]$ is the desired output
3. Coefficient adaptation:
$$\mathbf{c}[n+1] = \mathbf{c}[n] + \mu\, e^*[n]\, \mathbf{x}[n]$$

4 The RLS Adaptation Algorithm

The RLS (recursive least squares) algorithm is another algorithm for determining the coefficients of an adaptive filter. In contrast to the LMS algorithm, the RLS algorithm uses information from all past input samples (and not only from the current tap-input samples) to estimate the (inverse of the) autocorrelation matrix of the input vector. To decrease the influence of input samples from the far past, a weighting factor for the influence of each sample is used. This weighting factor is introduced in the cost function

$$J[n] = \sum_{i=1}^{n} \rho^{n-i} |e[i,n]|^2, \tag{10}$$

where the error signal $e[i,n]$ is computed for all times $1 \le i \le n$ using the current filter coefficients $\mathbf{c}[n]$: $e[i,n] = d[i] - \mathbf{c}^{\mathsf{H}}[n]\mathbf{x}[i]$.

When $\rho = 1$ the squared error for all sample times i up to current time n is considered in the cost function J equally. If $0 < \rho < 1$ the influence of past error values decays exponentially: method of *exponentially weighted least squares*. ρ is called the *forgetting factor*.

Analogous to the derivation of the LMS algorithm we find the gradient of the cost function with respect to the current weights

$$\nabla_{\mathbf{c}} J[n] = \sum_{i=1}^{n} \rho^{n-i} \left(-2\,\mathrm{E}(d[i]\,\mathbf{x}[i]) + 2\,\mathrm{E}(\mathbf{x}[i]\,\mathbf{x}^{\mathsf{H}}[i])\mathbf{c}[n] \right). \tag{11}$$

We now, however, *do* trust in the ability to estimate the expected values $\mathrm{E}(d\,\mathbf{x}) = \mathbf{p}$ and $\mathrm{E}(\mathbf{x}\,\mathbf{x}^{\mathsf{H}}) = \mathbf{R}$ with sufficient accuracy using all past samples, and do not use a gradient descent method, but immediately search for the minimum of the cost function by setting its gradient to zero $\nabla_{\mathbf{c}} J[n] = 0$. The resulting equation for the optimum filter coefficients at time n is

$$\begin{aligned} \mathbf{\Phi}[n]\mathbf{c}[n] &= \mathbf{z}[n], \\ \mathbf{c}[n] &= \mathbf{\Phi}^{-1}[n]\,\mathbf{z}[n], \end{aligned} \tag{12}$$

with $\mathbf{\Phi}[n] = \sum_{i=1}^{n} \rho^{n-i}\mathbf{x}[i]\,\mathbf{x}^{\mathsf{H}}[i]$, and $\mathbf{z}[n] = \sum_{i=1}^{n} \rho^{n-i} d^*[i]\,\mathbf{x}[i]$.

Both $\mathbf{\Phi}[n]$ and $\mathbf{z}[n]$ can be computed recursively:

$$\mathbf{\Phi}[n] = \rho\,\mathbf{\Phi}[n-1] + \mathbf{x}[n]\,\mathbf{x}^{\mathsf{H}}[n], \tag{13}$$

and

$$\mathbf{z}[n] = \rho\,\mathbf{z}[n-1] + d^*[n]\,\mathbf{x}[n]. \tag{14}$$

To find the coefficient vector $\mathbf{c}[n]$ we, however, need the inverse matrix $\mathbf{\Phi}^{-1}[n]$. Using a matrix inversion lemma [1] a recursive update equation for $\mathbf{P}[n] = \mathbf{\Phi}^{-1}[n]$ is found as:

$$\begin{aligned} \mathbf{P}[n] &= \rho^{-1}\,\mathbf{P}[n-1] + \rho^{-1}\,\mathbf{k}[n]\,\mathbf{x}[n], \\ \text{with} \quad \mathbf{k}[n] &= \frac{\rho^{-1}\,\mathbf{P}[n-1]\,\mathbf{x}[n]}{1 + \rho^{-1}\,\mathbf{x}^{\mathsf{H}}[n]\,\mathbf{P}[n-1]\,\mathbf{x}[n]}. \end{aligned} \tag{15}$$

Finally, the weights update equation is

$$\mathbf{c}[n] = \mathbf{c}[n-1] + \mathbf{k}[n]\left(d^*[n] - \mathbf{x}^{\mathsf{H}}[n]\mathbf{c}[n-1]\right). \tag{16}$$

The equations to solve in the RLS algorithm at each time step n are eqs. 15 and 16. The RLS algorithm is computationally more complex than the LMS algorithm. Note, however, that due the recursive updating the inversion of matrix $\mathbf{\Phi}[n]$ is not necessary (which would be a considerably higher computational load). The RLS algorithm typically shows a faster convergence compared to the LMS algorithm.

5 Applications of Adaptive Filters

Two possible application scenarios of adaptive filters are given in fig. 3, system identification and inverse filtering. For system identification the adaptive filter is used to approximate an unknown system. Both the unknown system and the adaptive filter are driven by the same input signal and the adaptive filter coefficients are adjusted in a way, that the output signal resembles the output of the unknown system, i.e., the adaptive filter is used to approximate the unknown system.

For inverse modeling or equalization the adaptive filter is used in series with the unknown system and the learning algorithm tries to compensate the influence of the unknown system on the test signal $u[n]$ by minimizing the (squared) difference between the adaptive filters output and the delayed test signal.

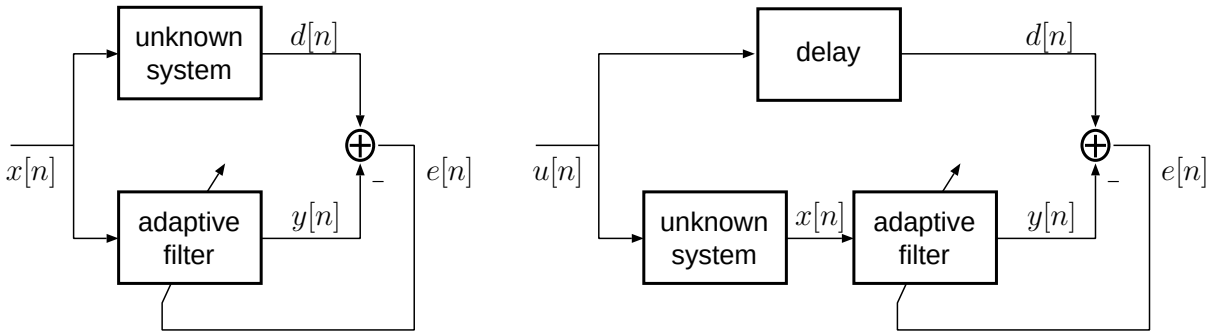

Figure 3: Two applications of adaptive filters: System identification (left) and inverse modeling/equalization (right)

Applications of adaptive filters further include the adaptive prediction of a signal, used for example in ADPCM[4] audio coding, adaptive noise or echo cancellation, and adaptive beam-forming (shaping of the acoustic/radio 'beam' transmitted/received by an array of loudspeakers/microphones/antennas).

References

[1] Simon Haykin: "Adaptive Filter Theory", Third Edition, Prentice-Hall, Inc., Upper Saddle River, NJ, 1996.

[2] Bernard Widrow and Samuel D. Stearns: "Adaptive Signal Processing", Prentice-Hall, Inc., Upper Saddle River, NJ, 1985.

[3] Edward A. Lee and David G. Messerschmitt: "Digital Communication", Kluwer Academic Publishers, Boston, 1988.

[4] Steven M. Kay: "Fundamentals of Statistical Signal Processing—Detection Theory", Volume 2, Prentice-Hall, Inc., 1998.

[4] adaptive differential pulse-code modulation

6 MATLAB Exercises

6.1 LMS Algorithm

Write a MATLAB function `y=lms1(x,d,N,mu)` which implements an adaptive transversal filter using the LMS algorithm. Use the following example file header as a starting point:

```
function y = lms1(x,d,N,mu)
% y = lms1(x,d,N,mu)
%   Adaptive transversal filter using LMS
% INPUT
%   x ... vector containing the samples of the input signal x[n]
%         size(x) = [xlen,1] ... column vector
%   d ... vector containing the samples of the desired output signal d[n]
%         size(d) = [xlen,1] ... column vector
%   N ... number of coefficients
%   mu .. step-size parameter
% OUTPUT
%   y ... vector containing the samples of the output signal y[n]
%         size(y) = [xlen,1] ... column vector
```

Test your function using the following setup:

$$x[n] = 2 \quad \text{for} \quad n = 0, 1, \ldots, 999 \qquad d[n] = 1 \quad \text{for} \quad n = 0, 1, \ldots, 999 \qquad N = 1$$

Try different values for μ. Plot $x[n]$, $y[n]$, and $d[n]$ into the same figure by executing `plot([x,y,d])` (note: `x`, `y`, and `d` should be column vectors).

6.2 Learning Curve

Often we are interested to see how the adaptation of the coefficients works or how the error behaves over time. Therefore we need the function `[y,e,c]=lms2(x,d,N,mu)` which provides us more output arguments. Extend the function from exercise 6.1 by the additional output arguments.

```
function [y,e,c] = lms2(x,d,N,mu)
% [y,e,c] = lms2(x,d,N,mu)
%   Adaptive transversal filter using LMS (for algorithm analysis)
% INPUT
%   x ... vector containing the samples of the input signal x[n]
%         size(x) = [xlen,1] ... column vector
%   d ... vector containing the samples of the desired output signal d[n]
%         size(d) = [xlen,1] ... column vector
%   N ... number of coefficients
%   mu .. step-size parameter
% OUTPUT
%   y ... vector containing the samples of the output signal y[n]
%         size(y) = [xlen,1] ... column vector
%   e ... vector containing the samples of the error signal e[n]
%         size(y) = [xlen,1] ... column vector
%   c ... matrix containing the coefficient vectors c[n]
%         size(c) = [N,xlen+1]
```

Test this function by applying the same input like in exercise 6.1 and plot the squared error $e^2[n]$ (*learning curve*). Use a logarithmic scale for the squared error in the learning curve.

Add a random signal to the desired output $d[n]$ (this would be the signal from a local speaker in an echo cancelation application):

```
>> dn = d + 1e-6*randn(length(d),1);
```

and plot the learning curve for the LMS algorithm using the noisy desired output signal. Compare to the learning curve found before. Do the coefficients converge?

6.3 System Identification

According to the system identification setup in fig. 3, investigate the ability of an adaptive filter to approximate an FIR filter (i.e., a transversal filter with fixed coefficients $\mathbf{h}$) of length M with an impulse response $\mathbf{h} = [h_0, h_1, \ldots, h_{M-1}]^{\mathsf{T}}$.

The desired (reference) output of the filter is

$$d[n] = \mathbf{h}^{\mathsf{H}}\mathbf{x}[n] = \sum_{k=0}^{M-1} h_k^* \, x[n-k].$$

For the unknown system, you can take any $\mathbf{h}$ and M you may wish. To calculate $d[n]$, use the MATLAB function `filter()`.

Use your function `lms2()` from exercise 6.2 and let $x[n]$ be normally distributed random numbers with mean zero and variance one (use the MATLAB function `randn()`). Choose a proper value for the step-size parameter μ.

Compare the output of the adaptive filter using the LMS algorithm to the desired signal and to the output of an adaptive filter using the RLS algorithm. The RLS algorithm is provided in the MATLAB file `rls1.m`. The forgetting factor ρ should be chosen between $0.95 \leq \rho \leq 1$.

```
function [y,e,c1]=rls1(x,d,N,rho)
%Adaptive transversal filter using RLS algorithm
%
%   [y,e,c1]=rls1(x,d,N,rho)
%
% input:
%   x   column vector containing the input samples x[n] (size(x)=[xlen,1])
%   d   column vector containing the samples of the desired output
%       signal d[n] (size(d)=[xlen,1])
%   N   number of coefficients
%   rho forgetting factor
%
% output:
%   y   column vector containing the samples of the
%       output y[n] (size(y)=[xlen,1])
%   e   column vector containing the samples of the
%       error signal e[n] (size(e)=[xlen,1])
%   c   matrix containing the coefficient vectors c[n]
%         size(c) = [N,xlen+1]
%
```

For the system identification application, write a MATLAB script to visualize the adaptation process in the time domain.

Compare the LMS and the RLS algorithms regarding the time until the coefficients converge.

Additionally, modify your script and examine the cases $N > M$ and $N < M$.

6.4 Coefficient Adaptation Behavior

For the two-dimensional case ($N = M = 2$), visualize the adaptation path in the $\mathbf{c}$-plane ($\mathbf{c}[n] = \Big[c_0[n], c_1[n]\Big]^{\mathsf{T}}$). Use both algorithms (`lms2()` and `rls1()`) and different input signals:

1. $x[n] = \text{sign}\left(\text{rand}[n] - 0.5\right) \quad \text{and} \quad \mu = 0.5$
2. $x[n] = \text{randn}[n] \quad \text{and} \quad \mu = 0.5$
3. $x[n] = \cos[\pi n] \quad \text{and} \quad \mu = 0.5$
4. $x[n] = \cos[\pi n] + 2 \quad \text{and} \quad \mu = 0.1$

Compare the results of this four input signals. Describe your observations.