Machine Learning

A Bayesian and Optimization Perspective

Academic Press, 2015

Sergios Theodoridis[1]

[1]Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece.

Spring 2015, Version I

Chapter 5
Stochastic Gradient Descent: The LMS and its Family

Online and Batch Processing Algorithms

- The focus in this series of lectures is to introduce online learning techniques for estimating the unknown parameter vector. These are time iterative schemes, which update the available estimate every time a measurement set (input-output pair of observations) is acquired.
- In contrast to the so-called batch processing methods, which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance.
- The fact that such learning algorithms work in a time iterative mode gives them the agility to learn and **track** slow time variations of the statistics of the involved processes/variables; this is the reason why these algorithms are also known as time-adaptive or simply adaptive, since they can adapt to the needs of a changing environment.
- More recently, the philosophy behind such schemes is gaining in popularity in the context of big data applications with massive number of data points that reside in large data bases, possibly distributed in various sites; for such tasks, storing all the data points for processing in the memory may not be possible and they have to be considered one at a time. Moreover, the complexity of block processing techniques can amount to prohibitive levels, for today's technology.

Online and Batch Processing Algorithms

- The focus in this series of lectures is to introduce online learning techniques for estimating the unknown parameter vector. These are time iterative schemes, which update the available estimate every time a measurement set (input-output pair of observations) is acquired.
- In contrast to the so-called batch processing methods, which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance.
- The fact that such learning algorithms work in a time iterative mode gives them the agility to learn and **track** slow time variations of the statistics of the involved processes/variables; this is the reason why these algorithms are also known as time-adaptive or simply adaptive, since they can adapt to the needs of a changing environment.
- More recently, the philosophy behind such schemes is gaining in popularity in the context of big data applications with massive number of data points that reside in large data bases, possibly distributed in various sites; for such tasks, storing all the data points for processing in the memory may not be possible and they have to be considered one at a time. Moreover, the complexity of block processing techniques can amount to prohibitive levels, for today's technology.

Online and Batch Processing Algorithms

- The focus in this series of lectures is to introduce online learning techniques for estimating the unknown parameter vector. These are time iterative schemes, which update the available estimate every time a measurement set (input-output pair of observations) is acquired.
- In contrast to the so-called batch processing methods, which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance.
- The fact that such learning algorithms work in a time iterative mode gives them the agility to learn and **track** slow time variations of the statistics of the involved processes/variables; this is the reason why these algorithms are also known as time-adaptive or simply adaptive, since they can adapt to the needs of a changing environment.
- More recently, the philosophy behind such schemes is gaining in popularity in the context of big data applications with massive number of data points that reside in large data bases, possibly distributed in various sites; for such tasks, storing all the data points for processing in the memory may not be possible and they have to be considered one at a time. Moreover, the complexity of block processing techniques can amount to prohibitive levels, for today's technology.

Online and Batch Processing Algorithms

- The focus in this series of lectures is to introduce online learning techniques for estimating the unknown parameter vector. These are time iterative schemes, which update the available estimate every time a measurement set (input-output pair of observations) is acquired.
- In contrast to the so-called batch processing methods, which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance.
- The fact that such learning algorithms work in a time iterative mode gives them the agility to learn and **track** slow time variations of the statistics of the involved processes/variables; this is the reason why these algorithms are also known as time-adaptive or simply adaptive, since they can adapt to the needs of a changing environment.
- More recently, the philosophy behind such schemes is gaining in popularity in the context of big data applications with massive number of data points that reside in large data bases, possibly distributed in various sites; for such tasks, storing all the data points for processing in the memory may not be possible and they have to be considered one at a time. Moreover, the complexity of block processing techniques can amount to prohibitive levels, for today's technology.

The Gradient Descent Method

- Our starting point is the method of gradient descent, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \mathbb{R}^l$. As any other iterative technique, the method starts from an initial estimate, $\boldsymbol{\theta}^{(0)}$, and generates a sequence, $\boldsymbol{\theta}^{(i)}$, $i = 1, 2, \ldots$, such that,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}, \ i > 0,$$

  where $\mu_i > 0$. Different rules for the choice of μ_i and $\Delta\boldsymbol{\theta}^{(i)}$ lead to different algorithms; the latter vector is known as the update direction or the search direction. The sequence μ_i is known as the step-size or the step-length, at the ith iteration; note that the values of μ_i may either be constant or change at each iteration.
- The choice of $\Delta\boldsymbol{\theta}^{(i)}$ is done such that to guarantee that

$$J(\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)}),$$

  except at a minimizer, θ_*.
- Assume that at the $i-1$ iteration step the value $\boldsymbol{\theta}^{(i-1)}$ has been obtained. Then, mobilizing a first order Taylor's expansion we can write

$$J\left(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}\right) \approx J(\boldsymbol{\theta}^{(i-1)}) + \mu_i \nabla^T J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}.$$

The Gradient Descent Method

- Our starting point is the method of gradient descent, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \mathbb{R}^l$. As any other iterative technique, the method starts from an initial estimate, $\boldsymbol{\theta}^{(0)}$, and generates a sequence, $\boldsymbol{\theta}^{(i)}$, $i = 1, 2, \ldots$, such that,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}, \ i > 0,$$

  where $\mu_i > 0$. Different rules for the choice of μ_i and $\Delta\boldsymbol{\theta}^{(i)}$ lead to different algorithms; the latter vector is known as the update direction or the search direction. The sequence μ_i is known as the step-size or the step-length, at the ith iteration; note that the values of μ_i may either be constant or change at each iteration.

- The choice of $\Delta\boldsymbol{\theta}^{(i)}$ is done such that to guarantee that

$$J(\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)}),$$

  except at a minimizer, $\boldsymbol{\theta}_*$.

- Assume that at the $i-1$ iteration step the value $\boldsymbol{\theta}^{(i-1)}$ has been obtained. Then, mobilizing a first order Taylor's expansion we can write

$$J\left(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}\right) \approx J(\boldsymbol{\theta}^{(i-1)}) + \mu_i \nabla^T J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}.$$

The Gradient Descent Method

- Our starting point is the method of gradient descent, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \mathbb{R}^l$. As any other iterative technique, the method starts from an initial estimate, $\boldsymbol{\theta}^{(0)}$, and generates a sequence, $\boldsymbol{\theta}^{(i)}$, $i = 1, 2, \ldots$, such that,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}, \ i > 0,$$

  where $\mu_i > 0$. Different rules for the choice of μ_i and $\Delta\boldsymbol{\theta}^{(i)}$ lead to different algorithms; the latter vector is known as the update direction or the search direction. The sequence μ_i is known as the step-size or the step-length, at the ith iteration; note that the values of μ_i may either be constant or change at each iteration.
- The choice of $\Delta\boldsymbol{\theta}^{(i)}$ is done such that to guarantee that

$$J(\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)}),$$

  except at a minimizer, $\boldsymbol{\theta}_*$.
- Assume that at the $i-1$ iteration step the value $\theta^{(i-1)}$ has been obtained. Then, mobilizing a first order Taylor's expansion we can write

$$J\left(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}\right) \approx J(\boldsymbol{\theta}^{(i-1)}) + \mu_i \nabla^T J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}.$$

The Gradient Descent Method

- Our starting point is the method of gradient descent, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \mathbb{R}^l$. As any other iterative technique, the method starts from an initial estimate, $\boldsymbol{\theta}^{(0)}$, and generates a sequence, $\boldsymbol{\theta}^{(i)}$, $i = 1, 2, \ldots$, such that,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}, \ i > 0,$$

  where $\mu_i > 0$. Different rules for the choice of μ_i and $\Delta\boldsymbol{\theta}^{(i)}$ lead to different algorithms; the latter vector is known as the update direction or the search direction. The sequence μ_i is known as the step-size or the step-length, at the ith iteration; note that the values of μ_i may either be constant or change at each iteration.
- The choice of $\Delta\boldsymbol{\theta}^{(i)}$ is done such that to guarantee that

$$J(\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)}),$$

  except at a minimizer, $\boldsymbol{\theta}_*$.
- Assume that at the $i-1$ iteration step the value $\boldsymbol{\theta}^{(i-1)}$ has been obtained. Then, mobilizing a first order Taylor's expansion we can write

$$J\left(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}\right) \approx J(\boldsymbol{\theta}^{(i-1)}) + \mu_i \nabla^T J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}.$$

The Gradient Descent Method

- Selecting the search direction so that

$$\nabla^T J(\boldsymbol{\theta}^{(i-1)})\Delta\boldsymbol{\theta}^{(i)} < 0,$$

  then it guarantees that $J(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)})$. For such a choice, $\Delta\boldsymbol{\theta}^{(i)}$ and $\nabla J(\boldsymbol{\theta}^{(i-1)})$ must form an obtuse angle.

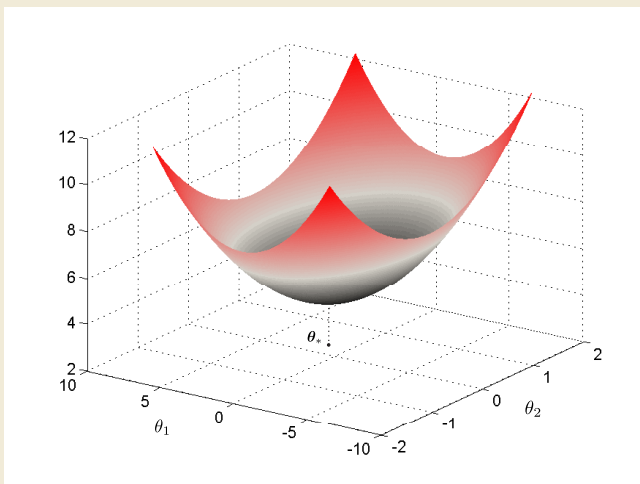
- Figure (a) shows the graph of a cost function in the two-dimensional case, $\boldsymbol{\theta} \in \mathbb{R}^2$ and Figure (b) shows the respective isovalue contours in the two-dimensional plane.

The Gradient Descent Method

- Selecting the search direction so that

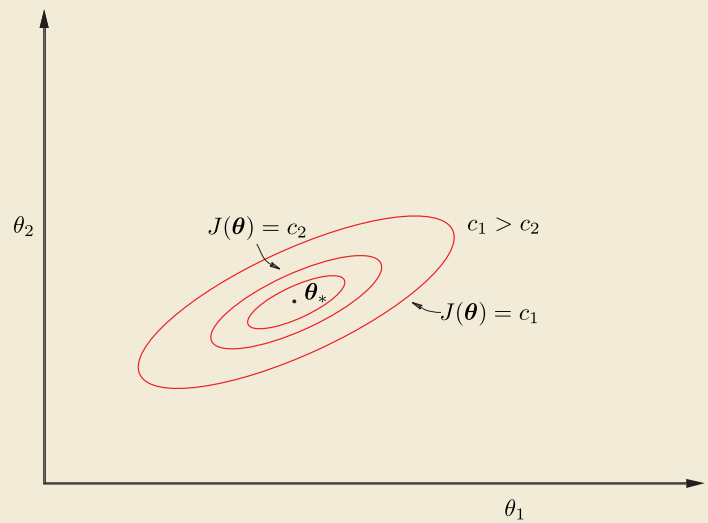
$$\nabla^T J(\boldsymbol{\theta}^{(i-1)})\Delta\boldsymbol{\theta}^{(i)} < 0,$$

  then it guarantees that $J(\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}) < J(\boldsymbol{\theta}^{(i-1)})$. For such a choice, $\Delta\boldsymbol{\theta}^{(i)}$ and $\nabla J(\boldsymbol{\theta}^{(i-1)})$ must form an obtuse angle.
- Figure (a) shows the graph of a cost function in the two-dimensional case, $\boldsymbol{\theta} \in \mathbb{R}^2$ and Figure (b) shows the respective isovalue contours in the two-dimensional plane.



(a)

A cost function in the 2-D parameter space.



(b)

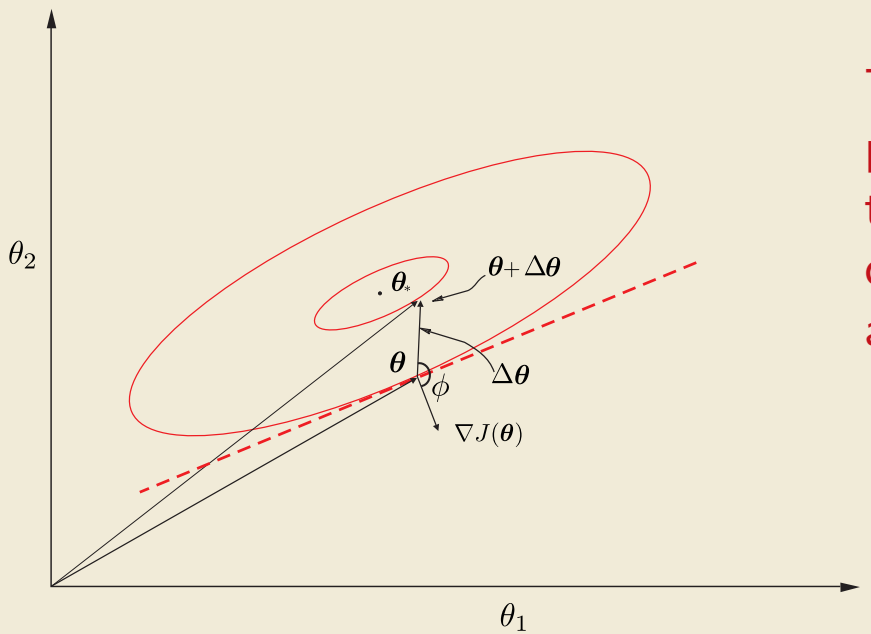
The isovalue curves of the cost function of Figure (a).

The Gradient Descent Method

- For a geometric interpretation of the gradient descent update step, recall that the gradient vector, $\nabla J(\boldsymbol{\theta})$, is perpendicular to the plane (line) tangent to the corresponding isovalue contour, at the point $\boldsymbol{\theta}$. The geometry is illustrated in the figure below; to facilitate the drawing and unclutter notation, we have got rid of the iteration index i. Note that by selecting the search direction to form an obtuse angle with the gradient, it places $\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}$ at a point on a contour which corresponds to a **lower value** of $J(\boldsymbol{\theta})$.

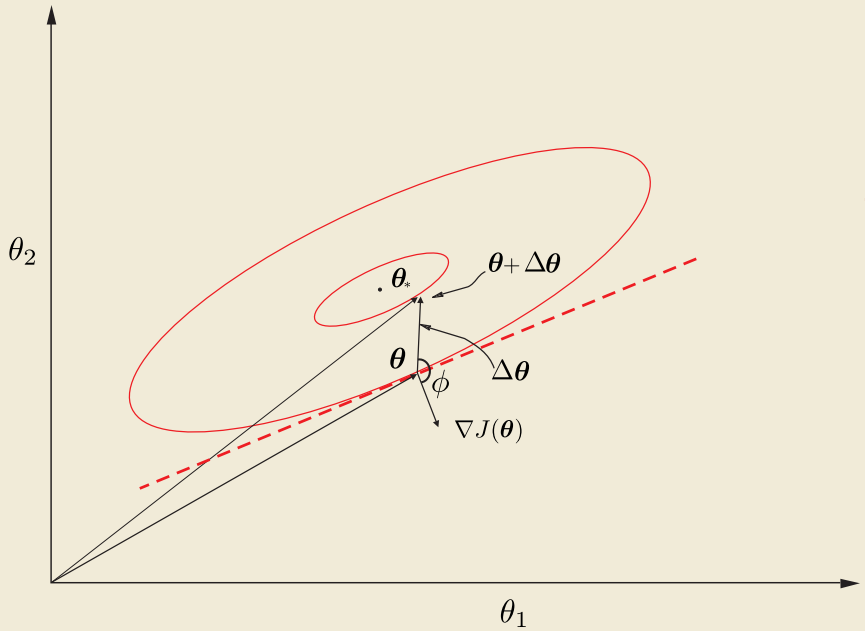
The Gradient Descent Method

- For a geometric interpretation of the gradient descent update step, recall that the gradient vector, $\nabla J(\boldsymbol{\theta})$, is perpendicular to the plane (line) tangent to the corresponding isovalue contour, at the point $\boldsymbol{\theta}$. The geometry is illustrated in the figure below; to facilitate the drawing and unclutter notation, we have got rid of the iteration index i. Note that by selecting the search direction to form an obtuse angle with the gradient, it places $\boldsymbol{\theta}^{(i-1)} + \mu_i \Delta\boldsymbol{\theta}^{(i)}$ at a point on a contour which corresponds to a **lower value** of $J(\boldsymbol{\theta})$.



The gradient vector at a point $\boldsymbol{\theta}$ is perpendicular to the tangent plane at the isovalue curve crossing $\boldsymbol{\theta}$. The descent direction forms an obtuse angle, ϕ, with the gradient vector.

The Gradient Descent Method



The gradient vector at a point $\boldsymbol{\theta}$ is perpendicular to the tangent plane at the isovalue curve crossing $\boldsymbol{\theta}$. The descent direction forms an obtuse angle, ϕ, with the gradient vector.

- There are two issues which are now raised: a) to choose the best search direction along which to move and b) to compute how far along this direction one can go. Even without much mathematics, it is obvious from the figure that if $\mu_i||\Delta\boldsymbol{\theta}^{(i)}||$ is too large, then the new point may be placed on a contour corresponding to a larger value; after all, the first order Taylor's expansion holds approximately true for small deviations from $\boldsymbol{\theta}^{(i)}$.

The Gradient Descent Method

- Search direction: Let us first assume that $\mu_i = 1$ and search for all vectors, $\boldsymbol{z}$, with unit Euclidean norm, $||\boldsymbol{z}|| = 1$. Then, for all possible directions, the one that gives the most negative value of the inner product, $\nabla^T J(\boldsymbol{\theta}^{(i-1)})\boldsymbol{z}$, is that of the negative gradient, i.e.,

$$\boldsymbol{z} = -\frac{\nabla J(\boldsymbol{\theta}^{(i-1)})}{||\nabla J(\boldsymbol{\theta}^{(i-1)})||}.$$

This is known as the gradient or steepest descent direction, and it leads to the following update recursion,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla J(\boldsymbol{\theta}^{(i-1)}). \tag{1}$$

The Gradient Descent Method

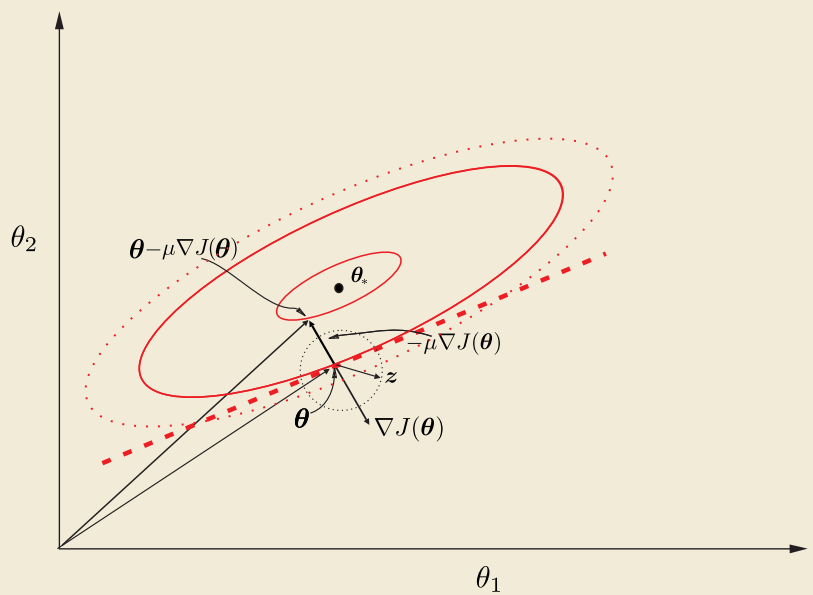
- Search direction: Let us first assume that $\mu_i = 1$ and search for all vectors, $\boldsymbol{z}$, with unit Euclidean norm, $||\boldsymbol{z}|| = 1$. Then, for all possible directions, the one that gives the most negative value of the inner product, $\nabla^T J(\boldsymbol{\theta}^{(i-1)})\boldsymbol{z}$, is that of the negative gradient, i.e.,

$$\boldsymbol{z} = -\frac{\nabla J(\boldsymbol{\theta}^{(i-1)})}{||\nabla J(\boldsymbol{\theta}^{(i-1)})||}.$$

This is known as the gradient or steepest descent direction, and it leads to the following update recursion,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla J(\boldsymbol{\theta}^{(i-1)}). \tag{1}$$

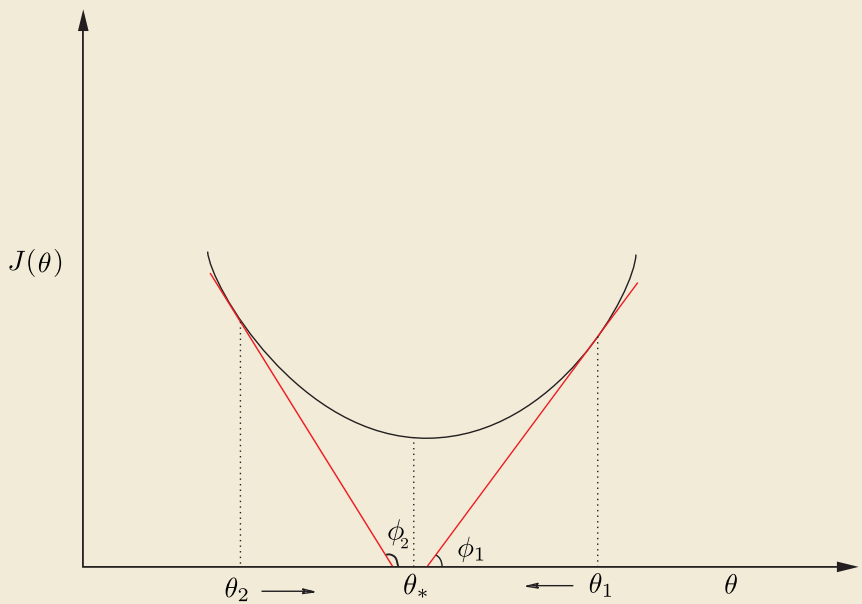
The respective geometry is shown in the figure below.



From all the descent directions of unit Euclidean norm (dotted circle), the negative gradient one leads to the maximum decrease of the cost function.

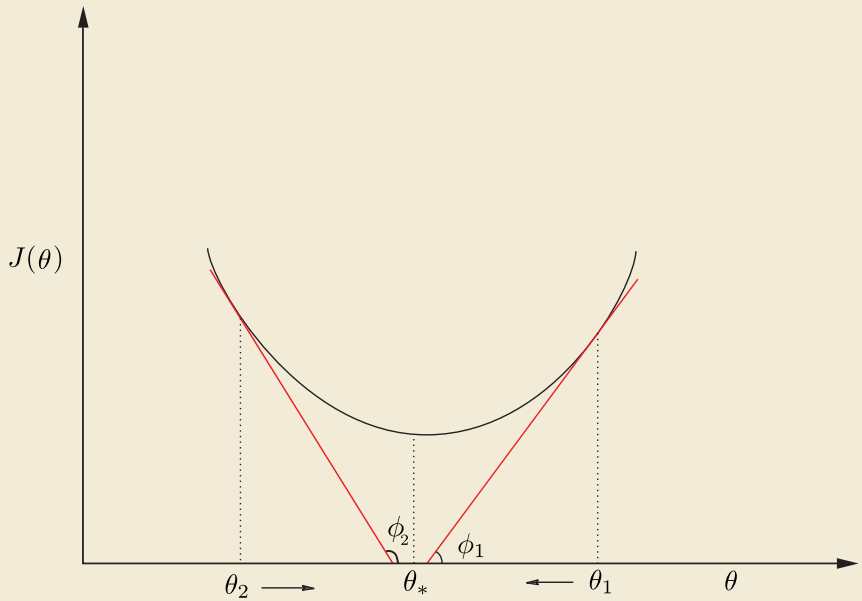
The Gradient Descent Method

- Iteration (1) is illustrated in the figure below, for the one-dimensional case. If at the current iteration the algorithm has "landed" at θ_1, then the derivative of $J(\boldsymbol{\theta})$ at this point is positive (the tangent of an acute angle, ϕ_1) and this will force the update to move to the left towards the minimum. The scenario is different if the current estimate was θ_2. The derivative is negative (the tangent of an obtuse angle, ϕ_2) and this will push the update to the right towards, again, the minimum.



Once the algorithm is at θ_1, the gradient descent will move the point to the left, towards the minimum. The opposite is true for the point θ_2.

The Gradient Descent Method



Once the algorithm is at θ_1, the gradient descent will move the point to the left, towards the minimum. The opposite is true for the point θ_2.

- Note, however, that it is important how far to the left or to the right one has to move. A large move from, say, θ_1, to the left may land the update on the other side of the optimal value. In such a case, the algorithm may oscillate around the minimum and never converge. This brings into the scene the issue on how small or big the step-size μ_i should be in order to guarantee convergence of the algorithm.

The Gradient Descent Method

- It turns out that, the gradient descent method exhibits approximately linear convergence; that is, the error between $\boldsymbol{\theta}^{(i)}$ and the true minimum converges to zero asymptotically in the form of a geometric series. However, the convergence rate depends heavily on the **condition number** of the Hessian matrix of $J(\boldsymbol{\theta})$. For very large values of the condition number, e.g., 1000, the rate of convergence can become extremely slow. The great advantage of the method lies in its low computational requirements.

- Soon, we are going to consider the convergence analysis, and derive bounds for the step-size that guarantee convergence, in the context of the Mean-Square-Error cost function.

The Gradient Descent Method

- It turns out that, the gradient descent method exhibits approximately linear convergence; that is, the error between $\boldsymbol{\theta}^{(i)}$ and the true minimum converges to zero asymptotically in the form of a geometric series. However, the convergence rate depends heavily on the **condition number** of the Hessian matrix of $J(\boldsymbol{\theta})$. For very large values of the condition number, e.g., 1000, the rate of convergence can become extremely slow. The great advantage of the method lies in its low computational requirements.

- Soon, we are going to consider the convergence analysis, and derive bounds for the step-size that guarantee convergence, in the context of the Mean-Square-Error cost function.

Application to the Mean-Square-Error Loss Function

- Let us now apply the gradient descent scheme to derive an iterative algorithm to minimize the MSE cost function,

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\left[(\mathrm{y} - \boldsymbol{\theta}^T \mathbf{x})^2\right] \\ &= \sigma_y^2 - 2\boldsymbol{\theta}^T \boldsymbol{p} + \boldsymbol{\theta}^T \Sigma_x \boldsymbol{\theta}, \end{aligned}$$

where

$$\Sigma := \mathbb{E}[\mathbf{x}\mathbf{x}^T], \quad \boldsymbol{p} := \mathbb{E}[\mathrm{y}\mathbf{x}],$$

are the covariance matrix and the cross-correlation vector and σ_y^2 the variance of y.

- The gradient of the cost function w.r. to θ is readily seen to be

$$\nabla J(\theta) = 2\Sigma_x \theta - 2\boldsymbol{p}.$$

- The fixed step-size case: Employing the above gradient in the update recursion in (1) and absorbing the factor 2 in the step-size, we obtain

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu\left(\Sigma_x \boldsymbol{\theta}^{(i-1)} - \boldsymbol{p}\right), \\ &= \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)}\right). \end{aligned} \tag{2}$$

Application to the Mean-Square-Error Loss Function

- Let us now apply the gradient descent scheme to derive an iterative algorithm to minimize the MSE cost function,

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\left[(\mathrm{y}-\boldsymbol{\theta}^T\mathbf{x})^2\right] \\ &= \sigma_y^2 - 2\boldsymbol{\theta}^T\boldsymbol{p} + \boldsymbol{\theta}^T\Sigma_x\boldsymbol{\theta}, \end{aligned}$$

  where

$$\Sigma := \mathbb{E}[\mathbf{x}\mathbf{x}^T], \quad \boldsymbol{p} := \mathbb{E}[\mathrm{y}\mathbf{x}],$$

  are the covariance matrix and the cross-correlation vector and σ_y^2 the variance of y.

- The gradient of the cost function w.r. to $\boldsymbol{\theta}$ is readily seen to be

$$\nabla J(\boldsymbol{\theta}) = 2\Sigma_x\boldsymbol{\theta} - 2\boldsymbol{p}.$$

- The fixed step-size case: Employing the above gradient in the update recursion in (1) and absorbing the factor 2 in the step-size, we obtain

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu\left(\Sigma_x\boldsymbol{\theta}^{(i-1)} - \boldsymbol{p}\right), \\ &= \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x\boldsymbol{\theta}^{(i-1)}\right). \end{aligned} \tag{2}$$

Application to the Mean-Square-Error Loss Function

- Let us now apply the gradient descent scheme to derive an iterative algorithm to minimize the MSE cost function,

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}\left[(\mathrm{y}-\boldsymbol{\theta}^T\mathbf{x})^2\right] \\ &= \sigma_y^2 - 2\boldsymbol{\theta}^T\boldsymbol{p} + \boldsymbol{\theta}^T\Sigma_x\boldsymbol{\theta}, \end{aligned}$$

  where

$$\Sigma := \mathbb{E}[\mathbf{x}\mathbf{x}^T], \quad \boldsymbol{p} := \mathbb{E}[\mathrm{y}\mathbf{x}],$$

  are the covariance matrix and the cross-correlation vector and σ_y^2 the variance of y.

- The gradient of the cost function w.r. to $\boldsymbol{\theta}$ is readily seen to be

$$\nabla J(\boldsymbol{\theta}) = 2\Sigma_x\boldsymbol{\theta} - 2\boldsymbol{p}.$$

- The fixed step-size case: Employing the above gradient in the update recursion in (1) and absorbing the factor 2 in the step-size, we obtain

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu\left(\Sigma_x\boldsymbol{\theta}^{(i-1)} - \boldsymbol{p}\right), \\ &= \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x\boldsymbol{\theta}^{(i-1)}\right). \end{aligned} \tag{2}$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- It turns out that the values of the step-size that guarantee convergence lie in the interval

$$0 < \mu < 2/\lambda_{\max}$$

where $\lambda_{\max}$ denotes the maximum eigenvalue of Σ_x.

- **Proof**: Define

$$\boldsymbol{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \tag{3}$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations, $\Sigma_x \boldsymbol{\theta}_* = \boldsymbol{p}$.

- Subtracting $\boldsymbol{\theta}_*$ from both sides of (2) and plugging in (3), we obtain

$$\begin{aligned} \boldsymbol{c}^{(i)} &= \boldsymbol{c}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{c}^{(i-1)} - \Sigma_x \boldsymbol{\theta}_*\right) \\ &= \boldsymbol{c}^{(i-1)} - \mu\Sigma_x \boldsymbol{c}^{(i-1)} = \left(I - \mu\Sigma_x\right)\boldsymbol{c}^{(i-1)}. \end{aligned} \tag{4}$$

- Recall that Σ_x is a symmetric positive definite matrix, hence it can be written as

$$\Sigma_x = Q\Lambda Q^T,$$

where

$$\Lambda := \text{diag}\{\lambda_1, \ldots, \lambda_l\} \quad \text{and} \quad Q := [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_l],$$

with λ_j, $\boldsymbol{q}_j$, $j = 1, 2, \ldots, l$, being the (positive) eigenvalues and the respective normalized (orthogonal) eigenvectors of the covariance matrix, i.e.,

$$\boldsymbol{q}_k^T \boldsymbol{q}_j = \delta_{kj}, \; k, j = 1, 2, \ldots, l \Longrightarrow Q^T = Q^{-1}.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- It turns out that the values of the step-size that guarantee convergence lie in the interval

$$0 < \mu < 2/\lambda_{\max}$$

where $\lambda_{\max}$ denotes the maximum eigenvalue of Σ_x.

- **Proof**: Define

$$\boldsymbol{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \tag{3}$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations, $\Sigma_x \boldsymbol{\theta}_* = \boldsymbol{p}$.

- Subtracting θ_* from both sides of (2) and plugging in (3), we obtain

$$\begin{aligned} \boldsymbol{c}^{(i)} &= \boldsymbol{c}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{c}^{(i-1)} - \Sigma_x \boldsymbol{\theta}_*\right) \\ &= \boldsymbol{c}^{(i-1)} - \mu\Sigma_x \boldsymbol{c}^{(i-1)} = \left(I - \mu\Sigma_x\right)\boldsymbol{c}^{(i-1)}. \end{aligned} \tag{4}$$

- Recall that Σ_x is a symmetric positive definite matrix, hence it can be written as

$$\Sigma_x = Q\Lambda Q^T,$$

where

$$\Lambda := \text{diag}\{\lambda_1, \ldots, \lambda_l\} \quad \text{and} \quad Q := [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_l],$$

with λ_j, $\boldsymbol{q}_j$, $j = 1, 2, \ldots, l$, being the (positive) eigenvalues and the respective normalized (orthogonal) eigenvectors of the covariance matrix, i.e.,

$$\boldsymbol{q}_k^T \boldsymbol{q}_j = \delta_{kj}, \; k, j = 1, 2, \ldots, l \Longrightarrow Q^T = Q^{-1}.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- It turns out that the values of the step-size that guarantee convergence lie in the interval

$$0 < \mu < 2/\lambda_{\max}$$

where $\lambda_{\max}$ denotes the maximum eigenvalue of Σ_x.

- **Proof**: Define

$$\boldsymbol{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \tag{3}$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations, $\Sigma_x \boldsymbol{\theta}_* = \boldsymbol{p}$.

- Subtracting $\boldsymbol{\theta}_*$ from both sides of (2) and plugging in (3), we obtain

$$\begin{aligned} \boldsymbol{c}^{(i)} &= \boldsymbol{c}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{c}^{(i-1)} - \Sigma_x \boldsymbol{\theta}_*\right) \\ &= \boldsymbol{c}^{(i-1)} - \mu\Sigma_x \boldsymbol{c}^{(i-1)} = \left(I - \mu\Sigma_x\right)\boldsymbol{c}^{(i-1)}. \end{aligned} \tag{4}$$

- Recall that Σ_x is a symmetric positive definite matrix, hence it can be written as

$$\Sigma_x = Q\Lambda Q^T,$$

where

$$\Lambda := \text{diag}\{\lambda_1, \ldots, \lambda_l\} \quad \text{and} \quad Q := [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_l],$$

with λ_j, $\boldsymbol{q}_j$, $j = 1, 2, \ldots, l$, being the (positive) eigenvalues and the respective normalized (orthogonal) eigenvectors of the covariance matrix, i.e.,

$$\boldsymbol{q}_k^T \boldsymbol{q}_j = \delta_{kj}, \; k, j = 1, 2, \ldots, l \Longrightarrow Q^T = Q^{-1}.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- It turns out that the values of the step-size that guarantee convergence lie in the interval

$$0 < \mu < 2/\lambda_{\max}$$

where $\lambda_{\max}$ denotes the maximum eigenvalue of Σ_x.

- **Proof**: Define

$$\boldsymbol{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \tag{3}$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations, $\Sigma_x \boldsymbol{\theta}_* = \boldsymbol{p}$.

- Subtracting $\boldsymbol{\theta}_*$ from both sides of (2) and plugging in (3), we obtain

$$\begin{aligned} \boldsymbol{c}^{(i)} &= \boldsymbol{c}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{c}^{(i-1)} - \Sigma_x \boldsymbol{\theta}_*\right) \\ &= \boldsymbol{c}^{(i-1)} - \mu\Sigma_x \boldsymbol{c}^{(i-1)} = \left(I - \mu\Sigma_x\right)\boldsymbol{c}^{(i-1)}. \end{aligned} \tag{4}$$

- Recall that Σ_x is a symmetric positive definite matrix, hence it can be written as

$$\Sigma_x = Q\Lambda Q^T,$$

where

$$\Lambda := \text{diag}\{\lambda_1, \ldots, \lambda_l\} \quad \text{and} \quad Q := [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_l],$$

with λ_j, $\boldsymbol{q}_j$, $j = 1, 2, \ldots, l$, being the (positive) eigenvalues and the respective normalized (orthogonal) eigenvectors of the covariance matrix, i.e.,

$$\boldsymbol{q}_k^T \boldsymbol{q}_j = \delta_{kj}, \; k, j = 1, 2, \ldots, l \Longrightarrow Q^T = Q^{-1}.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- Plugging the factorization of Σ_x into (4), we obtain

$$\boldsymbol{c}^{(i)} = Q\left(I - \mu\Lambda\right)Q^T\boldsymbol{c}^{(i-1)},$$

or

$$\boldsymbol{v}^{(i)} = \left(I - \mu\Lambda\right)\boldsymbol{v}^{(i-1)},$$

where

$$\boldsymbol{v}^{(i)} := Q^T\boldsymbol{c}^{(i)},\ i = 1, 2, \ldots. \tag{5}$$

- The previously used "trick" is a standard one and its aim is to "decouple" the various components of $\boldsymbol{\theta}^{(i)}$ in (2). Indeed, each one of the components, $v^{(i)}(j),\ j = 1, 2, \ldots, l$, of $\boldsymbol{v}^{(i)}$ follows an iteration path, which is independent on the rest of the components. If we denote by $v^{(0)}(j)$ is the jth component of $\boldsymbol{v}^{(0)}$, we get,

$$\begin{aligned} v^{(i)}(j) &= (1 - \mu\lambda_j)v^{(i-1)}(j) = (1 - \mu\lambda_j)^2 v^{(i-2)}(j) = \ldots \\ &= (1 - \mu\lambda_j)^i v^{(0)}(j), \end{aligned} \tag{6}$$

- It is now readily seen that if

$$|1 - \mu\lambda_j| < 1 \Longleftrightarrow -1 < 1 - \mu\lambda_j < 1,\ j = 1, 2, \ldots, l, \tag{7}$$

the geometric series converges to zero or

$$\boldsymbol{v}^{(i)} \longrightarrow \mathbf{0} \Longrightarrow Q^T(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*) \longrightarrow \mathbf{0} \Longrightarrow \boldsymbol{\theta}^{(i)} \longrightarrow \boldsymbol{\theta}_*.$$

- Note that (7) is equivalent to $0 < \mu < 2/\lambda_{\max}$ and the claim has been proved.

Convergence of the Gradient Descent Algorithm For The MSE Case

- Plugging the factorization of Σ_x into (4), we obtain

$$\boldsymbol{c}^{(i)} = Q\left(I - \mu\Lambda\right)Q^T\boldsymbol{c}^{(i-1)},$$

or

$$\boldsymbol{v}^{(i)} = \left(I - \mu\Lambda\right)\boldsymbol{v}^{(i-1)},$$

where

$$\boldsymbol{v}^{(i)} := Q^T\boldsymbol{c}^{(i)},\ i = 1, 2, \ldots. \tag{5}$$

- The previously used "trick" is a standard one and its aim is to "decouple" the various components of $\boldsymbol{\theta}^{(i)}$ in (2). Indeed, each one of the components, $v^{(i)}(j),\ j = 1, 2, \ldots, l$, of $\boldsymbol{v}^{(i)}$ follows an iteration path, which is independent on the rest of the components. If we denote by $v^{(0)}(j)$ is the jth component of $\boldsymbol{v}^{(0)}$, we get,

$$\begin{aligned} v^{(i)}(j) &= (1 - \mu\lambda_j)v^{(i-1)}(j) = (1 - \mu\lambda_j)^2 v^{(i-2)}(j) = \ldots \\ &= (1 - \mu\lambda_j)^i v^{(0)}(j), \end{aligned} \tag{6}$$

- It is now readily seen that if

$$|1 - \mu\lambda_j| < 1 \Longleftrightarrow -1 < 1 - \mu\lambda_j < 1,\ j = 1, 2, \ldots, l, \tag{7}$$

the geometric series converges to zero or

$$\boldsymbol{v}^{(i)} \longrightarrow \mathbf{0} \Longrightarrow Q^T(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*) \longrightarrow \mathbf{0} \Longrightarrow \boldsymbol{\theta}^{(i)} \longrightarrow \boldsymbol{\theta}_*.$$

- Note that (7) is equivalent to $0 < \mu < 2/\lambda_{\max}$ and the claim has been proved.

Convergence of the Gradient Descent Algorithm For The MSE Case

- Plugging the factorization of Σ_x into (4), we obtain

$$\boldsymbol{c}^{(i)} = Q\left(I - \mu\Lambda\right)Q^T\boldsymbol{c}^{(i-1)},$$

or

$$\boldsymbol{v}^{(i)} = \left(I - \mu\Lambda\right)\boldsymbol{v}^{(i-1)},$$

where

$$\boldsymbol{v}^{(i)} := Q^T\boldsymbol{c}^{(i)},\ i = 1, 2, \ldots. \tag{5}$$

- The previously used "trick" is a standard one and its aim is to "decouple" the various components of $\boldsymbol{\theta}^{(i)}$ in (2). Indeed, each one of the components, $v^{(i)}(j),\ j = 1, 2, \ldots, l$, of $\boldsymbol{v}^{(i)}$ follows an iteration path, which is independent on the rest of the components. If we denote by $v^{(0)}(j)$ is the jth component of $\boldsymbol{v}^{(0)}$, we get,

$$\begin{aligned} v^{(i)}(j) &= (1 - \mu\lambda_j)v^{(i-1)}(j) = (1 - \mu\lambda_j)^2 v^{(i-2)}(j) = \ldots \\ &= (1 - \mu\lambda_j)^i v^{(0)}(j), \end{aligned} \tag{6}$$

- It is now readily seen that if

$$|1 - \mu\lambda_j| < 1 \Longleftrightarrow -1 < 1 - \mu\lambda_j < 1,\ j = 1, 2, \ldots, l, \tag{7}$$

the geometric series converges to zero or

$$\boldsymbol{v}^{(i)} \longrightarrow \mathbf{0} \Longrightarrow Q^T(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*) \longrightarrow \mathbf{0} \Longrightarrow \boldsymbol{\theta}^{(i)} \longrightarrow \boldsymbol{\theta}_*.$$

- Note that (7) is equivalent to $0 < \mu < 2/\lambda_{\max}$ and the claim has been proved.

Convergence of the Gradient Descent Algorithm For The MSE Case

- Plugging the factorization of Σ_x into (4), we obtain

$$\boldsymbol{c}^{(i)} = Q\left(I - \mu\Lambda\right)Q^T\boldsymbol{c}^{(i-1)},$$

or

$$\boldsymbol{v}^{(i)} = \left(I - \mu\Lambda\right)\boldsymbol{v}^{(i-1)},$$

where

$$\boldsymbol{v}^{(i)} := Q^T\boldsymbol{c}^{(i)},\ i = 1, 2, \ldots. \tag{5}$$

- The previously used "trick" is a standard one and its aim is to "decouple" the various components of $\boldsymbol{\theta}^{(i)}$ in (2). Indeed, each one of the components, $v^{(i)}(j),\ j = 1, 2, \ldots, l$, of $\boldsymbol{v}^{(i)}$ follows an iteration path, which is independent on the rest of the components. If we denote by $v^{(0)}(j)$ is the jth component of $\boldsymbol{v}^{(0)}$, we get,

$$\begin{aligned} v^{(i)}(j) &= (1 - \mu\lambda_j)v^{(i-1)}(j) = (1 - \mu\lambda_j)^2 v^{(i-2)}(j) = \ldots \\ &= (1 - \mu\lambda_j)^i v^{(0)}(j), \end{aligned} \tag{6}$$

- It is now readily seen that if

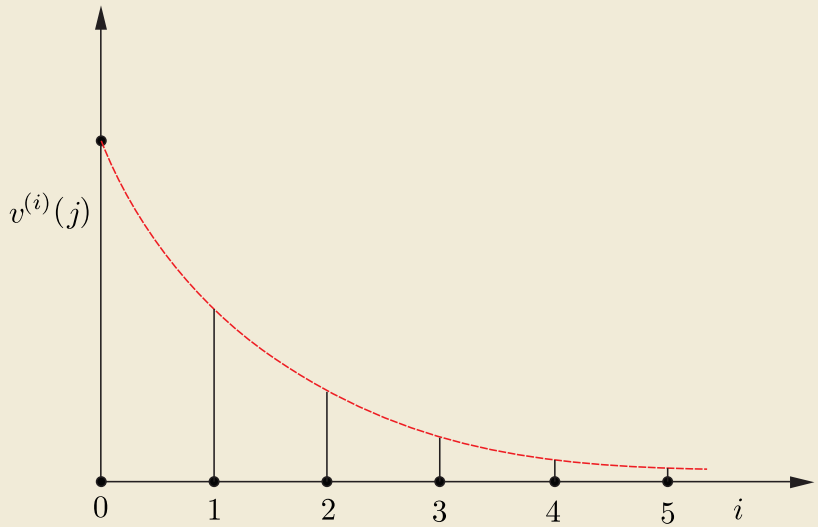
$$|1 - \mu\lambda_j| < 1 \Longleftrightarrow -1 < 1 - \mu\lambda_j < 1,\ j = 1, 2, \ldots, l, \tag{7}$$

the geometric series converges to zero or

$$\boldsymbol{v}^{(i)} \longrightarrow \mathbf{0} \Longrightarrow Q^T(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*) \longrightarrow \mathbf{0} \Longrightarrow \boldsymbol{\theta}^{(i)} \longrightarrow \boldsymbol{\theta}_*.$$

- Note that (7) is equivalent to $0 < \mu < 2/\lambda_{\text{max}}$ and the claim has been proved.

Convergence of the Gradient Descent Algorithm For The MSE Case



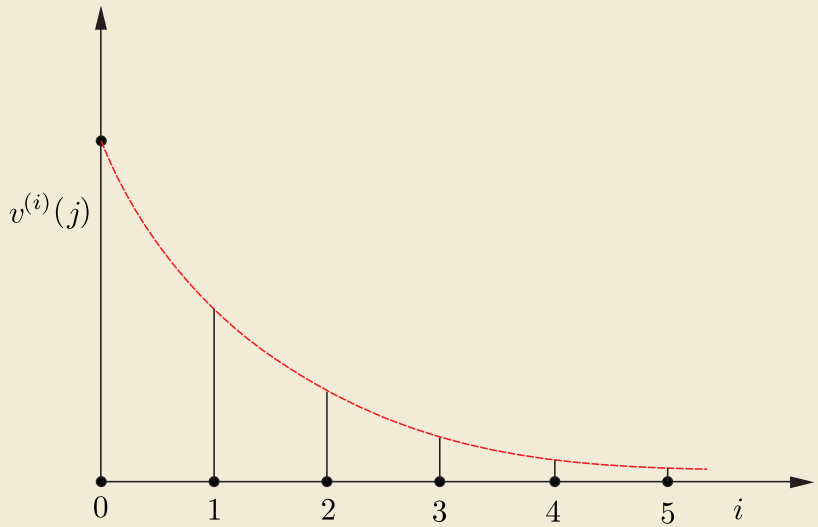
The figure shows a typical sketch of the evolution of a transformed vector component, $v^{(i)}(j)$, as a function of the iteration steps for the case $0 < 1 - \mu\lambda_j < 1$. Observe that the curve is of an approximate exponentially decreasing type.

- Time Constant: Assume that the envelope, denoted by the red line in the figure above, is (approximately) of an exponential form, $f(t) = \exp(-t/\tau_j)$. Plug into $f(t)$, as the values corresponding at the time instants, $t = iT$ and $t = (i-1)T$, the values of $v^{(i)}(j)$, $v^{(i-1)}(j)$ from (6); then, the time constant results as

$$\tau_j = \frac{-1}{\ln(1 - \mu\lambda_j)},$$

assuming that the sampling time between two successive iterations is $T = 1$. For small values of μ, we can write

$$\tau_j \approx \frac{1}{\mu\lambda_j}, \text{ for } \mu \ll 1.$$



The figure shows a typical sketch of the evolution of a transformed vector component, $v^{(i)}(j)$, as a function of the iteration steps for the case $0 < 1 - \mu\lambda_j < 1$. Observe that the curve is of an approximate exponentially decreasing type.

- Time Constant: Assume that the envelope, denoted by the red line in the figure above, is (approximately) of an exponential form, $f(t) = \exp(-t/\tau_j)$. Plug into $f(t)$, as the values corresponding at the time instants, $t = iT$ and $t = (i-1)T$, the values of $v^{(i)}(j),\ v^{(i-1)}(j)$ from (6); then, the time constant results as

$$\tau_j = \frac{-1}{\ln(1-\mu\lambda_j)},$$

assuming that the sampling time between two successive iterations is $T = 1$. For small values of μ, we can write

$$\tau_j \approx \frac{1}{\mu\lambda_j},\ \text{for } \mu \ll 1.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- That is, the slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue. However, this is only true for small enough values of μ. For the more general case, this may not be true. Recall that the rate of convergence depends on the value of the term $1-\mu\lambda_j$. This is also known as the jth mode. Its value depends not only on λ_j but also on μ.
- Let us take, as an example, the case of μ taking a value very close to the maximum allowable one, $\mu \simeq 2/\lambda_{\max}$. Then, the mode corresponding to the maximum eigenvalue will have an absolute value very close to one. On the other hand, the time constant of the mode corresponding to the minimum eigenvalue will be controlled by the value of $|1-2\lambda_{\min}/\lambda_{\max}|$, which can be much smaller than one. In such a case, the mode corresponding to the maximum eigenvalue exhibits slower convergence.
- In order to obtain the optimum value for the step-size, one has to select its value in a way so that the resulting maximum absolute mode value to be minimum. This is a min/max task, i.e.,

$$\begin{aligned} \mu_o &= \arg\min_{\mu}\max_{j}|1-\mu\lambda_j|, \\ \text{s.t.} &\quad |1-\mu\lambda_i|<1,\ j=1,2,\ldots,l. \end{aligned}$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- That is, the slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue. However, this is only true for small enough values of μ. For the more general case, this may not be true. Recall that the rate of convergence depends on the value of the term $1-\mu\lambda_j$. This is also known as the jth mode. Its value depends not only on λ_j but also on μ.
- Let us take, as an example, the case of μ taking a value very close to the maximum allowable one, $\mu \simeq 2/\lambda_{\text{max}}$. Then, the mode corresponding to the maximum eigenvalue will have an absolute value very close to one. On the other hand, the time constant of the mode corresponding to the minimum eigenvalue will be controlled by the value of $|1-2\lambda_{\text{min}}/\lambda_{\text{max}}|$, which can be much smaller than one. In such a case, the mode corresponding to the maximum eigenvalue exhibits slower convergence.
- In order to obtain the optimum value for the step-size, one has to select its value in a way so that the resulting maximum absolute mode value to be minimum. This is a min/max task, i.e.,

$$\mu_o = \arg\min_{\mu}\max_{j}|1-\mu\lambda_j|,$$

$$\text{s.t.} \quad |1-\mu\lambda_i|<1,\ j=1,2,\ldots,l.$$

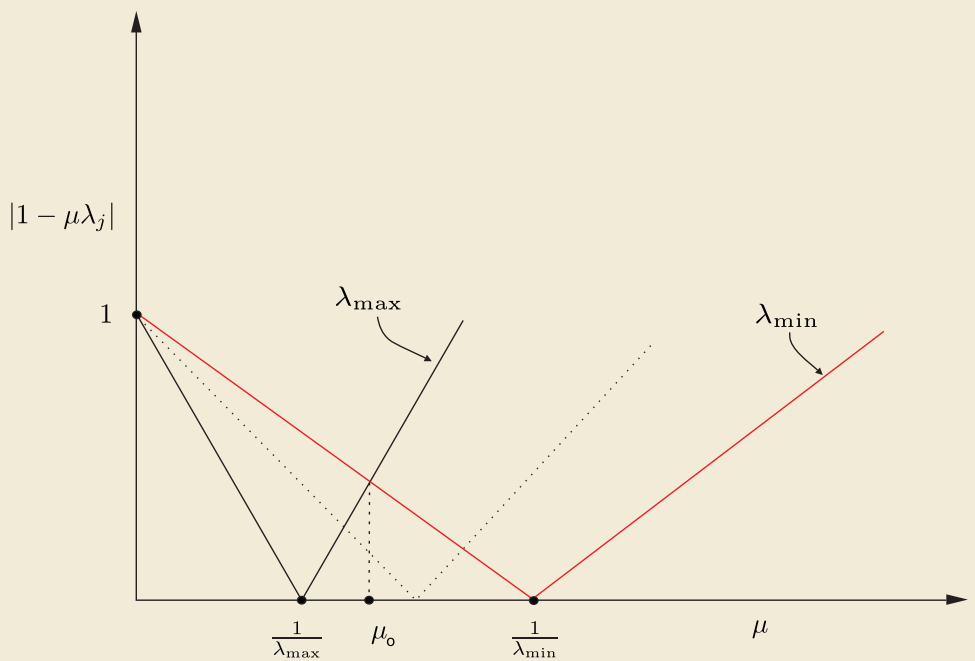
Convergence of the Gradient Descent Algorithm For The MSE Case

- That is, the slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue. However, this is only true for small enough values of μ. For the more general case, this may not be true. Recall that the rate of convergence depends on the value of the term $1-\mu\lambda_j$. This is also known as the jth mode. Its value depends not only on λ_j but also on μ.
- Let us take, as an example, the case of μ taking a value very close to the maximum allowable one, $\mu \simeq 2/\lambda_{\text{max}}$. Then, the mode corresponding to the maximum eigenvalue will have an absolute value very close to one. On the other hand, the time constant of the mode corresponding to the minimum eigenvalue will be controlled by the value of $|1-2\lambda_{\text{min}}/\lambda_{\text{max}}|$, which can be much smaller than one. In such a case, the mode corresponding to the maximum eigenvalue exhibits slower convergence.
- In order to obtain the optimum value for the step-size, one has to select its value in a way so that the resulting maximum absolute mode value to be minimum. This is a min/max task, i.e.,

$$\begin{aligned} \mu_o &= \arg \min_{\mu} \max_{j} |1-\mu\lambda_j|, \\ \text{s.t.} & \quad |1-\mu\lambda_i| < 1, \ j=1,2,\ldots,l. \end{aligned}$$

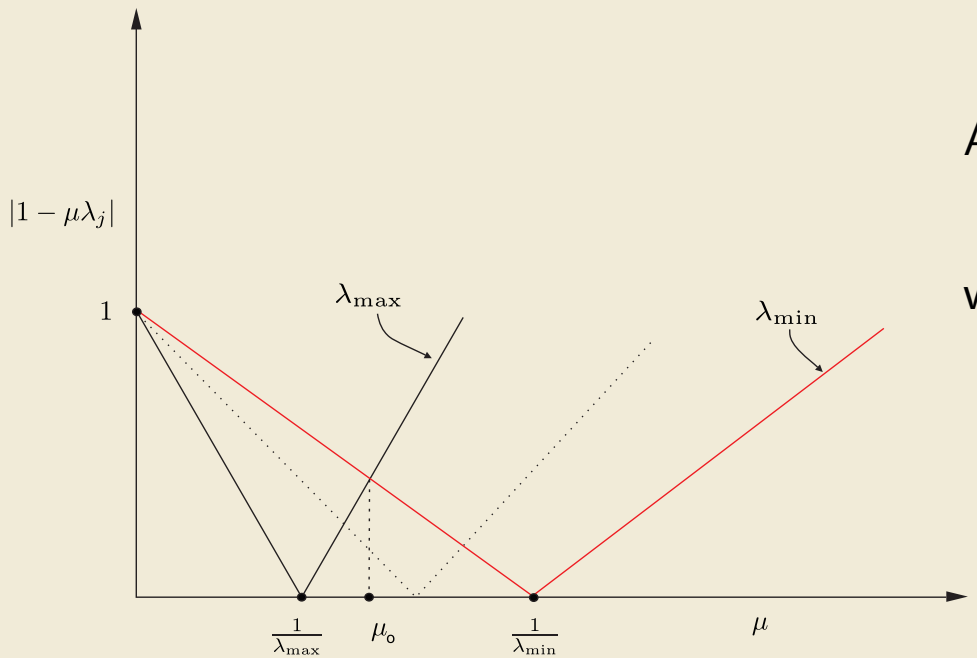
Convergence of the Gradient Descent Algorithm For The MSE Case

- The task can be solved easily graphically. The figure below shows the absolute values of the modes (corresponding to the maximum, minimum and an intermediate one eigenvalues). The (absolute) values of the modes initially decrease, as μ increases and then they start increasing. Observe that the optimal value results when the curves for the maximum and minimum eigenvalues intersect. Indeed, this corresponds to the minimum maximum value.



Convergence of the Gradient Descent Algorithm For The MSE Case

- The task can be solved easily graphically. The figure below shows the absolute values of the modes (corresponding to the maximum, minimum and an intermediate one eigenvalues). The (absolute) values of the modes initially decrease, as μ increases and then they start increasing. Observe that the optimal value results when the curves for the maximum and minimum eigenvalues intersect. Indeed, this corresponds to the minimum maximum value.



At the intersection, we have

$$1-\mu_o\lambda_{\min} = -(1-\mu_o\lambda_{\max}),$$

which results in

$$\mu_o = \frac{2}{\lambda_{\max}+\lambda_{\min}}.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- At the optimal value, μ_o, there are two slowest modes; one corresponding to $\lambda_{\min}$ (i.e., $1-\mu_o\lambda_{\min}$) and another one corresponding to $\lambda_{\max}$ (i.e., $1-\mu_o\lambda_{\max}$). They have equal magnitudes but opposite signs, and they are given by,

$$\pm\frac{\rho-1}{\rho+1}, \text{ where } \rho := \frac{\lambda_{\max}}{\lambda_{\min}}.$$

  In other words, the convergence rate depends on the **eigenvalues spread** of the covariance matrix.

- Parameter Error Vector Convergence: From the definitions in (3) and (5), we get

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}_* + Q\boldsymbol{v}^{(i)} \\ &= \boldsymbol{\theta}_* + [\boldsymbol{q}_1,\ldots,\boldsymbol{q}_l][v^{(i)}(1),\ldots,v^{(i)}(l)]^T \\ &= \boldsymbol{\theta}_* + \sum_{k=1}^{l}\boldsymbol{q}_k v^{(i)}(k). \qquad (8)\end{aligned}$$

- The above is written component-wise as

$$\theta^{(i)}(j) = \theta_*^{(i)}(j) + \sum_{k=1}^{l} q_k(j)v^{(0)}(k)(1-\mu\lambda_k)^i,\ j=1,2,\ldots l.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- At the optimal value, μ_o, there are two slowest modes; one corresponding to $\lambda_{\min}$ (i.e., $1-\mu_o\lambda_{\min}$) and another one corresponding to $\lambda_{\max}$ (i.e., $1-\mu_o\lambda_{\max}$). They have equal magnitudes but opposite signs, and they are given by,

$$\pm\frac{\rho-1}{\rho+1}, \text{ where } \rho := \frac{\lambda_{\max}}{\lambda_{\min}}.$$

  In other words, the convergence rate depends on the **eigenvalues spread** of the covariance matrix.

- Parameter Error Vector Convergence: From the definitions in (3) and (5), we get

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}_* + Q\boldsymbol{v}^{(i)} \\ &= \boldsymbol{\theta}_* + [\boldsymbol{q}_1,\ldots,\boldsymbol{q}_l][v^{(i)}(1),\ldots,v^{(i)}(l)]^T \\ &= \boldsymbol{\theta}_* + \sum_{k=1}^{l}\boldsymbol{q}_k v^{(i)}(k). \end{aligned} \tag{8}$$

- The above is written component-wise as

$$\theta^{(i)}(j) = \theta_*^{(i)}(j) + \sum_{k=1}^{l} q_k(j)v^{(0)}(k)(1-\mu\lambda_k)^i,\ j=1,2,\ldots l.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- At the optimal value, μ_o, there are two slowest modes; one corresponding to $\lambda_{\min}$ (i.e., $1-\mu_o\lambda_{\min}$) and another one corresponding to $\lambda_{\max}$ (i.e., $1-\mu_o\lambda_{\max}$). They have equal magnitudes but opposite signs, and they are given by,

$$\pm\frac{\rho-1}{\rho+1}, \text{ where } \rho := \frac{\lambda_{\max}}{\lambda_{\min}}.$$

  In other words, the convergence rate depends on the **eigenvalues spread** of the covariance matrix.

- Parameter Error Vector Convergence: From the definitions in (3) and (5), we get

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}_* + Q\boldsymbol{v}^{(i)} \\ &= \boldsymbol{\theta}_* + [\boldsymbol{q}_1,\ldots,\boldsymbol{q}_l][v^{(i)}(1),\ldots,v^{(i)}(l)]^T \\ &= \boldsymbol{\theta}_* + \sum_{k=1}^{l}\boldsymbol{q}_k v^{(i)}(k). \end{aligned} \tag{8}$$

- The above is written component-wise as

$$\theta^{(i)}(j) = \theta_*^{(i)}(j) + \sum_{k=1}^{l} q_k(j)v^{(0)}(k)(1-\mu\lambda_k)^i,\ j=1,2,\ldots l.$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- In other words, the components of $\boldsymbol{\theta}^{(i)}$ converge into the respective components of the optimum vector $\boldsymbol{\theta}_*$ as a weighted average of exponentials, $(1-\mu\lambda_k)^i$. Computing the respective time constant in close form is not possible; however, we can state lower and upper bounds. The lower bound corresponds to the time constant of the fastest converging mode and the upper bound to the slowest of the modes. For small values of $\mu << 1$, we can write

$$\frac{1}{\mu\lambda_{\max}} \leq \tau \leq \frac{1}{\mu\lambda_{\min}}.$$

- The learning curve: We now turn our focus on the mean-square error. In chapter 4, it has been shown that

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*)^T \Sigma_x (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*),$$

or, mobilizing (8), the factorization $\Sigma_x = Q\Lambda Q^T$ and the orthonormality of the eigenvectors, we obtain

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + \sum_{j=1}^{l} \lambda_j |v^{(i)}(j)|^2$$

or

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + \sum_{j=1}^{l} \lambda_j (1-\mu\lambda_j)^{2i} |v^{(0)}(j)|^2. \quad (9)$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- In other words, the components of $\boldsymbol{\theta}^{(i)}$ converge into the respective components of the optimum vector $\boldsymbol{\theta}_*$ as a weighted average of exponentials, $(1-\mu\lambda_k)^i$. Computing the respective time constant in close form is not possible; however, we can state lower and upper bounds. The lower bound corresponds to the time constant of the fastest converging mode and the upper bound to the slowest of the modes. For small values of $\mu << 1$, we can write

$$\frac{1}{\mu\lambda_{\max}} \leq \tau \leq \frac{1}{\mu\lambda_{\min}}.$$

- The learning curve: We now turn our focus on the mean-square error. In chapter 4, it has been shown that

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*)^T \Sigma_x (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*),$$

or, mobilizing (8), the factorization $\Sigma_x = Q\Lambda Q^T$ and the orthonormality of the eigenvectors, we obtain

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + \sum_{j=1}^{l} \lambda_j |v^{(i)}(j)|^2$$

or

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + \sum_{j=1}^{l} \lambda_j (1-\mu\lambda_j)^{2i} |v^{(0)}(j)|^2. \tag{9}$$

Convergence of the Gradient Descent Algorithm For The MSE Case

- For $i \longrightarrow \infty$, recursion (9) converges to the minimum value $J(\boldsymbol{\theta}_*)$ asymptotically. Moreover, observe that this convergence is monotonic, since $\lambda_j(1-\mu\lambda_j)^2$ is positive. Following similar arguments as before, the respective time constants for each one of the modes are now,

$$\tau_j^{\text{mse}} = \frac{-1}{2\ln(1-\mu\lambda_j)} \approx \frac{1}{2\mu\lambda_j}.$$

- The previous approximation for the time constant makes crystal clear the role that the step-size, μ, plays in the gradient descent iterative scheme. Its choice is crucial not only to guarantee convergence, but also it determines the speed with which the algorithm converges to the solution. The smaller its value is the slower the convergence rate becomes.
- The other factor which the convergence speed depends on is the eigenvalue structure of the input data covariance matrix.

Convergence of the Gradient Descent Algorithm For The MSE Case

- For $i \longrightarrow \infty$, recursion (9) converges to the minimum value $J(\boldsymbol{\theta}_*)$ asymptotically. Moreover, observe that this convergence is monotonic, since $\lambda_j(1-\mu\lambda_j)^2$ is positive. Following similar arguments as before, the respective time constants for each one of the modes are now,

$$\tau_j^{\text{mse}} = \frac{-1}{2\ln(1-\mu\lambda_j)} \approx \frac{1}{2\mu\lambda_j}.$$

- The previous approximation for the time constant makes crystal clear the role that the step-size, μ, plays in the gradient descent iterative scheme. Its choice is crucial not only to guarantee convergence, but also it determines the speed with which the algorithm converges to the solution. The smaller its value is the slower the convergence rate becomes.
- The other factor which the convergence speed depends on is the eigenvalue structure of the input data covariance matrix.

Convergence of the Gradient Descent Algorithm For The MSE Case

- For $i \longrightarrow \infty$, recursion (9) converges to the minimum value $J(\boldsymbol{\theta}_*)$ asymptotically. Moreover, observe that this convergence is monotonic, since $\lambda_j(1-\mu\lambda_j)^2$ is positive. Following similar arguments as before, the respective time constants for each one of the modes are now,

$$\tau_j^{\text{mse}} = \frac{-1}{2\ln(1-\mu\lambda_j)} \approx \frac{1}{2\mu\lambda_j}.$$

- The previous approximation for the time constant makes crystal clear the role that the step-size, μ, plays in the gradient descent iterative scheme. Its choice is crucial not only to guarantee convergence, but also it determines the speed with which the algorithm converges to the solution. The smaller its value is the slower the convergence rate becomes.
- The other factor which the convergence speed depends on is the eigenvalue structure of the input data covariance matrix.

Gradient Descent: Some Examples

- The aim of the example is to demonstrate what we have said so far, concerning the convergence issues of the gradient descent scheme

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu \left(\boldsymbol{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)} \right).$$

The cross-correlation vector was chosen to be

$$\boldsymbol{p} = [0.05, 0.03]^T,$$

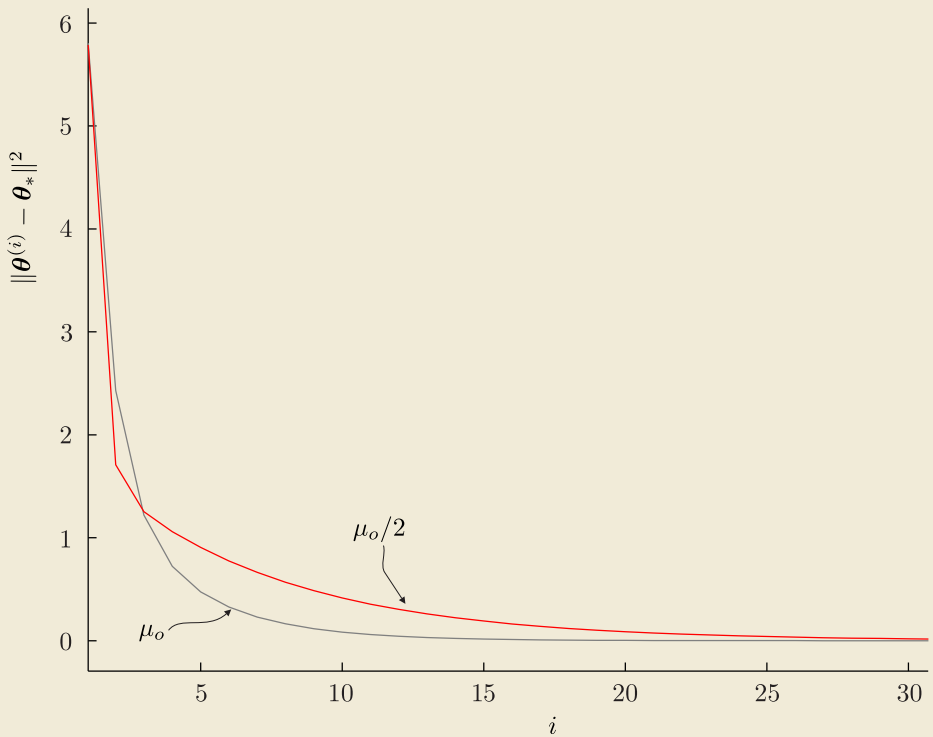
and we consider two different covariance matrices,

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that, for the case of Σ_2, both eigenvalues are qual to 1 and for Σ_1 they are $\lambda_1 = 1$ and $\lambda_2 = 0.1$ (for diagonal matrices the eigenvalues are equal to the diagonal elements of the matrix).

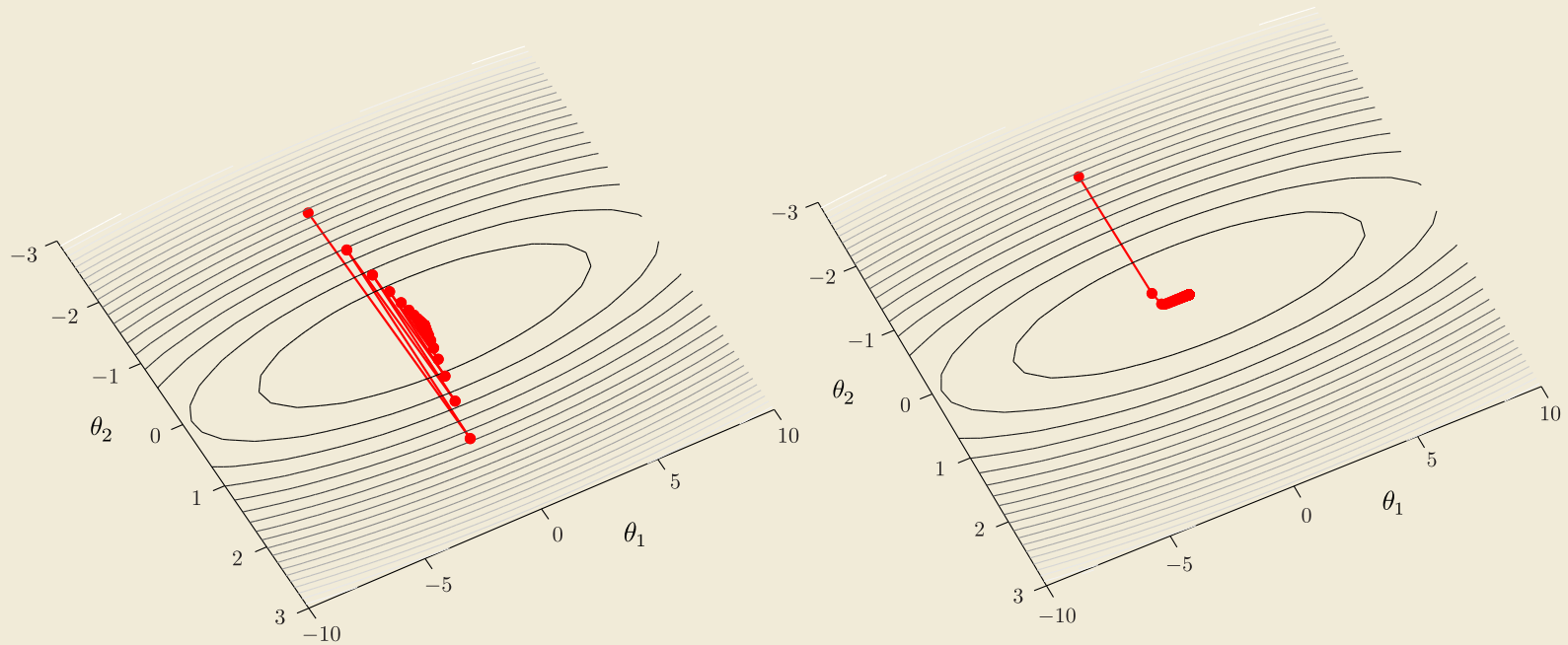
Gradient Descent: Some Examples

- The figure below shows the error curves for two values of μ, for the case of Σ_1; the gray one corresponds to the optimum value ($\mu_o = 1.81$) and the red one to $\mu = \mu_o/2 = 0.9$. Observe the faster convergence towards zero that is achieved by the optimal value. Note that it may happen, as it is the case in this figure, that the initial convergence for some $\mu \neq \mu_o$ to be faster compared to μ_o. What the theory guarantees is that, eventually, the curve corresponding to the optimal will tend to zero faster than for any other value of μ.



Gradient Descent: Some Examples

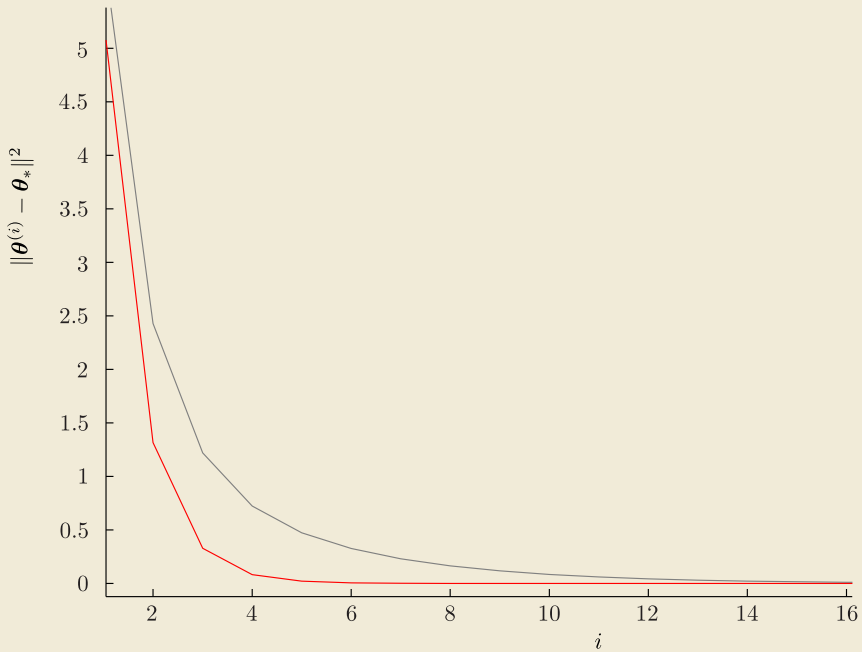
- The figures below show the respective trajectories of the successive estimates in the two-dimensional space, together with the isovalue curves; the latter are ellipses, due to the quadratic nature of the cost function. Observe the zig-zag path, which corresponds to the larger value of $\mu = 1.81$ compared to the smoother one obtained for the smaller step size $\mu = 0.9$



The trajectories of the successive estimates (dots) obtained by the gradient descent algorithm for a) the larger value of $\mu = 1.81$ and b) for the smaller value of $\mu = 0.9$. In b), the trajectory towards the minimum is smooth. In contrast in a), the trajectory consists of zig-zags.

Gradient Descent: Some Examples

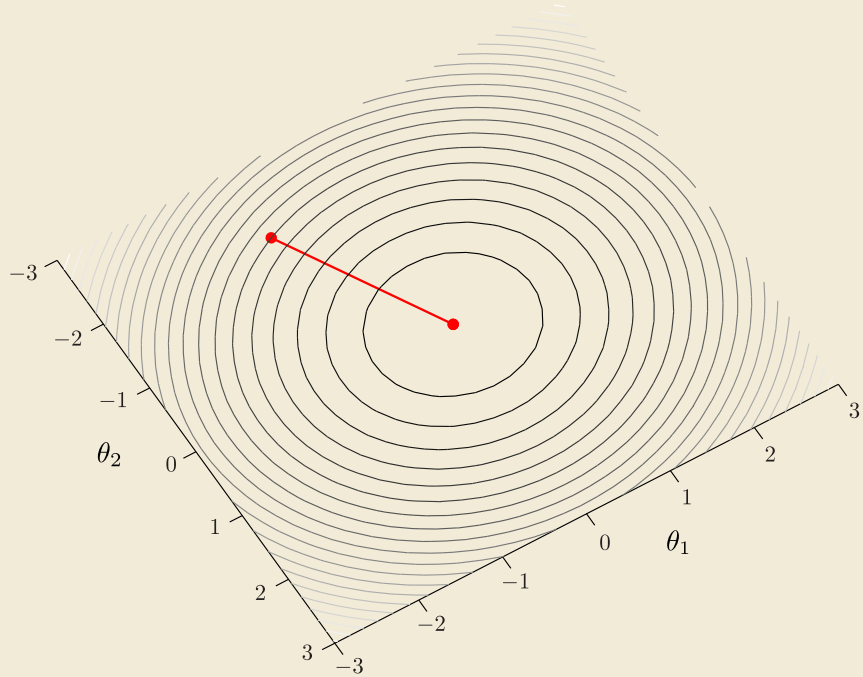
- For comparison reasons, in order to demonstrate the dependence of the convergence speed on the eigenvalues spread, Figure (a) shows the error curves using the same step size, $\mu = 1.81$, for both cases, Σ_1 and Σ_2. Observe that large eigenvalues spread of the input covariance matrix slows down the convergence rate.



(a)

For the same value of $\mu = 1.81$, the error curves for the case of unequal eigenvalues ($\lambda_1 = 1$ and $\lambda_2 = 0.1$) (gray) and for equal eigenvalues ($\lambda_1 = \lambda_2 = 1$).

- Note that, if the eigenvalues of the covariance matrix are equal to, say, λ, the isovalue curves are circles; the optimal step-size in this case is $\mu = 1/\lambda$ and convergence is achieved in only one step, Figure (b).



(b)

When the eigenvalues of the covariance matrix are all equal to a values, λ, the use of the optimal $\mu_o = 1/\lambda$ achieves convergence in one step.

Gradient Descent: Time Varying Step-Sizes

- Time-varying step-sizes: The previous analysis cannot be carried out for the case of an iteration-dependent step-size. It can be shown that, in this case, the gradient descent algorithm converges if
  - $\mu_i \longrightarrow 0$, as $i \longrightarrow \infty$
  - $\sum_{i=1}^{\infty} \mu_i = \infty$.
- A typical example of sequences, which comply with both conditions, are those which satisfy the following:

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty,$$

  as, for example, the sequence,

$$\mu_i = \frac{1}{i}.$$

- The two (sufficient) conditions require that the sequence tends to zero, yet its infinite sum **diverges**. In words, the step-size has to decrease as iterations progress, but not in an aggressive manner; thus, the algorithm remains active for a sufficient number of iterations in order to learn the solution. If the step-size tends to zero very fast, then updates are practically frozen after a few iterations, without the algorithm having acquired enough information so that to get close to the solution.

Gradient Descent: Time Varying Step-Sizes

- Time-varying step-sizes: The previous analysis cannot be carried out for the case of an iteration-dependent step-size. It can be shown that, in this case, the gradient descent algorithm converges if
  - $\mu_i \longrightarrow 0$, as $i \longrightarrow \infty$
  - $\sum_{i=1}^{\infty} \mu_i = \infty$.
- A typical example of sequences, which comply with both conditions, are those which satisfy the following:

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty,$$

  as, for example, the sequence,

$$\mu_i = \frac{1}{i}.$$

- The two (sufficient) conditions require that the sequence tends to zero, yet its infinite sum **diverges**. In words, the step-size has to decrease as iterations progress, but not in an aggressive manner; thus, the algorithm remains active for a sufficient number of iterations in order to learn the solution. If the step-size tends to zero very fast, then updates are practically frozen after a few iterations, without the algorithm having acquired enough information so that to get close to the solution.

Gradient Descent: Time Varying Step-Sizes

- Time-varying step-sizes: The previous analysis cannot be carried out for the case of an iteration-dependent step-size. It can be shown that, in this case, the gradient descent algorithm converges if
  - $\mu_i \longrightarrow 0$, as $i \longrightarrow \infty$
  - $\sum_{i=1}^{\infty} \mu_i = \infty$.
- A typical example of sequences, which comply with both conditions, are those which satisfy the following:

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty,$$

  as, for example, the sequence,

$$\mu_i = \frac{1}{i}.$$

- The two (sufficient) conditions require that the sequence tends to zero, yet its infinite sum **diverges**. In words, the step-size has to decrease as iterations progress, but not in an aggressive manner; thus, the algorithm remains active for a sufficient number of iterations in order to learn the solution. If the step-size tends to zero very fast, then updates are practically frozen after a few iterations, without the algorithm having acquired enough information so that to get close to the solution.

Stochastic Approximation

- Solving for the normal equations as well as using the gradient descent iterative scheme (for the case of the MSE), one has to have access to the second order statistics of the involved processes/variables. However, in most of the cases, this is not known and it has to be approximated using a set of measurements.
- We now turn our attention to algorithms that can learn the statistics iteratively via the training set. Such techniques evolve around the method of stochastic approximation, or the Robbins-Monro algorithm.
- Let us consider the case of a function which is defined in terms of the expected value of another one, i.e.,

$$f(\boldsymbol{\theta}) = \mathbb{E}\big[\phi(\boldsymbol{\theta}, \boldsymbol{\eta})\big], \ \boldsymbol{\theta} \in \mathbb{R}^l,$$

where $\boldsymbol{\eta}$ is a random vector of unknown statistics. The goal is to compute a root of $f(\boldsymbol{\theta})$. If the statistics were known, the expectation could be computed, at least in principle, and one could use any root-finding algorithm to compute the roots. The problem emerges when the statistics is unknown, hence the exact form of $f(\boldsymbol{\theta})$ is not known. All one has at his/her disposal is a sequence of i.i.d observations $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \ldots$.

Stochastic Approximation

- Solving for the normal equations as well as using the gradient descent iterative scheme (for the case of the MSE), one has to have access to the second order statistics of the involved processes/variables. However, in most of the cases, this is not known and it has to be approximated using a set of measurements.
- We now turn our attention to algorithms that can learn the statistics iteratively via the training set. Such techniques evolve around the method of stochastic approximation, or the Robbins-Monro algorithm.
- Let us consider the case of a function which is defined in terms of the expected value of another one, i.e.,

$$f(\boldsymbol{\theta}) = \mathbb{E}\big[\phi(\boldsymbol{\theta}, \boldsymbol{\eta})\big], \ \boldsymbol{\theta} \in \mathbb{R}^l,$$

where $\boldsymbol{\eta}$ is a random vector of unknown statistics. The goal is to compute a root of $f(\boldsymbol{\theta})$. If the statistics were known, the expectation could be computed, at least in principle, and one could use any root-finding algorithm to compute the roots. The problem emerges when the statistics is unknown, hence the exact form of $f(\boldsymbol{\theta})$ is not known. All one has at his/her disposal is a sequence of i.i.d observations $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \ldots$.

Stochastic Approximation

- Solving for the normal equations as well as using the gradient descent iterative scheme (for the case of the MSE), one has to have access to the second order statistics of the involved processes/variables. However, in most of the cases, this is not known and it has to be approximated using a set of measurements.
- We now turn our attention to algorithms that can learn the statistics iteratively via the training set. Such techniques evolve around the method of stochastic approximation, or the Robbins-Monro algorithm.
- Let us consider the case of a function which is defined in terms of the expected value of another one, i.e.,

$$f(\boldsymbol{\theta}) = \mathbb{E}\big[\phi(\boldsymbol{\theta}, \boldsymbol{\eta})\big], \ \boldsymbol{\theta} \in \mathbb{R}^l,$$

where $\boldsymbol{\eta}$ is a random vector of unknown statistics. The goal is to compute a root of $f(\boldsymbol{\theta})$. If the statistics were known, the expectation could be computed, at least in principle, and one could use any root-finding algorithm to compute the roots. The problem emerges when the statistics is unknown, hence the exact form of $f(\boldsymbol{\theta})$ is not known. All one has at his/her disposal is a sequence of i.i.d observations $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \ldots$.

Stochastic Approximation

- The Robbins-Monro algorithm: Robbins and Monro proved that the following iterative scheme

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \phi(\boldsymbol{\theta}_{n-1}, \boldsymbol{\eta}_n),$$

starting from an arbitrary initial condition, $\boldsymbol{\theta}_{-1}$, converges (in probability) to a root of $f(\boldsymbol{\theta})$, under some general conditions and provided that

$$\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \longrightarrow \infty. \tag{10}$$

- In other words, in the previous iterative scheme, we get rid of the expectation operation and use the value of $\phi(\cdot, \cdot)$, which is computed using the **current** observations/measurements and the currently available estimate. That is, the algorithm learns both the statistics as well as the root; two into one! The same comments made in the previous slide for the convergence conditions, met in the iteration dependent step-size case, are valid here too.

Stochastic Approximation

- The Robbins-Monro algorithm: Robbins and Monro proved that the following iterative scheme

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \phi(\boldsymbol{\theta}_{n-1}, \boldsymbol{\eta}_n),$$

starting from an arbitrary initial condition, $\boldsymbol{\theta}_{-1}$, converges (in probability) to a root of $f(\boldsymbol{\theta})$, under some general conditions and provided that

$$\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \longrightarrow \infty. \tag{10}$$

- In other words, in the previous iterative scheme, we get rid of the expectation operation and use the value of $\phi(\cdot, \cdot)$, which is computed using the **current** observations/measurements and the currently available estimate. That is, the algorithm learns both the statistics as well as the root; two into one! The same comments made in the previous slide for the convergence conditions, met in the iteration dependent step-size case, are valid here too.

Stochastic Approximation

- Cost function optimization: In the context of optimizing a general differentiable cost function of the form,

$$J(\boldsymbol{\theta}) = \mathbb{E}\big[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\big],$$

the Robbins-Monro scheme can be mobilized to find a root of the respected gradient, i.e,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}\big[\nabla \mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\big],$$

where the expectation is w.r. to the pair $(\mathrm{y}, \mathbf{x})$. Recall that, such cost functions in the Machine Learning terminology are also known as the expected risk or the expected loss.

- Given the sequence of observations $(y_n, \boldsymbol{x}_n),\ n = 0, 1, \ldots$, the Robbins-Monro recursion now becomes

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \nabla \mathcal{L}(\boldsymbol{\theta}_{n-1}, y_n, \boldsymbol{x}_n).$$

- Let us now assume, for simplicity, that the expected risk has a unique minimum, θ_*. Then, according to Robbins-Monro theorem and using an appropriate sequence μ_n, θ_n will converge to θ_*. However, although this information is important, it is not by itself enough. In practice, one has to seize iterations after a finite number of steps.

Stochastic Approximation

- Cost function optimization: In the context of optimizing a general differentiable cost function of the form,

$$J(\boldsymbol{\theta}) = \mathbb{E}\big[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\big],$$

the Robbins-Monro scheme can be mobilized to find a root of the respected gradient, i.e,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}\big[\nabla \mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\big],$$

where the expectation is w.r. to the pair $(\mathrm{y}, \mathbf{x})$. Recall that, such cost functions in the Machine Learning terminology are also known as the expected risk or the expected loss.

- Given the sequence of observations $(y_n, \boldsymbol{x}_n),\ n = 0, 1, \ldots$, the Robbins-Monro recursion now becomes

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \nabla \mathcal{L}(\boldsymbol{\theta}_{n-1}, y_n, \boldsymbol{x}_n).$$

- Let us now assume, for simplicity, that the expected risk has a unique minimum, θ_*. Then, according to Robbins-Monro theorem and using an appropriate sequence μ_n, θ_n will converge to θ_*. However, although this information is important, it is not by itself enough. In practice, one has to seize iterations after a finite number of steps.

Stochastic Approximation

- Cost function optimization: In the context of optimizing a general differentiable cost function of the form,

$$J(\boldsymbol{\theta}) = \mathbb{E}\big[\mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\big],$$

the Robbins-Monro scheme can be mobilized to find a root of the respected gradient, i.e,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}\big[\nabla \mathcal{L}(\boldsymbol{\theta}, \mathrm{y}, \mathbf{x})\big],$$

where the expectation is w.r. to the pair $(\mathrm{y}, \mathbf{x})$. Recall that, such cost functions in the Machine Learning terminology are also known as the expected risk or the expected loss.

- Given the sequence of observations $(y_n, \boldsymbol{x}_n),\ n = 0, 1, \ldots$, the Robbins-Monro recursion now becomes

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \nabla \mathcal{L}(\boldsymbol{\theta}_{n-1}, y_n, \boldsymbol{x}_n).$$

- Let us now assume, for simplicity, that the expected risk has a unique minimum, $\boldsymbol{\theta}_*$. Then, according to Robbins-Monro theorem and using an appropriate sequence μ_n, $\boldsymbol{\theta}_n$ will converge to $\boldsymbol{\theta}_*$. However, although this information is important, it is not by itself enough. In practice, one has to seize iterations after a finite number of steps.

Stochastic Approximation

- To this end, two quantities are of interest, namely the mean and the covariance matrix of the estimate **at iteration** n, i.e.,

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right], \ \mathrm{Cov}(\boldsymbol{\theta}_n).$$

It can be shown that, if $\mu_n = \mathcal{O}(1/n)$ and assuming that iterations have brought the estimate close to the optimal value, then

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right] = \boldsymbol{\theta}_* + \frac{1}{n}\boldsymbol{c}, \quad \mathrm{Cov}(\boldsymbol{\theta}_n) = \frac{1}{n}V + O(1/n^2),$$

where $\boldsymbol{c}$ and V are constants that depend on the cost function.

- That is, both the mean as well as the standard deviations of the components follow an $\mathcal{O}(1/n)$ pattern. Moreover, these formulae indicate that the parameter vector estimate **fluctuates** around the optimal value.
- This fluctuation depends on the choice of the sequence μ_n, being smaller for smaller values of the step-size sequence. However, μ_n cannot be made to decrease very fast due to the two convergence conditions, as discussed before. This is the price one pays for using the **noisy version** of the gradient and it is the reason that such schemes suffer from relatively slow convergence rates.

- To this end, two quantities are of interest, namely the mean and the covariance matrix of the estimate **at iteration** n, i.e.,

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right], \ \mathrm{Cov}(\boldsymbol{\theta}_n).$$

  It can be shown that, if $\mu_n = \mathcal{O}(1/n)$ and assuming that iterations have brought the estimate close to the optimal value, then

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right] = \boldsymbol{\theta}_* + \frac{1}{n}\boldsymbol{c}, \quad \mathrm{Cov}(\boldsymbol{\theta}_n) = \frac{1}{n}V + O(1/n^2),$$

  where $\boldsymbol{c}$ and V are constants that depend on the cost function.

- That is, both the mean as well as the standard deviations of the components follow an $\mathcal{O}(1/n)$ pattern. Moreover, these formulae indicate that the parameter vector estimate **fluctuates** around the optimal value.

- This fluctuation depends on the choice of the sequence μ_n, being smaller for smaller values of the step-size sequence. However, μ_n cannot be made to decrease very fast due to the two convergence conditions, as discussed before. This is the price one pays for using the **noisy version** of the gradient and it is the reason that such schemes suffer from relatively slow convergence rates.

- To this end, two quantities are of interest, namely the mean and the covariance matrix of the estimate **at iteration** n, i.e.,

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right], \ \mathrm{Cov}(\boldsymbol{\theta}_n).$$

  It can be shown that, if $\mu_n = \mathcal{O}(1/n)$ and assuming that iterations have brought the estimate close to the optimal value, then

$$\mathbb{E}\left[\boldsymbol{\theta}_n\right] = \boldsymbol{\theta}_* + \frac{1}{n}\boldsymbol{c}, \quad \mathrm{Cov}(\boldsymbol{\theta}_n) = \frac{1}{n}V + O(1/n^2),$$

  where $\boldsymbol{c}$ and V are constants that depend on the cost function.

- That is, both the mean as well as the standard deviations of the components follow an $\mathcal{O}(1/n)$ pattern. Moreover, these formulae indicate that the parameter vector estimate **fluctuates** around the optimal value.
- This fluctuation depends on the choice of the sequence μ_n, being smaller for smaller values of the step-size sequence. However, μ_n cannot be made to decrease very fast due to the two convergence conditions, as discussed before. This is the price one pays for using the **noisy version** of the gradient and it is the reason that such schemes suffer from relatively slow convergence rates.

Stochastic Approximation And The MSE Linear Estimation

- Let us apply the Robbins-Monro algorithm to solve for the optimal MSE linear estimator if the covariance matrix and the cross-correlation vector are unknown. We know that the solution corresponds to the root of the gradient of the cost function, which can be written in the form (recall the orthogonality theorem from Chapter 4),

$$\Sigma_x \boldsymbol{\theta} - \boldsymbol{p} = \mathbb{E}\left[\mathbf{x}(\mathbf{x}^T\boldsymbol{\theta} - \mathrm{y})\right] = \mathbf{0}.$$

- Given the training sequence of observations, $(y_n, \boldsymbol{x}_n)$, which are assumed to be i.i.d. drawn from the joint distribution of $(\mathrm{y}, \mathbf{x})$, the Robbins-Monro algorithm becomes,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \boldsymbol{x}_n \left(y_n - \boldsymbol{x}_n^T \boldsymbol{\theta}_{n-1}\right), \tag{11}$$

  which converges to the optimal MSE solution provided that the two sufficient conditions in (10) are satisfied.

- Compare the above Robbins-Monro recursions with the gradient descent one, i.e,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)}\right).$$

  The former equation results from the latter one by dropping out the expectation operations and using an iteration-dependent step-size. Moreover, the iterations in (11) coincide with time updates; time has now explicitly entered into the scene.

Stochastic Approximation And The MSE Linear Estimation

- Let us apply the Robbins-Monro algorithm to solve for the optimal MSE linear estimator if the covariance matrix and the cross-correlation vector are unknown. We know that the solution corresponds to the root of the gradient of the cost function, which can be written in the form (recall the orthogonality theorem from Chapter 4),

$$\Sigma_x \boldsymbol{\theta} - \boldsymbol{p} = \mathbb{E}\left[\mathbf{x}(\mathbf{x}^T\boldsymbol{\theta} - \mathrm{y})\right] = \mathbf{0}.$$

- Given the training sequence of observations, $(y_n, \boldsymbol{x}_n)$, which are assumed to be i.i.d. drawn from the joint distribution of $(\mathrm{y}, \mathbf{x})$, the Robbins-Monro algorithm becomes,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \boldsymbol{x}_n \left(y_n - \boldsymbol{x}_n^T \boldsymbol{\theta}_{n-1}\right), \tag{11}$$

  which converges to the optimal MSE solution provided that the two sufficient conditions in (10) are satisfied.

- Compare the above Robbins-Monro recursions with the gradient descent one, i.e,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)}\right).$$

  The former equation results from the latter one by dropping out the expectation operations and using an iteration-dependent step-size. Moreover, the iterations in (11) coincide with time updates; time has now explicitly entered into the scene.

Stochastic Approximation And The MSE Linear Estimation

- Let us apply the Robbins-Monro algorithm to solve for the optimal MSE linear estimator if the covariance matrix and the cross-correlation vector are unknown. We know that the solution corresponds to the root of the gradient of the cost function, which can be written in the form (recall the orthogonality theorem from Chapter 4),

$$\Sigma_x \boldsymbol{\theta} - \boldsymbol{p} = \mathbb{E}\left[\mathbf{x}(\mathbf{x}^T\boldsymbol{\theta} - \mathrm{y})\right] = \mathbf{0}.$$

- Given the training sequence of observations, $(y_n, \boldsymbol{x}_n)$, which are assumed to be i.i.d. drawn from the joint distribution of $(\mathrm{y}, \mathbf{x})$, the Robbins-Monro algorithm becomes,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \boldsymbol{x}_n \left(y_n - \boldsymbol{x}_n^T \boldsymbol{\theta}_{n-1}\right), \tag{11}$$

  which converges to the optimal MSE solution provided that the two sufficient conditions in (10) are satisfied.

- Compare the above Robbins-Monro recursions with the gradient descent one, i.e,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu\left(\boldsymbol{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)}\right).$$

  The former equation results from the latter one by dropping out the expectation operations and using an iteration-dependent step-size. Moreover, the iterations in (11) coincide with time updates; time has now explicitly entered into the scene.

Stochastic Gradient Descent

- Algorithms such as the one in (11), which result from the generic gradient descent formulation by replacing the expectation by the respective instantaneous observations, are also known as stochastic gradient descent schemes.
- Now that each iteration step coincides with **time** updates, provides us with the spark of starting thinking on modifying such schemes appropriately so that to track time-varying environments.
- All the algorithms to be derived next can also be applied to nonlinear estimation/filtering tasks of the form,

$$\widehat{\mathrm{y}} = \sum_{k=1}^{l} \theta_k \phi_k(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\Phi},$$

and the place of $\mathbf{x}$ is taken by $\boldsymbol{\Phi}$, where

$$\boldsymbol{\Phi} = [\phi_1(\mathbf{x}), \ldots, \phi_l(\mathbf{x})]^T.$$

Stochastic Gradient Descent

- Algorithms such as the one in (11), which result from the generic gradient descent formulation by replacing the expectation by the respective instantaneous observations, are also known as stochastic gradient descent schemes.
- Now that each iteration step coincides with **time** updates, provides us with the spark of starting thinking on modifying such schemes appropriately so that to track time-varying environments.
- All the algorithms to be derived next can also be applied to nonlinear estimation/filtering tasks of the form,

$$\widehat{\mathrm{y}} = \sum_{k=1}^{l} \theta_k \phi_k(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\Phi},$$

and the place of $\mathbf{x}$ is taken by $\boldsymbol{\Phi}$, where

$$\boldsymbol{\Phi} = [\phi_1(\mathbf{x}), \ldots, \phi_l(\mathbf{x})]^T.$$

Stochastic Gradient Descent

- Algorithms such as the one in (11), which result from the generic gradient descent formulation by replacing the expectation by the respective instantaneous observations, are also known as stochastic gradient descent schemes.
- Now that each iteration step coincides with **time** updates, provides us with the spark of starting thinking on modifying such schemes appropriately so that to track time-varying environments.
- All the algorithms to be derived next can also be applied to nonlinear estimation/filtering tasks of the form,

$$\widehat{\mathrm{y}} = \sum_{k=1}^{l} \theta_k \phi_k(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\Phi},$$

and the place of $\mathbf{x}$ is taken by $\boldsymbol{\Phi}$, where

$$\boldsymbol{\Phi} = [\phi_1(\mathbf{x}), \ldots, \phi_l(\mathbf{x})]^T.$$

An Example

- The aim of this example is to demonstrate the convergence of the stochastic gradient descent concerning the mean and the variance of the obtained estimates, as a function of time.
- Data samples were first generated according to the regression model

$$y_n = \boldsymbol{\theta}^T \boldsymbol{x}_n + \eta_n,$$

where, $\boldsymbol{\theta} \in \mathbb{R}^2$ was randomly chosen and then fixed. The elements of $\boldsymbol{x}_n$ were i.i.d generated via a normal distribution $\mathcal{N}(0, 1)$ and η_n is a white noise sequence with variance equal to $\sigma^2 = 0.1$. Then, the observations $(y_n, \boldsymbol{x}_n)$ were used in the recursive scheme in (11) in order to obtain an estimate of $\boldsymbol{\theta}$. The experiment was repeated 200 times and the mean and variance of the obtained estimates were computed, for each iteration step.

An Example

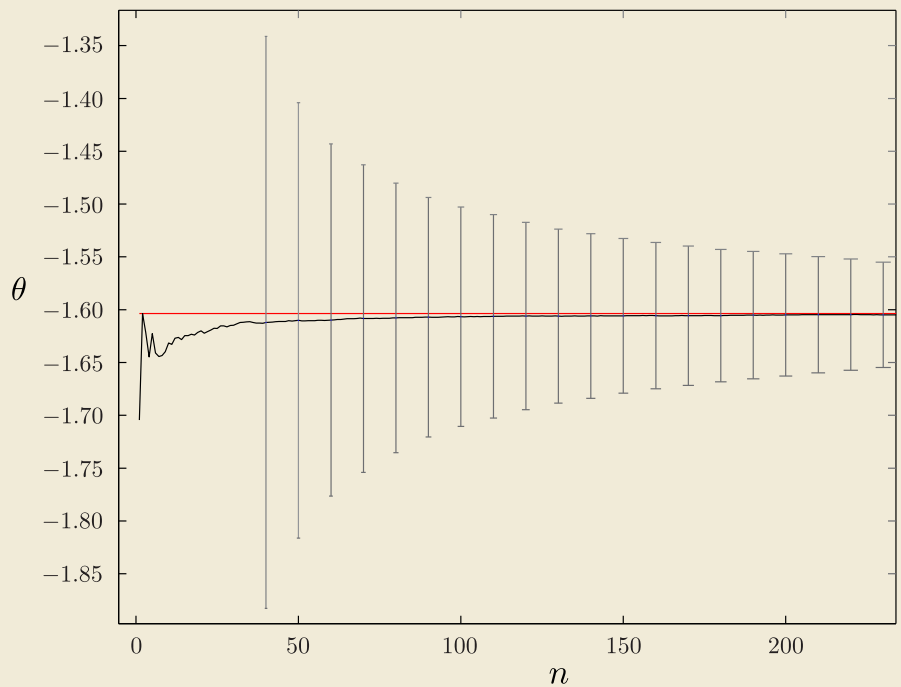
- The aim of this example is to demonstrate the convergence of the stochastic gradient descent concerning the mean and the variance of the obtained estimates, as a function of time.
- Data samples were first generated according to the regression model

$$y_n = \boldsymbol{\theta}^T \boldsymbol{x}_n + \eta_n,$$

where, $\boldsymbol{\theta} \in \mathbb{R}^2$ was randomly chosen and then fixed. The elements of $\boldsymbol{x}_n$ were i.i.d generated via a normal distribution $\mathcal{N}(0,1)$ and η_n is a white noise sequence with variance equal to $\sigma^2 = 0.1$. Then, the observations $(y_n, \boldsymbol{x}_n)$ were used in the recursive scheme in (11) in order to obtain an estimate of $\boldsymbol{\theta}$. The experiment was repeated 200 times and the mean and variance of the obtained estimates were computed, for each iteration step.

An Example

- The figure below shows the resulting curve for one of the parameters (the other one being similar). Observe that the mean values of the estimates tend to the true value, corresponding to the red line, and the standard deviation keeps decreasing as n grows. The step-size was chosen equal $\mu_n = 1/n$.



The red line corresponds to the true value of the unknown parameter. The black curve corresponds to the average over 200 realizations of the experiment. Observe that the mean value converges to the true value. The bars correspond to the respective standard deviation, which keeps decreasing as n grows.

The Least-Mean-Square Adaptive Algorithm (LMS)

- The stochastic gradient algorithm in (11) converges to the optimal Mean-Square-Error solution, provided μ_n satisfies the two convergence conditions. Once the algorithm has converged, it "locks" at the obtained solution. In case the statistics of the involved variables/ unknown parameters start changing, the algorithm cannot track the changes. Note that if such changes occur, the error term,

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n,$$

  gets larger values; however, since μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate.
- This can be overcome if one sets the value of μ_n to a preselected fixed value, μ. The resulting algorithm is the celebrated LMS algorithm
- **The LMS Algorithm**
  - Initialize
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$; other values can also be used.
    - Select the value of μ.
  - **For** $n = 0, 1, \ldots,$ **Do**
    - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu e_n \boldsymbol{x}_n$
  - **End For**

The Least-Mean-Square Adaptive Algorithm (LMS)

- The stochastic gradient algorithm in (11) converges to the optimal Mean-Square-Error solution, provided μ_n satisfies the two convergence conditions. Once the algorithm has converged, it "locks" at the obtained solution. In case the statistics of the involved variables/ unknown parameters start changing, the algorithm cannot track the changes. Note that if such changes occur, the error term,

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n,$$

  gets larger values; however, since μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate.
- This can be overcome if one sets the value of μ_n to a preselected fixed value, μ. The resulting algorithm is the celebrated LMS algorithm
- **The LMS Algorithm**
  - Initialize
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$; other values can also be used.
    - Select the value of μ.
  - **For** $n = 0, 1, \ldots,$ **Do**
    - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu e_n \boldsymbol{x}_n$
  - **End For**

The Least-Mean-Square Adaptive Algorithm (LMS)

- The stochastic gradient algorithm in (11) converges to the optimal Mean-Square-Error solution, provided μ_n satisfies the two convergence conditions. Once the algorithm has converged, it "locks" at the obtained solution. In case the statistics of the involved variables/ unknown parameters start changing, the algorithm cannot track the changes. Note that if such changes occur, the error term,

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n,$$

  gets larger values; however, since μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate.
- This can be overcome if one sets the value of μ_n to a preselected fixed value, μ. The resulting algorithm is the celebrated LMS algorithm
- **The LMS Algorithm**
  - Initialize
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$; other values can also be used.
    - Select the value of μ.
  - **For** $n = 0, 1, \ldots,$ **Do**
    - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu e_n \boldsymbol{x}_n$
  - **End For**

The Least-Mean-Square Adaptive Algorithm (LMS)

- In case the input is a time series, denoted as $\mathrm{u}_n, \mathrm{u}_{n-1}, \ldots$ the initialization also involves the samples, $u_{-1}, \ldots, u_{-l+1} = 0$, in order to form the input vectors, $\boldsymbol{u}_n, n = 0, 1, \ldots, l-2$. The complexity of the algorithm amounts to $2l$ multiplications/additions (MADs) per time update. We have assumed that observations start arriving at time instant $n = 0$, to be in line with most references treating the LMS.
- Assume that the algorithm has converged close to the solution; then the error term is expected to take small values and thus the updates will remain close to the solution. If now the statistics and/or the system parameters start changing, the error values are expected to increase. Given that μ has a constant value, the algorithm has now the "**agility**" to update the estimates so that to "**push**" the error to lower values.
- This small variation of the iterative scheme has important implications. The resulting algorithm is **no more** a member of the Robbins-Monro stochastic approximation family. Thus, one has to study its convergence conditions as well as its performance properties. Moreover, since the algorithm has now the potential to track changes in time-varying environments one has to study its performance in non-stationary environments; this is associated to what is known as the tracking performance of the algorithm.

The Least-Mean-Square Adaptive Algorithm (LMS)

- In case the input is a time series, denoted as $\mathrm{u}_n, \mathrm{u}_{n-1}, \ldots$ the initialization also involves the samples, $u_{-1}, \ldots, u_{-l+1} = 0$, in order to form the input vectors, $\boldsymbol{u}_n, n = 0, 1, \ldots, l-2$. The complexity of the algorithm amounts to $2l$ multiplications/additions (MADs) per time update. We have assumed that observations start arriving at time instant $n = 0$, to be in line with most references treating the LMS.
- Assume that the algorithm has converged close to the solution; then the error term is expected to take small values and thus the updates will remain close to the solution. If now the statistics and/or the system parameters start changing, the error values are expected to increase. Given that μ has a constant value, the algorithm has now the "**agility**" to update the estimates so that to "**push**" the error to lower values.
- This small variation of the iterative scheme has important implications. The resulting algorithm is **no more** a member of the Robbins-Monro stochastic approximation family. Thus, one has to study its convergence conditions as well as its performance properties. Moreover, since the algorithm has now the potential to track changes in time-varying environments one has to study its performance in non-stationary environments; this is associated to what is known as the tracking performance of the algorithm.

The Least-Mean-Square Adaptive Algorithm (LMS)

- In case the input is a time series, denoted as $\mathrm{u}_n, \mathrm{u}_{n-1}, \ldots$ the initialization also involves the samples, $u_{-1}, \ldots, u_{-l+1} = 0$, in order to form the input vectors, $\boldsymbol{u}_n, n = 0, 1, \ldots, l-2$. The complexity of the algorithm amounts to $2l$ multiplications/additions (MADs) per time update. We have assumed that observations start arriving at time instant $n = 0$, to be in line with most references treating the LMS.
- Assume that the algorithm has converged close to the solution; then the error term is expected to take small values and thus the updates will remain close to the solution. If now the statistics and/or the system parameters start changing, the error values are expected to increase. Given that μ has a constant value, the algorithm has now the "**agility**" to update the estimates so that to "**push**" the error to lower values.
- This small variation of the iterative scheme has important implications. The resulting algorithm is **no more** a member of the Robbins-Monro stochastic approximation family. Thus, one has to study its convergence conditions as well as its performance properties. Moreover, since the algorithm has now the potential to track changes in time-varying environments one has to study its performance in non-stationary environments; this is associated to what is known as the tracking performance of the algorithm.

Convergence And Steady-State Performance of the LMS in Stationary Environments

- Our focus now shifts to the study the performance of the LMS in stationary environments. That is, to answer the questions: a) does the scheme converge and under which conditions and b) if it converges, where does it converge. Although we introduced the scheme having in mind non-stationary environments, still we have to know how it behaves under stationarity; after all, the environment can change very slowly and it can be considered "locally" stationary.
- The convergence properties of the LMS, as well as of any other online/adaptive algorithm, is related to its transient characteristics; that is, the period from the initial estimate till the algorithm reaches a steady-state mode of operation. The latter refers to the period, after convergence, where the statistical properties of the estimator do not change with time.
- In general, analyzing the transient performance of an online algorithm is a formidable task indeed. This is also true even for the very simple structure of the LMS algorithm. The LMS update recursions are equivalent to a time-varying, nonlinear and stochastic in nature estimator.

Convergence And Steady-State Performance of the LMS in Stationary Environments

- Our focus now shifts to the study the performance of the LMS in stationary environments. That is, to answer the questions: a) does the scheme converge and under which conditions and b) if it converges, where does it converge. Although we introduced the scheme having in mind non-stationary environments, still we have to know how it behaves under stationarity; after all, the environment can change very slowly and it can be considered "locally" stationary.
- The convergence properties of the LMS, as well as of any other online/adaptive algorithm, is related to its transient characteristics; that is, the period from the initial estimate till the algorithm reaches a steady-state mode of operation. The latter refers to the period, after convergence, where the statistical properties of the estimator do not change with time.
- In general, analyzing the transient performance of an online algorithm is a formidable task indeed. This is also true even for the very simple structure of the LMS algorithm. The LMS update recursions are equivalent to a time-varying, nonlinear and stochastic in nature estimator.

Convergence And Steady-State Performance of the LMS in Stationary Environments

- Our focus now shifts to the study the performance of the LMS in stationary environments. That is, to answer the questions: a) does the scheme converge and under which conditions and b) if it converges, where does it converge. Although we introduced the scheme having in mind non-stationary environments, still we have to know how it behaves under stationarity; after all, the environment can change very slowly and it can be considered "locally" stationary.
- The convergence properties of the LMS, as well as of any other online/adaptive algorithm, is related to its transient characteristics; that is, the period from the initial estimate till the algorithm reaches a steady-state mode of operation. The latter refers to the period, after convergence, where the statistical properties of the estimator do not change with time.
- In general, analyzing the transient performance of an online algorithm is a formidable task indeed. This is also true even for the very simple structure of the LMS algorithm. The LMS update recursions are equivalent to a time-varying, nonlinear and stochastic in nature estimator.

Convergence of the Parameter Error Vector

- Convergence in the mean: The LMS is a variant of the stochastic gradient descent algorithm for the solution of the MSE linear estimation task. Let us assume that $\boldsymbol{\theta}_*$ is the solution of the respective normal equations, $\Sigma_x \boldsymbol{\theta} = \boldsymbol{p}$. Then, it can be shown that the estimator, θ_n, which results from the LMS, converges in the mean to the MSE solution, i.e.,

$$\mathbb{E}[\theta_n] \longrightarrow \boldsymbol{\theta}_*, \text{ as } n \longrightarrow \infty,$$

  provided that

$$0 < \mu < \frac{2}{\lambda_{\max}},$$

  where $\lambda_{\max}$ is the maximum eigenvalue of Σ_x. Recall that the gradient algorithm in (2), for the solution of the MSE, converges under the same condition to $\boldsymbol{\theta}_*$. The LMS converges only in the mean. This is the price one pays for using an estimate of the gradient and not the gradient itself.
- Thus, by fixing the value of the step-size to be a constant, we lose something; the obtained estimates, even after convergence, hover around the optimal solution.

Convergence of the Parameter Error Vector

- Convergence in the mean: The LMS is a variant of the stochastic gradient descent algorithm for the solution of the MSE linear estimation task. Let us assume that $\boldsymbol{\theta}_*$ is the solution of the respective normal equations, $\Sigma_x \boldsymbol{\theta} = \boldsymbol{p}$. Then, it can be shown that the estimator, θ_n, which results from the LMS, converges in the mean to the MSE solution, i.e.,

$$\mathbb{E}[\theta_n] \longrightarrow \boldsymbol{\theta}_*, \text{ as } n \longrightarrow \infty,$$

  provided that

$$0 < \mu < \frac{2}{\lambda_{\max}},$$

  where $\lambda_{\max}$ is the maximum eigenvalue of Σ_x. Recall that the gradient algorithm in (2), for the solution of the MSE, converges under the same condition to $\boldsymbol{\theta}_*$. The LMS converges only in the mean. This is the price one pays for using an estimate of the gradient and not the gradient itself.
- Thus, by fixing the value of the step-size to be a constant, we lose something; the obtained estimates, even after convergence, hover around the optimal solution.

Convergence of the Parameter Error Vector

- **Proof**: Define

$$\boldsymbol{c}_n := \boldsymbol{\theta}_n - \boldsymbol{\theta}_*,$$

where $\boldsymbol{\theta}_*$ is the optimal solution resulting from the normal equations. The LMS update recursion can now be written as

$$\boldsymbol{c}_n = \boldsymbol{c}_{n-1} + \mu \boldsymbol{x}_n (y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n + \boldsymbol{\theta}_*^T \boldsymbol{x}_n - \boldsymbol{\theta}_*^T \boldsymbol{x}_n).$$

- Since we are going to study the statistical properties of the obtained estimates, we have to switch our notation to involve the respective random variables. Then we can write that,

$$\begin{aligned} \mathbf{c}_n &= \mathbf{c}_{n-1} + \mu \mathbf{x}(\mathrm{y} - \theta_{n-1}^T \mathbf{x} + \boldsymbol{\theta}_*^T \mathbf{x} - \boldsymbol{\theta}_*^T \mathbf{x}) \\ &= \mathbf{c}_{n-1} - \mu \mathbf{x}\mathbf{x}^T \mathbf{c}_{n-1} + \mu \mathbf{x} \mathrm{e}_* \\ &= (I - \mu \mathbf{x}\mathbf{x}^T)\mathbf{c}_{n-1} + \mu \mathbf{x} \mathrm{e}_*, \end{aligned}$$

where

$$\mathrm{e}_* = \mathrm{y} - \boldsymbol{\theta}_*^T \mathbf{x},$$

is the error random variable associated with the optimal $\boldsymbol{\theta}_*$. Taking expectations on both sides, we obtain,

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}\left[(I - \mu \mathbf{x}\mathbf{x}^T)\mathbf{c}_{n-1}\right] + \mu \mathbb{E}[\mathbf{x}\mathrm{e}_*]. \tag{12}$$

In order to proceed, it is time to introduce assumptions.

Convergence of the Parameter Error Vector

- **Proof**: Define

$$\boldsymbol{c}_n := \boldsymbol{\theta}_n - \boldsymbol{\theta}_*,$$

where $\boldsymbol{\theta}_*$ is the optimal solution resulting from the normal equations. The LMS update recursion can now be written as

$$\boldsymbol{c}_n = \boldsymbol{c}_{n-1} + \mu \boldsymbol{x}_n (y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n + \boldsymbol{\theta}_*^T \boldsymbol{x}_n - \boldsymbol{\theta}_*^T \boldsymbol{x}_n).$$

- Since we are going to study the statistical properties of the obtained estimates, we have to switch our notation to involve the respective random variables. Then we can write that,

$$\begin{aligned} \mathbf{c}_n &= \mathbf{c}_{n-1} + \mu \mathbf{x}(\mathrm{y} - \boldsymbol{\theta}_{n-1}^T \mathbf{x} + \boldsymbol{\theta}_*^T \mathbf{x} - \boldsymbol{\theta}_*^T \mathbf{x}) \\ &= \mathbf{c}_{n-1} - \mu \mathbf{x}\mathbf{x}^T \mathbf{c}_{n-1} + \mu \mathbf{x} \mathrm{e}_* \\ &= (I - \mu \mathbf{x}\mathbf{x}^T)\mathbf{c}_{n-1} + \mu \mathbf{x} \mathrm{e}_*, \end{aligned}$$

where

$$\mathrm{e}_* = \mathrm{y} - \boldsymbol{\theta}_*^T \mathbf{x},$$

is the error random variable associated with the optimal $\boldsymbol{\theta}_*$. Taking expectations on both sides, we obtain,

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}\left[(I - \mu \mathbf{x}\mathbf{x}^T)\mathbf{c}_{n-1}\right] + \mu \mathbb{E}[\mathbf{x}\mathrm{e}_*]. \tag{12}$$

In order to proceed, it is time to introduce assumptions.

Convergence of the Parameter Error Vector

- **Proof**: Define

$$\boldsymbol{c}_n := \boldsymbol{\theta}_n - \boldsymbol{\theta}_*,$$

where $\boldsymbol{\theta}_*$ is the optimal solution resulting from the normal equations. The LMS update recursion can now be written as

$$\boldsymbol{c}_n = \boldsymbol{c}_{n-1} + \mu \boldsymbol{x}_n (y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n + \boldsymbol{\theta}_*^T \boldsymbol{x}_n - \boldsymbol{\theta}_*^T \boldsymbol{x}_n).$$

- Since we are going to study the statistical properties of the obtained estimates, we have to switch our notation to involve the respective random variables. Then we can write that,

$$\begin{aligned} \mathbf{c}_n &= \mathbf{c}_{n-1} + \mu \mathbf{x}(\mathrm{y} - \boldsymbol{\theta}_{n-1}^T \mathbf{x} + \boldsymbol{\theta}_*^T \mathbf{x} - \boldsymbol{\theta}_*^T \mathbf{x}) \\ &= \mathbf{c}_{n-1} - \mu \mathbf{x}\mathbf{x}^T \mathbf{c}_{n-1} + \mu \mathbf{x} \mathrm{e}_* \\ &= (I - \mu \mathbf{x}\mathbf{x}^T)\mathbf{c}_{n-1} + \mu \mathbf{x} \mathrm{e}_*, \end{aligned}$$

where

$$\mathrm{e}_* = \mathrm{y} - \boldsymbol{\theta}_*^T \mathbf{x},$$

is the error random variable associated with the optimal $\boldsymbol{\theta}_*$. Taking expectations on both sides, we obtain,

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}\left[(I - \mu \mathbf{x}\mathbf{x}^T)\mathbf{c}_{n-1}\right] + \mu \mathbb{E}[\mathbf{x}\mathrm{e}_*]. \tag{12}$$

In order to proceed, it is time to introduce assumptions.

Convergence of the Parameter Error Vector

- Assumption 1: The involved random variables are jointly linked via the regression model,

$$\mathrm{y} = \boldsymbol{\theta}_o^T \mathbf{x} + \eta,$$

where η is the noise variable with variance σ_η^2 and it is assumed to be independent of $\mathbf{x}$. Moreover, successive samples η_n, that generate the data, are assumed to be i.i.d. We know form chapter 4 that, in this case, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$, and $\sigma_{e_*}^2 = \sigma_\eta^2$. Also, due to the orthogonality condition, $\mathbb{E}[\mathbf{x}\mathrm{e}_*] = \mathbf{0}$. In addition, a stronger condition will be adopted, and e_* and $\mathbf{x}$ will be assumed to be statistically independent. This is justified by the fact that under the above model, $e_{*,n} = \eta_n$, and the noise sequence has been assumed to be independent of the input.

Convergence of the Parameter Error Vector

- Assumption 1: The involved random variables are jointly linked via the regression model,

$$\mathrm{y} = \boldsymbol{\theta}_o^T \mathbf{x} + \eta,$$

where η is the noise variable with variance σ_η^2 and it is assumed to be independent of $\mathbf{x}$. Moreover, successive samples η_n, that generate the data, are assumed to be i.i.d. We know form chapter 4 that, in this case, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$, and $\sigma_{e_*}^2 = \sigma_\eta^2$. Also, due to the orthogonality condition, $\mathbb{E}[\mathbf{x}\mathrm{e}_*] = \mathbf{0}$. In addition, a stronger condition will be adopted, and e_* and $\mathbf{x}$ will be assumed to be statistically independent. This is justified by the fact that under the above model, $e_{*,n} = \eta_n$, and the noise sequence has been assumed to be independent of the input.

- Assumption 2- **Independence Assumption**: Assume that $\mathbf{c}_{n-1}$ is statistically independent on both $\mathbf{x}$ and e_*. No doubt this is a strong assumption. It will be adopted in order to simplify computations.

Convergence of the Parameter Error Vector

- Assumption 1: The involved random variables are jointly linked via the regression model,

$$\mathrm{y} = \boldsymbol{\theta}_o^T \mathbf{x} + \eta,$$

where η is the noise variable with variance σ_η^2 and it is assumed to be independent of $\mathbf{x}$. Moreover, successive samples η_n, that generate the data, are assumed to be i.i.d. We know form chapter 4 that, in this case, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$, and $\sigma_{e_*}^2 = \sigma_\eta^2$. Also, due to the orthogonality condition, $\mathbb{E}[\mathbf{x}\mathrm{e}_*] = \mathbf{0}$. In addition, a stronger condition will be adopted, and e_* and $\mathbf{x}$ will be assumed to be statistically independent. This is justified by the fact that under the above model, $e_{*,n} = \eta_n$, and the noise sequence has been assumed to be independent of the input.

- Assumption 2- **Independence Assumption**: Assume that $\mathbf{c}_{n-1}$ is statistically independent on both $\mathbf{x}$ and e_*. No doubt this is a strong assumption. It will be adopted in order to simplify computations.

- Having adopted the previous assumptions, (12) becomes

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}\left[\left(I - \mu \mathbf{x}\mathbf{x}^T\right)\mathbf{c}_{n-1}\right] = (I - \mu\Sigma_x)\mathbb{E}[\mathbf{c}_{n-1}].$$

In the sequel, we follow similar arguments as we did for its gradient descent counterpart.

Convergence of the Parameter Error Vector

- Recall, $\Sigma_x = Q\Lambda Q^T$ and define $\mathbf{v}_n = Q^T\mathbf{c}_n$. Then, plugging into

$$\mathbb{E}[\mathbf{c}_n] = (I - \mu\Sigma_x)\mathbb{E}[\mathbf{c}_{n-1}],$$

we finally obtain

$$\mathbb{E}[\mathbf{v}_n] = (I - \mu\Lambda)\mathbb{E}[\mathbf{v}_{n-1}].$$

The last equation leads to,

$$\mathbb{E}[\boldsymbol{\theta}_n] \longrightarrow \boldsymbol{\theta}_*, \text{ as } n \longrightarrow \infty,$$

provided that

$$0 < \mu < \frac{2}{\lambda_{\max}},$$

and the claim has been proved.

Covariance and Excess Mean-Square Error

- Convergence in the mean is important, but it does not say much by itself. One has to get extra information concerning the involved variances. It can be shown that the covariance matrix of the parameter error vector,

$$\Sigma_c := \mathbb{E}[\mathbf{c}\mathbf{c}^T],$$

remains bounded as $n \longrightarrow \infty$, provided that

$$\mu < \frac{2}{\text{trace}\{\Sigma_x\}}.$$

- Excess MSE and misadjustment: We know that the minimum MSE is achieved at θ_*. Any other weight vector results in higher values of the squared error. We have already said that in the steady-state, the estimates obtained via the LMS fluctuate randomly around θ_*; thus, the mean-square error will be larger than the minimum $J_{\min}$. This "extra" error power, denoted as J_{exc}, is known as the excess mean-square error. Also the ratio

$$\mathcal{M} := \frac{J_{\text{exc}}}{J_{\min}},$$

is known as the misadjustment.

Covariance and Excess Mean-Square Error

- Convergence in the mean is important, but it does not say much by itself. One has to get extra information concerning the involved variances. It can be shown that the covariance matrix of the parameter error vector,

$$\Sigma_c := \mathbb{E}[\mathbf{c}\mathbf{c}^T],$$

remains bounded as $n \longrightarrow \infty$, provided that

$$\mu < \frac{2}{\text{trace}\{\Sigma_x\}}.$$

- Excess MSE and misadjustment: We know that the minimum MSE is achieved at $\boldsymbol{\theta}_*$. Any other weight vector results in higher values of the squared error. We have already said that in the steady-state, the estimates obtained via the LMS fluctuate randomly around $\boldsymbol{\theta}_*$; thus, the mean-square error will be larger than the minimum $J_{\min}$. This "extra" error power, denoted as J_{exc}, is known as the excess mean-square error. Also the ratio

$$\mathcal{M} := \frac{J_{\text{exc}}}{J_{\min}},$$

is known as the misadjustment.

Covariance and Excess Mean-Square Error

- It can be shown, under a number of assumptions (details in the text) that,

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Moreover, asymptotically, at the steady-state, which is defined as the state where,

$$\mathbb{E}[\boldsymbol{\theta}_n] = \mathbb{E}[\boldsymbol{\theta}_{n-1}] = \text{Constant, and } \Sigma_{\theta,n} = \Sigma_{\theta,n-1} = \text{Constant},$$

and for small values of μ, we get

$$J_{\text{exc},\infty} \simeq \frac{1}{2}\mu\sigma_n^2 \text{trace}\{\Sigma_x\}, \quad \mathcal{M} \simeq \frac{1}{2}\mu \text{trace}\{\Sigma_x\}.$$

- Time constant: It turns out that the time constant for the jth mode for the LMS is given by

$$\tau_j^{\text{LMS}} \simeq \frac{1}{2\mu\lambda_j}.$$

- That is, the time constant for each one of the modes is inversely proportional to μ. Hence, the slower the rate of convergence (small values of μ) the lower the misadjustment and vice versa. Viewing it differently, the more time the algorithm spends on learning, prior to reaching the steady state, the smaller its deviation from the optimal is.

Covariance and Excess Mean-Square Error

- It can be shown, under a number of assumptions (details in the text) that,

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Moreover, asymptotically, at the steady-state, which is defined as the state where,

$$\mathbb{E}[\boldsymbol{\theta}_n] = \mathbb{E}[\boldsymbol{\theta}_{n-1}] = \text{Constant, and } \Sigma_{\theta,n} = \Sigma_{\theta,n-1} = \text{Constant},$$

and for small values of μ, we get

$$J_{\text{exc},\infty} \simeq \frac{1}{2}\mu\sigma_n^2 \text{trace}\{\Sigma_x\}, \quad \mathcal{M} \simeq \frac{1}{2}\mu \text{trace}\{\Sigma_x\}.$$

- Time constant: It turns out that the time constant for the jth mode for the LMS is given by

$$\tau_j^{\text{LMS}} \simeq \frac{1}{2\mu\lambda_j}.$$

- That is, the time constant for each one of the modes is inversely proportional to μ. Hence, the slower the rate of convergence (small values of μ) the lower the misadjustment and vice versa. Viewing it differently, the more time the algorithm spends on learning, prior to reaching the steady state, the smaller its deviation from the optimal is.

Covariance and Excess Mean-Square Error

- It can be shown, under a number of assumptions (details in the text) that,

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

  Moreover, asymptotically, at the steady-state, which is defined as the state where,

$$\mathbb{E}[\boldsymbol{\theta}_n] = \mathbb{E}[\boldsymbol{\theta}_{n-1}] = \text{Constant, and } \Sigma_{\theta,n} = \Sigma_{\theta,n-1} = \text{Constant},$$

  and for small values of μ, we get

$$J_{\text{exc},\infty} \simeq \frac{1}{2}\mu\sigma_n^2 \text{trace}\{\Sigma_x\}, \quad \mathcal{M} \simeq \frac{1}{2}\mu \text{trace}\{\Sigma_x\}.$$

- Time constant: It turns out that the time constant for the jth mode for the LMS is given by

$$\tau_j^{\text{LMS}} \simeq \frac{1}{2\mu\lambda_j}.$$

- That is, the time constant for each one of the modes is inversely proportional to μ. Hence, the slower the rate of convergence (small values of μ) the lower the misadjustment and vice versa. Viewing it differently, the more time the algorithm spends on learning, prior to reaching the steady state, the smaller its deviation from the optimal is.

The LMS: Some Simulation Examples

- The goal of this example is to demonstrate the sensitivity of the convergence rate of the LMS on the eigenvalues spread of the input covariance matrix. To this end, two sets of data were generated, in the context of the regression task,

$$y_n = \boldsymbol{\theta}^T \boldsymbol{x}_n + \eta_n.$$

  The parameters, $\boldsymbol{\theta} \in \mathbb{R}^{10}$, were randomly chosen from a $\mathcal{N}(0, 1)$ and then frozen.

- For the first set, the input vectors were formed by a white noise sequence with samples i.i.d drawn from a $\mathcal{N}(0, 1)$. Thus, the input covariance matrix was diagonal with all the elements being equal to the corresponding noise variance (as explained in Chapter 2). The noise samples, η_n, were also i.i.d drawn with zero mean and variance $\sigma^2 = 0.01$.

- For the second set, the input vectors were formed by an AR(1) process with coefficient equal to $a_1 = 0.85$ and the corresponding white noise excitation was of variance equal to 1 (Chapter 2). Thus, the input covariance matrix is no more diagonal and the eigenvalues are not equal. The LMS was run on both cases with the same step size $\mu = 0.01$.

The LMS: Some Simulation Examples

- The goal of this example is to demonstrate the sensitivity of the convergence rate of the LMS on the eigenvalues spread of the input covariance matrix. To this end, two sets of data were generated, in the context of the regression task,

$$y_n = \boldsymbol{\theta}^T \boldsymbol{x}_n + \eta_n.$$

  The parameters, $\boldsymbol{\theta} \in \mathbb{R}^{10}$, were randomly chosen from a $\mathcal{N}(0, 1)$ and then frozen.
- For the first set, the input vectors were formed by a white noise sequence with samples i.i.d drawn from a $\mathcal{N}(0, 1)$. Thus, the input covariance matrix was diagonal with all the elements being equal to the corresponding noise variance (as explained in Chapter 2). The noise samples, η_n, were also i.i.d drawn with zero mean and variance $\sigma^2 = 0.01$.
- For the second set, the input vectors were formed by an AR(1) process with coefficient equal to $a_1 = 0.85$ and the corresponding white noise excitation was of variance equal to 1 (Chapter 2). Thus, the input covariance matrix is no more diagonal and the eigenvalues are not equal. The LMS was run on both cases with the same step size $\mu = 0.01$.

The LMS: Some Simulation Examples

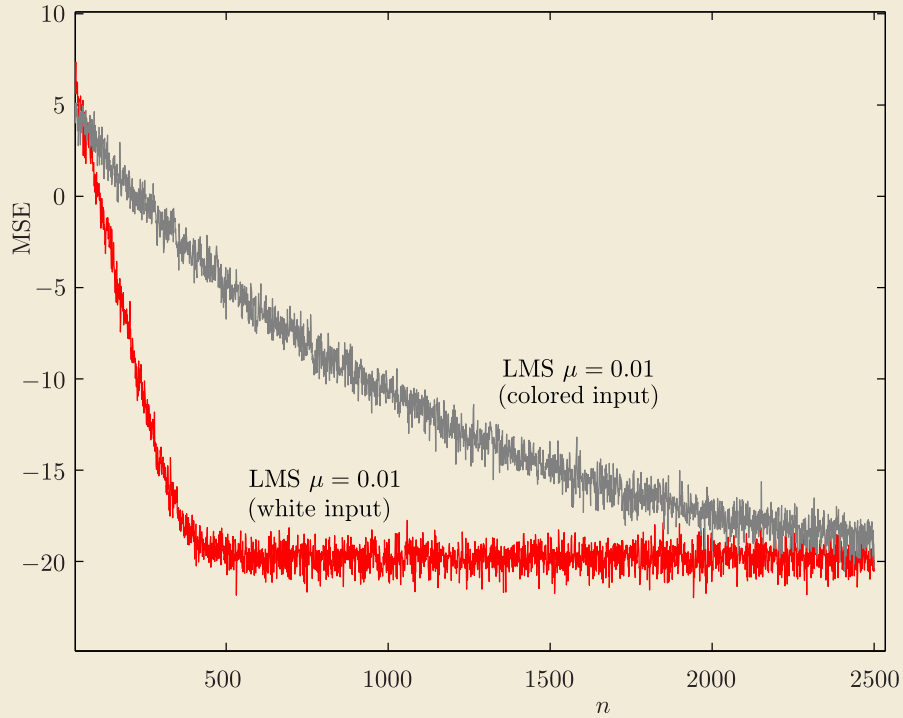
- The goal of this example is to demonstrate the sensitivity of the convergence rate of the LMS on the eigenvalues spread of the input covariance matrix. To this end, two sets of data were generated, in the context of the regression task,

$$y_n = \boldsymbol{\theta}^T \boldsymbol{x}_n + \eta_n.$$

  The parameters, $\boldsymbol{\theta} \in \mathbb{R}^{10}$, were randomly chosen from a $\mathcal{N}(0, 1)$ and then frozen.
- For the first set, the input vectors were formed by a white noise sequence with samples i.i.d drawn from a $\mathcal{N}(0, 1)$. Thus, the input covariance matrix was diagonal with all the elements being equal to the corresponding noise variance (as explained in Chapter 2). The noise samples, η_n, were also i.i.d drawn with zero mean and variance $\sigma^2 = 0.01$.
- For the second set, the input vectors were formed by an AR(1) process with coefficient equal to $a_1 = 0.85$ and the corresponding white noise excitation was of variance equal to 1 (Chapter 2). Thus, the input covariance matrix is no more diagonal and the eigenvalues are not equal. The LMS was run on both cases with the same step size $\mu = 0.01$.

The LMS: Some Simulation Examples

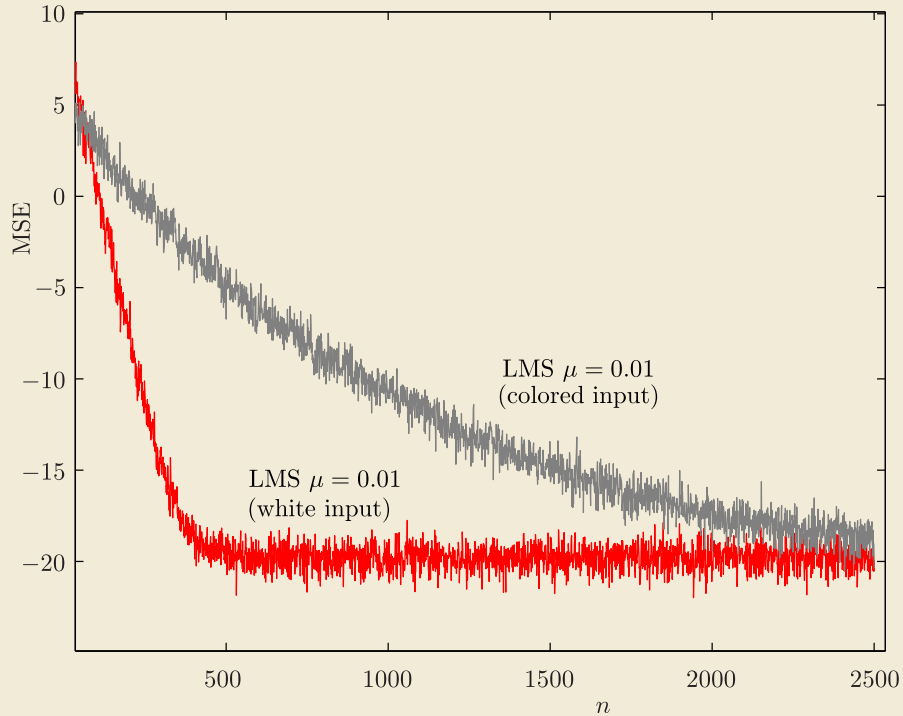
- The figure below summarizes the results. The vertical axis (denoted as MSE) shows the squared error, e_n^2, and the horizontal axis the time instants (iterations) n. Note that both curves level out at the same error floor (same misadjustment). However, the convergence rate for the case of the white noise sequence is significantly higher.



- When comparing convergence performance of different algorithms, either all algorithms should converge to the same error floor and compare the respective convergence rates, or all algorithms should have the same convergence rate and compare respective error floors.

The LMS: Some Simulation Examples

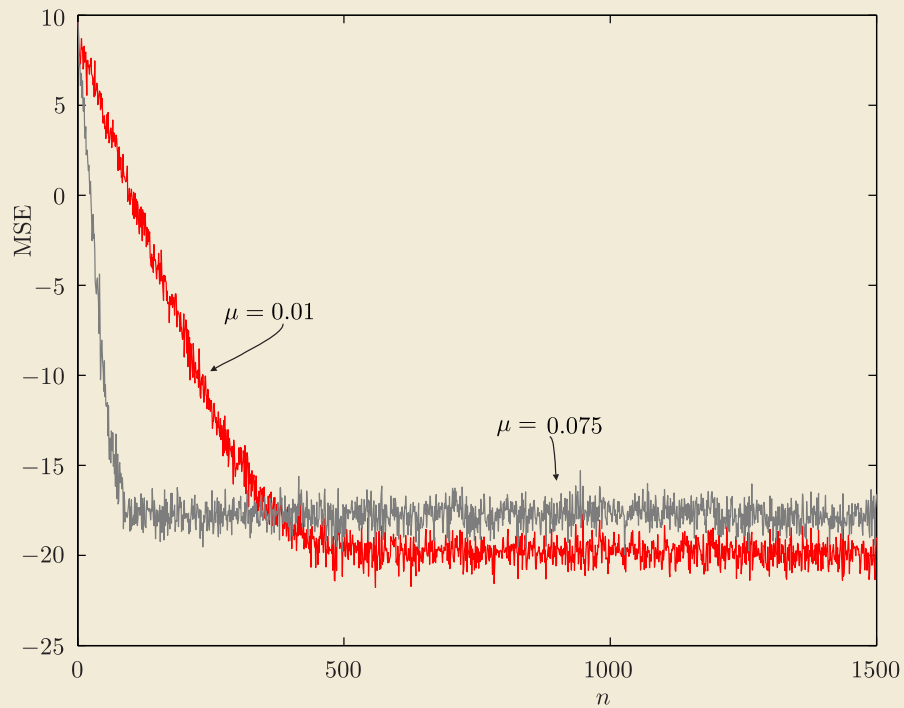
- The figure below summarizes the results. The vertical axis (denoted as MSE) shows the squared error, e_n^2, and the horizontal axis the time instants (iterations) n. Note that both curves level out at the same error floor (same misadjustment). However, the convergence rate for the case of the white noise sequence is significantly higher.



- When comparing convergence performance of different algorithms, either all algorithms should converge to the same error floor and compare the respective convergence rates, or all algorithms should have the same convergence rate and compare respective error floors.

The LMS: Some Simulation Examples

- In this example, the dependence of the LMS on the choice of the step-size is demonstrated. The unknown parameters, $\boldsymbol{\theta} \in \mathbb{R}^{10}$, as well as the data were exactly the same with the white noise case of the previous example. The LMS was run using the generated samples, with two different step-sizes, namely, $\mu = 0.01$ and $\mu = 0.0075$. The obtained error curves are shown in the figure below. Observe that the larger the step-size, the faster the convergence becomes albeit at the expense of a higher error floor (misadjustment), in accordance to the theory, which has already been presented.



The Affine Projection Algorithm

- A major drawback of the basic LMS scheme is its fairly slow convergence speed. In an attempt to improve upon it, a number of variants have been proposed over the years. The Affine Projection Algorithm (APA) belongs to the so-called data-reusing family, where, at each time instant, past data are reused. Such a rationale helps the algorithm to "learn faster" and improve the convergence speed. Besides the increased complexity, the faster convergence speed is achieved at the expense of a misadjustment level that increases with the values of q.
- Let the currently available estimate be $\boldsymbol{\theta}_{n-1}$. According to APA, the updated estimate, $\boldsymbol{\theta}$, must satisfy the following constraints,

$$\boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \ldots, q-1.$$

  In other words, we force the parameter vector, $\boldsymbol{\theta}$, to provide at its output the desired response samples, for the q most recent time instants, where q is a user-defined parameter.
- At the same time, APA requires $\boldsymbol{\theta}$ to be as close as possible, in the Euclidean norm sense, to the current estimate, $\boldsymbol{\theta}_{n-1}$. Thus, APA, at each time instant, solves the following constrained optimization task,

$$\begin{aligned} \boldsymbol{\theta}_n &= \arg\min_{\boldsymbol{\theta}} ||\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}||^2 \\ \text{s.t.} \quad & \boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \; i = 0, 1, \ldots, q-1. \end{aligned}$$

The Affine Projection Algorithm

- A major drawback of the basic LMS scheme is its fairly slow convergence speed. In an attempt to improve upon it, a number of variants have been proposed over the years. The Affine Projection Algorithm (APA) belongs to the so-called data-reusing family, where, at each time instant, past data are reused. Such a rationale helps the algorithm to "learn faster" and improve the convergence speed. Besides the increased complexity, the faster convergence speed is achieved at the expense of a misadjustment level that increases with the values of q.
- Let the currently available estimate be $\boldsymbol{\theta}_{n-1}$. According to APA, the updated estimate, $\boldsymbol{\theta}$, must satisfy the following constraints,

$$\boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \ldots, q-1.$$

  In other words, we force the parameter vector, $\boldsymbol{\theta}$, to provide at its output the desired response samples, for the q most recent time instants, where q is a user-defined parameter.
- At the same time, APA requires $\boldsymbol{\theta}$ to be as close as possible, in the Euclidean norm sense, to the current estimate, $\boldsymbol{\theta}_{n-1}$. Thus, APA, at each time instant, solves the following constrained optimization task,

$$\begin{aligned} \boldsymbol{\theta}_n &= \arg\min_{\boldsymbol{\theta}} ||\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}||^2 \\ \text{s.t.} \quad & \boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \; i = 0, 1, \ldots, q-1. \end{aligned}$$

The Affine Projection Algorithm

- A major drawback of the basic LMS scheme is its fairly slow convergence speed. In an attempt to improve upon it, a number of variants have been proposed over the years. The Affine Projection Algorithm (APA) belongs to the so-called data-reusing family, where, at each time instant, past data are reused. Such a rationale helps the algorithm to "learn faster" and improve the convergence speed. Besides the increased complexity, the faster convergence speed is achieved at the expense of a misadjustment level that increases with the values of q.
- Let the currently available estimate be $\boldsymbol{\theta}_{n-1}$. According to APA, the updated estimate, $\boldsymbol{\theta}$, must satisfy the following constraints,

$$\boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \ldots, q-1.$$

  In other words, we force the parameter vector, $\boldsymbol{\theta}$, to provide at its output the desired response samples, for the q most recent time instants, where q is a user-defined parameter.
- At the same time, APA requires $\boldsymbol{\theta}$ to be as close as possible, in the Euclidean norm sense, to the current estimate, $\boldsymbol{\theta}_{n-1}$. Thus, APA, at each time instant, solves the following constrained optimization task,

$$\begin{aligned} \boldsymbol{\theta}_n \quad &= \quad \arg\min_{\boldsymbol{\theta}} ||\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}||^2 \\ \text{s.t.} \quad &\quad \boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \ i = 0, 1, \ldots, q-1. \end{aligned}$$

The Affine Projection Algorithm

- Let us define the $q \times l$ matrix

$$X_n = \begin{bmatrix} \boldsymbol{x}_n^T \\ \vdots \\ \boldsymbol{x}_{n-q+1}^T \end{bmatrix},$$

then the set of constraints can be compactly written as,

$$X_n \boldsymbol{\theta} = \boldsymbol{y}_n, \quad \boldsymbol{y}_n = [y_n \dots y_{n-q+1}]^T.$$

- Using Lagrange multipliers to solve the previous constrained optimization task, provided that $X_n X_n^T$ is invertible, we obtain,

$$\begin{aligned} \boldsymbol{\theta}_n &= \boldsymbol{\theta}_{n-1} + X_n^T \left(X_n X_n^T\right)^{-1} \boldsymbol{e}_n, \\ \boldsymbol{e}_n &= \boldsymbol{y}_n - X_n \boldsymbol{\theta}_{n-1}. \end{aligned}$$

- Usually, a step-size is also used to control the update. The resulting scheme is summarized in the next slide.

The Affine Projection Algorithm

- Let us define the $q \times l$ matrix

$$X_n = \begin{bmatrix} \boldsymbol{x}_n^T \\ \vdots \\ \boldsymbol{x}_{n-q+1}^T \end{bmatrix},$$

then the set of constraints can be compactly written as,

$$X_n \boldsymbol{\theta} = \boldsymbol{y}_n, \quad \boldsymbol{y}_n = [y_n \ldots y_{n-q+1}]^T.$$

- Using Lagrange multipliers to solve the previous constrained optimization task, provided that $X_n X_n^T$ is invertible, we obtain,

$$\begin{aligned} \boldsymbol{\theta}_n &= \boldsymbol{\theta}_{n-1} + X_n^T \left(X_n X_n^T\right)^{-1} \boldsymbol{e}_n, \\ \boldsymbol{e}_n &= \boldsymbol{y}_n - X_n \boldsymbol{\theta}_{n-1}. \end{aligned}$$

- Usually, a step-size is also used to control the update. The resulting scheme is summarized in the next slide.

The Affine Projection Algorithm

- Let us define the $q \times l$ matrix

$$X_n = \begin{bmatrix} \boldsymbol{x}_n^T \\ \vdots \\ \boldsymbol{x}_{n-q+1}^T \end{bmatrix},$$

then the set of constraints can be compactly written as,

$$X_n \boldsymbol{\theta} = \boldsymbol{y}_n, \quad \boldsymbol{y}_n = [y_n \dots y_{n-q+1}]^T.$$

- Using Lagrange multipliers to solve the previous constrained optimization task, provided that $X_n X_n^T$ is invertible, we obtain,

$$\begin{aligned} \boldsymbol{\theta}_n &= \boldsymbol{\theta}_{n-1} + X_n^T \left(X_n X_n^T\right)^{-1} \boldsymbol{e}_n, \\ \boldsymbol{e}_n &= \boldsymbol{y}_n - X_n \boldsymbol{\theta}_{n-1}. \end{aligned}$$

- Usually, a step-size is also used to control the update. The resulting scheme is summarized in the next slide.

- **The Affine Projection Algorithm**
  - Initialize
    - $\boldsymbol{x}_{-1} = \ldots = \boldsymbol{x}_{-q+1} = \mathbf{0}$, $y_{-1} \ldots y_{-q+1} = 0$
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$ (or any other value).
    - Choose $0 < \mu < 2$ and δ to be small.
  - **For** $n = 0, 1, \ldots,$ **Do**
    - $\boldsymbol{e}_n = \boldsymbol{y}_n - X_n \boldsymbol{\theta}_{n-1}$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu X_n^T \left(\delta I + X_n X_n^T\right)^{-1} \boldsymbol{e}_n$
  - **End For**
- When the input is a time series, the corresponding input vector, denoted as $\boldsymbol{u}_n$, is initialized by setting to zero all required samples with negative time index, $u_{-1}, u_{-2}, \ldots$ Note that in the algorithm a parameter, δ, of a small value has also been used in order to prevent numerical problems in the associated matrix inversion. Also, a step-size μ has been introduced, that controls the size of the update and whose presence will be justified soon. The complexity of APA is increased, compared to that of the LMS, due to the involved matrix inversion and matrix operations, requiring $O(q^3)$ MADs.

- **The Affine Projection Algorithm**
  - Initialize
    - $\boldsymbol{x}_{-1} = \ldots = \boldsymbol{x}_{-q+1} = \mathbf{0}$, $y_{-1} \ldots y_{-q+1} = 0$
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$ (or any other value).
    - Choose $0 < \mu < 2$ and δ to be small.
  - **For** $n = 0, 1, \ldots,$ **Do**
    - $\boldsymbol{e}_n = \boldsymbol{y}_n - X_n \boldsymbol{\theta}_{n-1}$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu X_n^T \left(\delta I + X_n X_n^T\right)^{-1} \boldsymbol{e}_n$
  - **End For**
- When the input is a time series, the corresponding input vector, denoted as $\boldsymbol{u}_n$, is initialized by setting to zero all required samples with negative time index, $u_{-1}, u_{-2}, \ldots$. Note that in the algorithm a parameter, δ, of a small value has also been used in order to prevent numerical problems in the associated matrix inversion. Also, a step-size μ has been introduced, that controls the size of the update and whose presence will be justified soon. The complexity of APA is increased, compared to that of the LMS, due to the involved matrix inversion and matrix operations, requiring $O(q^3)$ MADs.

The Affine Projection Algorithm

- The convergence analysis of the APA is more involved than that of the LMS. It turns out that provided that $0 < \mu < 2$, stability of the algorithm is guaranteed. The misadjustment is approximately given by,

$$\mathcal{M} \simeq \frac{\mu q \sigma_\eta^2}{2-\mu} \mathbb{E}\left[\frac{1}{||\mathbf{x}_n||^2}\right] \text{trace}\{\Sigma_x\}.$$

  In words, the misadjustment increases as the parameter q increases; that is, as the number of the reused past data increases.

- Geometric Interpretation of APA: Let us recall the optimization task associated with the APA. Each one of the q constraints $(\boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i},\ i = 0, 1, \ldots, q-1)$ defines a hyperplane in the l-dimensional space. Hence, $\boldsymbol{\theta}_n$ is constrained to lie on **all** these hyperplanes; it thus lies in their intersection.

- Provided that $\boldsymbol{x}_{n-i},\ i = 0, \ldots, q-1$, are linearly independent, these hyperplanes share a nonempty intersection, which is an affine set of dimension $l - q$. An affine set is the translation of a linear subspace (i.e., a plane crossing the origin) by a constant vector; that is, it is a plane in a general position.

The Affine Projection Algorithm

- The convergence analysis of the APA is more involved than that of the LMS. It turns out that provided that $0 < \mu < 2$, stability of the algorithm is guaranteed. The misadjustment is approximately given by,

$$\mathcal{M} \simeq \frac{\mu q \sigma_\eta^2}{2-\mu} \mathbb{E}\left[\frac{1}{||\mathbf{x}_n||^2}\right] \text{trace}\{\Sigma_x\}.$$

  In words, the misadjustment increases as the parameter q increases; that is, as the number of the reused past data increases.

- Geometric Interpretation of APA: Let us recall the optimization task associated with the APA. Each one of the q constraints $(\boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i},\ i = 0, 1, \ldots, q-1)$ defines a hyperplane in the l-dimensional space. Hence, $\boldsymbol{\theta}_n$ is constrained to lie on **all** these hyperplanes; it thus lies in their intersection.

- Provided that $x_{n-i},\ i = 0, \ldots, q-1$, are linearly independent, these hyperplanes share a nonempty intersection, which is an affine set of dimension $l - q$. An affine set is the translation of a linear subspace (i.e., a plane crossing the origin) by a constant vector; that is, it is a plane in a general position.

The Affine Projection Algorithm

- The convergence analysis of the APA is more involved than that of the LMS. It turns out that provided that $0 < \mu < 2$, stability of the algorithm is guaranteed. The misadjustment is approximately given by,

$$\mathcal{M} \simeq \frac{\mu q \sigma_\eta^2}{2-\mu} \mathbb{E}\left[\frac{1}{||\mathbf{x}_n||^2}\right] \text{trace}\{\Sigma_x\}.$$

  In words, the misadjustment increases as the parameter q increases; that is, as the number of the reused past data increases.

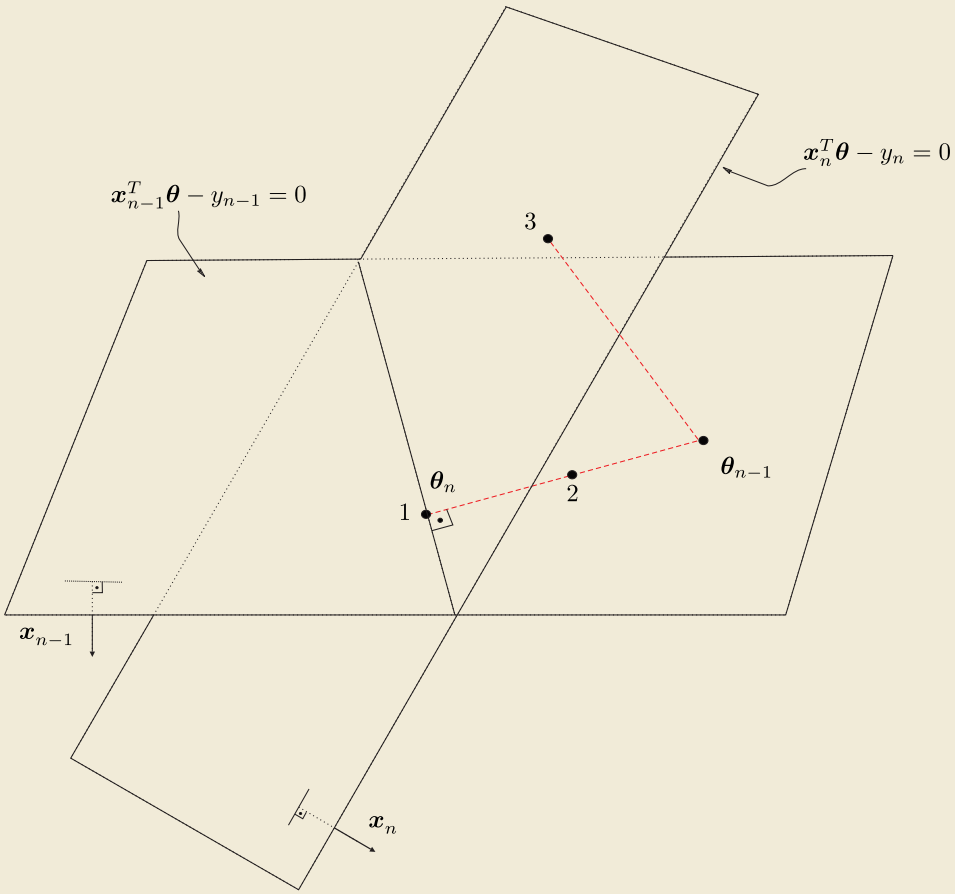
- Geometric Interpretation of APA: Let us recall the optimization task associated with the APA. Each one of the q constraints $(\boldsymbol{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i},\ i = 0, 1, \ldots, q-1)$ defines a hyperplane in the l-dimensional space. Hence, $\boldsymbol{\theta}_n$ is constrained to lie on **all** these hyperplanes; it thus lies in their intersection.

- Provided that $\boldsymbol{x}_{n-i},\ i = 0, \ldots, q-1$, are linearly independent, these hyperplanes share a nonempty intersection, which is an affine set of dimension $l - q$. An affine set is the translation of a linear subspace (i.e., a plane crossing the origin) by a constant vector; that is, it is a plane in a general position.

The Affine Projection Algorithm

- Thus, $\boldsymbol{\theta}_n$ can lie anywhere in this affine set. From the infinite many points lying in this set, APA selects the one which lies closest, in the Euclidean sense, to $\boldsymbol{\theta}_{n-1}$. In other words, $\boldsymbol{\theta}_n$ is the projection of $\boldsymbol{\theta}_{n-1}$ on the affine set defined by the intersection of the q hyperplanes.

- Thus, $\boldsymbol{\theta}_n$ can lie anywhere in this affine set. From the infinite many points lying in this set, APA selects the one which lies closest, in the Euclidean sense, to $\boldsymbol{\theta}_{n-1}$. In other words, $\boldsymbol{\theta}_n$ is the projection of $\boldsymbol{\theta}_{n-1}$ on the affine set defined by the intersection of the q hyperplanes. The figure below illustrates the geometry for the case of $q = 2$; this special case of APA is also known as the binormalized data reusing LMS.



The geometry associated with the APA algorithm, for $q = 2$ and $l = 3$. The intersection of the two hyperplanes is a straight line (affine set of dimension $3 - 2 = 1$). $\boldsymbol{\theta}_n$ is the projection of $\boldsymbol{\theta}_{n-1}$ on this line (point 1) for $\mu = 1$ and $\delta = 0$. Point 2 corresponds to the case $\mu < 1$. Point 3 is the projection of $\boldsymbol{\theta}_n$ on the hyperplane defined by $(y_n, \boldsymbol{x}_n)$. This is the case for $q = 1$. The latter case corresponds to the normalized LMS.

The Normalized LMS

- The normalized LMS a special case of the APA and it corresponds to $q = 1$. We treat it separately due to its popularity and it is summarized below.
- **The Normalized LMS**
  - Initialization
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$, or any other value.
    - Choose $0 < \mu < 2$, and δ a small value.
  - **For** $n = 0, 1, 2, \ldots,$ **Do**
    - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu}{\delta + \boldsymbol{x}_n^T \boldsymbol{x}_n} \boldsymbol{x}_n e_n$
  - **End For**
- The complexity of the normalized LMS, is $3l$ MADs. Stability of the normalized LMS is guaranteed if $0 < \mu < 2$. One can look at the normalized LMS as an LMS, whose step-size is left to vary with the iterations, i.e.,

$$\mu_n = \frac{\mu}{\delta + \boldsymbol{x}_n^T \boldsymbol{x}_n},$$

which turns out to have a beneficial effect on the convergence speed, compared to the LMS.

The Normalized LMS

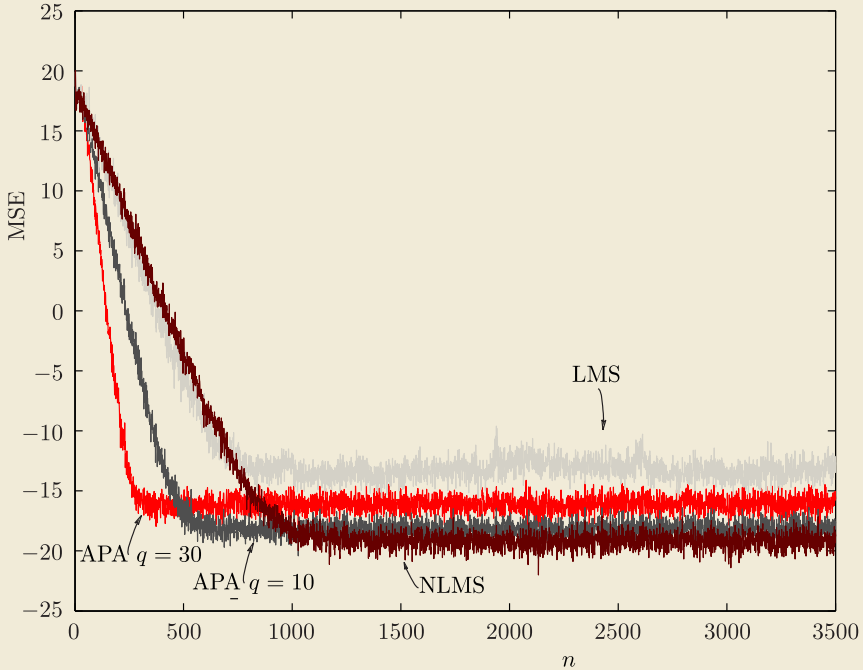
- The normalized LMS a special case of the APA and it corresponds to $q = 1$. We treat it separately due to its popularity and it is summarized below.
- **The Normalized LMS**
  - Initialization
    - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$, or any other value.
    - Choose $0 < \mu < 2$, and δ a small value.
  - **For** $n = 0, 1, 2, \ldots,$ **Do**
    - $e_n = y_n - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_n$
    - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu}{\delta + \boldsymbol{x}_n^T \boldsymbol{x}_n} \boldsymbol{x}_n e_n$
  - **End For**
- The complexity of the normalized LMS, is $3l$ MADs. Stability of the normalized LMS is guaranteed if $0 < \mu < 2$. One can look at the normalized LMS as an LMS, whose step-size is left to vary with the iterations, i.e.,

$$\mu_n = \frac{\mu}{\delta + \boldsymbol{x}_n^T \boldsymbol{x}_n},$$

which turns out to have a beneficial effect on the convergence speed, compared to the LMS.

Simulation Examples

- The experimental set up was that of the regression model, as in the example before, with the only exception that the unknown parameter vector was of higher dimension, $\boldsymbol{\theta} \in \mathbb{R}^{60}$, so that the differences in the performance of the algorithms to be more clear. The goal is to compare the LMS, the normalized LMS (NLMS) and the affine projection algorithm (APA). The figure below shows the obtained error curves.

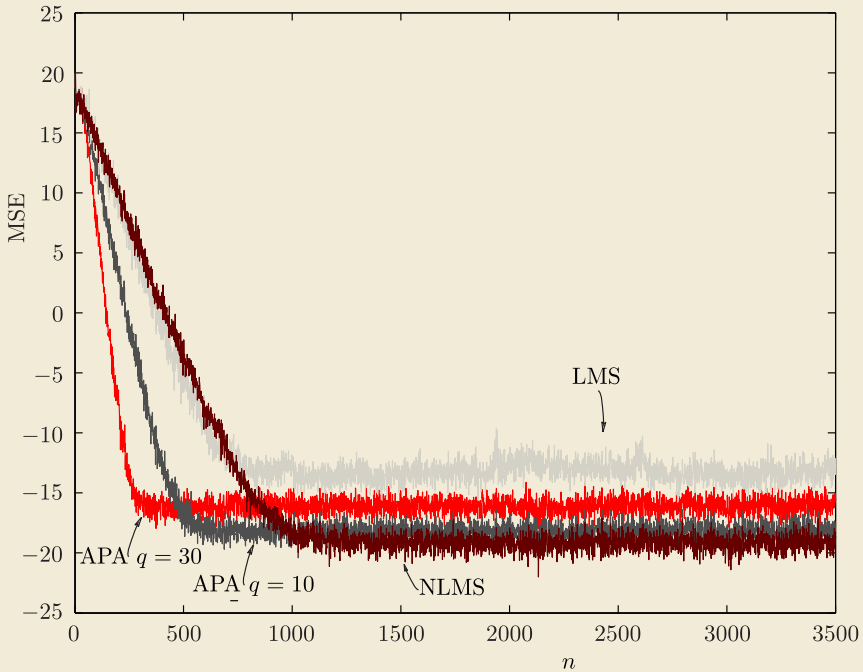


The step-size of the LMS was chosen equal to $\mu = 0.025$ and for the NLMS $\mu = 0.35$ and $\delta = 0.001$, so that both algorithms to have similar convergence rate. The step-size for the APA was chosen equal to $\mu = 0.1$, so that for $q = 10$ to settle at the same error floor as that of the NLMS. For the APA, we also chose $\delta = 0.001$.

- Observe the faster convergence obtained by the NLMS compared to the LMS and the improved performance obtained by the APA for $q = 10$. Increasing, the value to $q = 30$, we can see the improved convergence rate which is obtained, however at the expense of higher error floor.

Simulation Examples

- The experimental set up was that of the regression model, as in the example before, with the only exception that the unknown parameter vector was of higher dimension, $\boldsymbol{\theta} \in \mathbb{R}^{60}$, so that the differences in the performance of the algorithms to be more clear. The goal is to compare the LMS, the normalized LMS (NLMS) and the affine projection algorithm (APA). The figure below shows the obtained error curves.



The step-size of the LMS was chosen equal to $\mu = 0.025$ and for the NLMS $\mu = 0.35$ and $\delta = 0.001$, so that both algorithms to have similar convergence rate. The step-size for the APA was chosen equal to $\mu = 0.1$, so that for $q = 10$ to settle at the same error floor as that of the NLMS. For the APA, we also chose $\delta = 0.001$.

- Observe the faster convergence obtained by the NLMS compared to the LMS and the improved performance obtained by the APA for $q = 10$. Increasing, the value to $q = 30$, we can see the improved convergence rate which is obtained, however at the expense of higher error floor.

Relatives of the LMS

- The Sign-Error LMS: The update recursion for this algorithm becomes,
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \text{sgn}[e_n]\boldsymbol{x}_n.$$
If in addition, μ is chosen to be a power of two, then the recursion becomes multiplication free, and l multiplications are only needed for the computation of the error. It turns out that the algorithm minimizes, in the stochastic approximation sense, the following cost function,
$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y} - \boldsymbol{\theta}^T \mathbf{x}|\right],$$
and stability is guaranteed for sufficiently small values of μ.
- The Least Mean Fourth Algorithm (LMF) The scheme minimizes the following cost function,
$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y} - \boldsymbol{\theta}^T \boldsymbol{x}|^4\right],$$
and the corresponding update recursion is given by,
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu |e_n|^2 \boldsymbol{x}_n e_n.$$
- Minimization of the fourth power of the error may lead to an adaptive scheme with better compromise between convergence rate and excess mean-square error than the LMS, if the noise source is sub-Gaussian. In a sub-Gaussian distribution, the tails of the pdf graph are decaying at a faster rate compared to the Gaussian one.

Relatives of the LMS

- The Sign-Error LMS: The update recursion for this algorithm becomes,
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \text{sgn}[e_n]\boldsymbol{x}_n.$$
If in addition, μ is chosen to be a power of two, then the recursion becomes multiplication free, and l multiplications are only needed for the computation of the error. It turns out that the algorithm minimizes, in the stochastic approximation sense, the following cost function,
$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y} - \boldsymbol{\theta}^T \mathbf{x}|\right],$$
and stability is guaranteed for sufficiently small values of μ.
- The Least Mean Fourth Algorithm (LMF) The scheme minimizes the following cost function,
$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y} - \boldsymbol{\theta}^T \boldsymbol{x}|^4\right],$$
and the corresponding update recursion is given by,
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu |e_n|^2 \boldsymbol{x}_n e_n.$$
- Minimization of the fourth power of the error may lead to an adaptive scheme with better compromise between convergence rate and excess mean-square error than the LMS, if the noise source is sub-Gaussian. In a sub-Gaussian distribution, the tails of the pdf graph are decaying at a faster rate compared to the Gaussian one.

Relatives of the LMS

- The Sign-Error LMS: The update recursion for this algorithm becomes,
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \text{sgn}[e_n]\boldsymbol{x}_n.$$
If in addition, μ is chosen to be a power of two, then the recursion becomes multiplication free, and l multiplications are only needed for the computation of the error. It turns out that the algorithm minimizes, in the stochastic approximation sense, the following cost function,
$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y} - \boldsymbol{\theta}^T \mathbf{x}|\right],$$
and stability is guaranteed for sufficiently small values of μ.
- The Least Mean Fourth Algorithm (LMF) The scheme minimizes the following cost function,
$$J(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y} - \boldsymbol{\theta}^T \boldsymbol{x}|^4\right],$$
and the corresponding update recursion is given by,
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu |e_n|^2 \boldsymbol{x}_n e_n.$$
- Minimization of the fourth power of the error may lead to an adaptive scheme with better compromise between convergence rate and excess mean-square error than the LMS, if the noise source is sub-Gaussian. In a sub-Gaussian distribution, the tails of the pdf graph are decaying at a faster rate compared to the Gaussian one.

Relatives of the LMS

- Transform-Domain LMS: We have already commented that the convergence speed of the LMS heavily depends on the condition number $\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)$ of the covariance matrix.
- Transform domain techniques exploit the de-correlation properties of certain transforms, such as DFT and DCT, in order to de-correlate the input variables. When the input comprises a stochastic process, we say that such transforms "pre-whiten" the input process.
- Let T be a unitary transform in the complex domain, i.e., $TT^H = T^H T = I$. Furthermore, define

$$\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n,$$

and the diagonal matrix D, such as

$$[D]_{ii} = \mathbb{E}\left[(\hat{x}_n(i))^2\right] = \sigma_i^2, \quad i = 1, 2, \ldots, l,$$

- Then, the transform domain LMS, in the complex domain, becomes:

Relatives of the LMS

- Transform-Domain LMS: We have already commented that the convergence speed of the LMS heavily depends on the condition number $\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)$ of the covariance matrix.
- Transform domain techniques exploit the de-correlation properties of certain transforms, such as DFT and DCT, in order to de-correlate the input variables. When the input comprises a stochastic process, we say that such transforms "pre-whiten" the input process.
- Let T be a unitary transform in the complex domain, i.e., $TT^H = T^H T = I$. Furthermore, define

$$\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n,$$

and the diagonal matrix D, such as

$$[D]_{ii} = \mathbb{E}\left[(\hat{x}_n(i))^2\right] = \sigma_i^2, \quad i = 1, 2, \ldots, l,$$

- Then, the transform domain LMS, in the complex domain, becomes:

Relatives of the LMS

- Transform-Domain LMS: We have already commented that the convergence speed of the LMS heavily depends on the condition number $\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)$ of the covariance matrix.
- Transform domain techniques exploit the de-correlation properties of certain transforms, such as DFT and DCT, in order to de-correlate the input variables. When the input comprises a stochastic process, we say that such transforms "pre-whiten" the input process.
- Let T be a unitary transform in the complex domain, i.e., $TT^H = T^H T = I$. Furthermore, define

$$\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n,$$

and the diagonal matrix D, such as

$$[D]_{ii} = \mathbb{E}\left[(\hat{x}_n(i))^2\right] = \sigma_i^2, \quad i = 1, 2, \ldots, l,$$

- Then, the transform domain LMS, in the complex domain, becomes:

Relatives of the LMS

- Transform-Domain LMS: We have already commented that the convergence speed of the LMS heavily depends on the condition number $\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)$ of the covariance matrix.
- Transform domain techniques exploit the de-correlation properties of certain transforms, such as DFT and DCT, in order to de-correlate the input variables. When the input comprises a stochastic process, we say that such transforms "pre-whiten" the input process.
- Let T be a unitary transform in the complex domain, i.e., $TT^H = T^H T = I$. Furthermore, define

$$\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n,$$

and the diagonal matrix D, such as

$$[D]_{ii} = \mathbb{E}\left[(\hat{x}_n(i))^2\right] = \sigma_i^2, \quad i = 1, 2, \ldots, l,$$

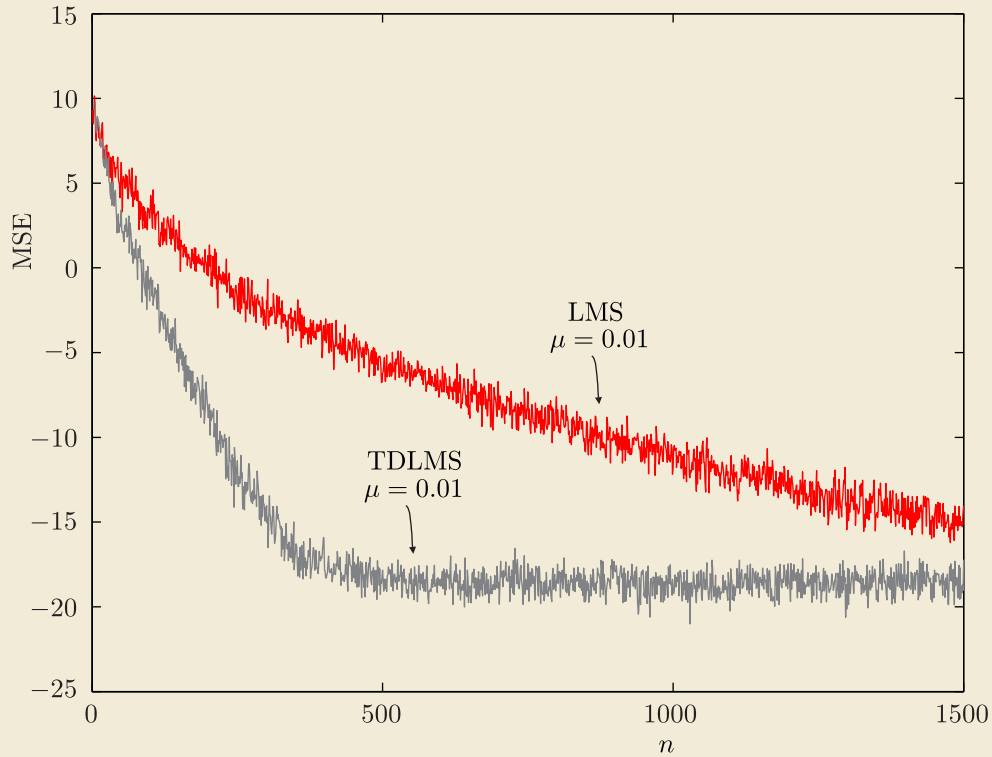
- Then, the transform domain LMS, in the complex domain, becomes:

Relatives of the LMS

- **Transform Domain LMS**
  - Initialization
    - $\hat{\boldsymbol{\theta}}_{-1} = \mathbf{0}$; or any other value.
    - $\sigma_{-1}^2 = \delta,\ i = 1, 2, \ldots, l$; δ a small value.
    - Choose μ, and $0 << \beta < 1$.
  - **For** $n = 0, 1, 2, \ldots,$ **Do**
    - $\hat{\boldsymbol{x}}_n = T^H \boldsymbol{x}_n$
    - $e_n = y_n - \hat{\boldsymbol{\theta}}_{n-1}^H \hat{\boldsymbol{x}}_n$
    - $\hat{\boldsymbol{\theta}}_n = \hat{\boldsymbol{\theta}}_{n-1} + \mu D^{-1} \hat{\boldsymbol{x}}_n e_n^*$
    - **For** $i = 1, 2, \ldots, l,$ **DO**
      - $\sigma_i^2(n) = \beta \sigma_i^2(n-1) + (1-\beta)|\hat{x}_n(i)|^2$
    - **End For**
    - $D = \text{diag}\{\sigma_i^2(n)\}$
  - **End For**

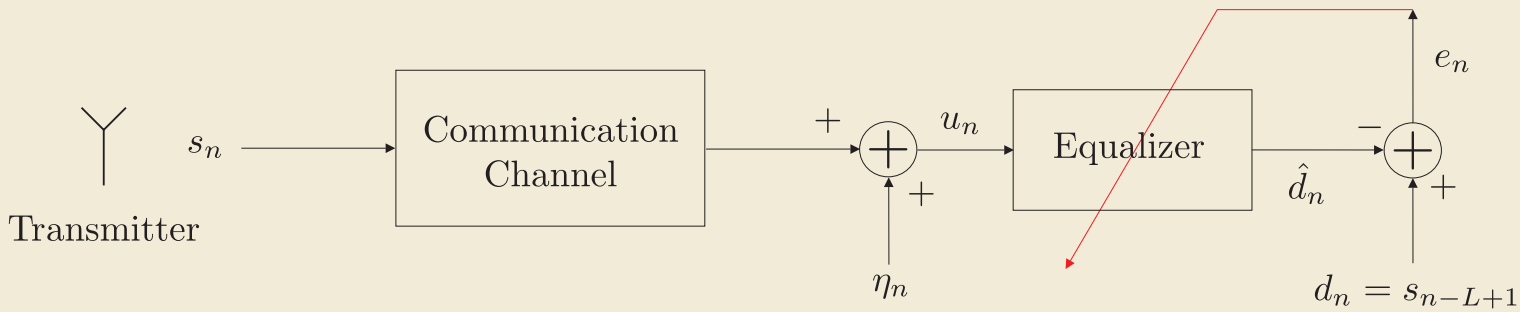
Simulation Examples

- In this example, the stage of the experimental set up is the same as the one previously considered for the LMS, when the input was excited by a first order autoregressive AR(1) sequence. The goal is to compare the LMS and the transform domain LMS. The figure below shows the obtained error curves, employing the DCT transform. The step-size was the same as the one used before, i.e., $\mu = 0.01$. Observe the significantly faster convergence achieved by the transform domain LMS, due to its (approximate) whitening effect on the input.



Adaptive Decision Feedback Equalization

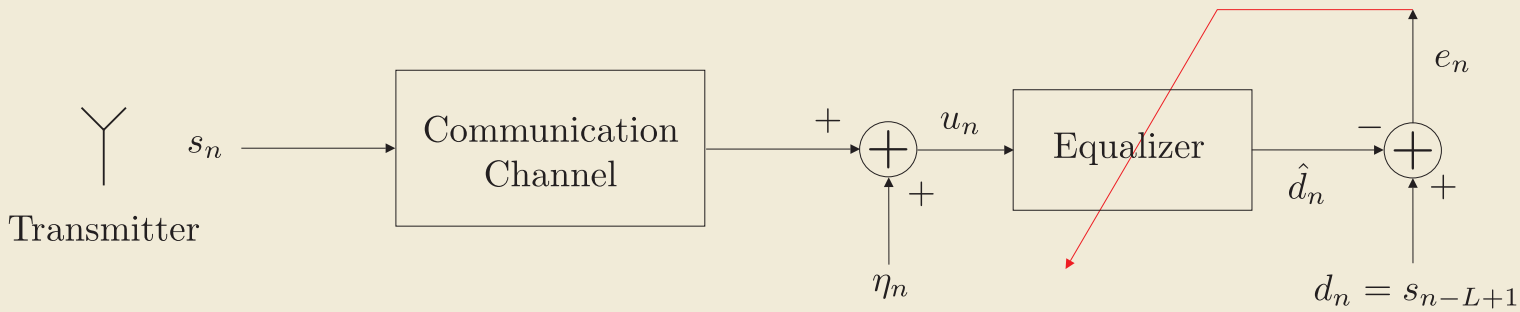
- The task of channel equalization is introduced in Chapter 4 and it is illustrated in the figure below. The input to the equalizer is a stochastic process (random signal), u_n. Note that upon receiving the noisy and distorted by the (communications) channel sample, u_n, one has to obtain an estimate of the originally transmitted information sequence, s_n, delayed by L time lags, which accounts for the various delays imposed by the overall transmission system involved.



- Thus, at time n, the equalizer decides for $\hat{s}_{n-L+1}$. Ideally, if one knew the true values of the initially transmitted information sequence up to and including time instant $n-L$, i.e., $s_{n-L}, s_{n-L-1}, \ldots$, it could only be beneficial to use this information, together with the received sequence, u_n, in order to recover an estimate of $\hat{s}_{n-L+1}$.

Adaptive Decision Feedback Equalization

- The task of channel equalization is introduced in Chapter 4 and it is illustrated in the figure below. The input to the equalizer is a stochastic process (random signal), u_n. Note that upon receiving the noisy and distorted by the (communications) channel sample, u_n, one has to obtain an estimate of the originally transmitted information sequence, s_n, delayed by L time lags, which accounts for the various delays imposed by the overall transmission system involved.



- Thus, at time n, the equalizer decides for $\hat{s}_{n-L+1}$. Ideally, if one knew the true values of the initially transmitted information sequence up to and including time instant $n-L$, i.e., $s_{n-L}, s_{n-L-1}, \ldots$, it could only be beneficial to use this information, together with the received sequence, u_n, in order to recover an estimate of $\hat{s}_{n-L+1}$.

Adaptive Decision Feedback Equalization

- This idea is explored in the Decision Feedback Equalizer. The equalizer's output is now written as

$$\hat{d}_n = \sum_{i=0}^{L-1} w_i^f u_{n-i} + \sum_{i=0}^{l-1} w_i^b s_{n-L-i} = \boldsymbol{w}^T \boldsymbol{u}_{e,n}$$

where,

$$\boldsymbol{w} := \begin{bmatrix} \boldsymbol{w}^f \\ \boldsymbol{w}^b \end{bmatrix} \in \mathbb{R}^{L+l}, \quad \boldsymbol{u}_{e,n} := \begin{bmatrix} \boldsymbol{u}_n \\ \boldsymbol{s}_n \end{bmatrix} \in \mathbb{R}^{L+l},$$

and the desired response process is $d_n = s_{n-L+1}$. In practice, after the initial training period, the information samples, s_{n-L-i} are replaced by their computed estimates, $\hat{s}_{n-L-i},\ i = 0, 1, \ldots, l-1$, which are available from decisions taken in previous time instants. It is said that the equalizer operates in the decision directed mode.

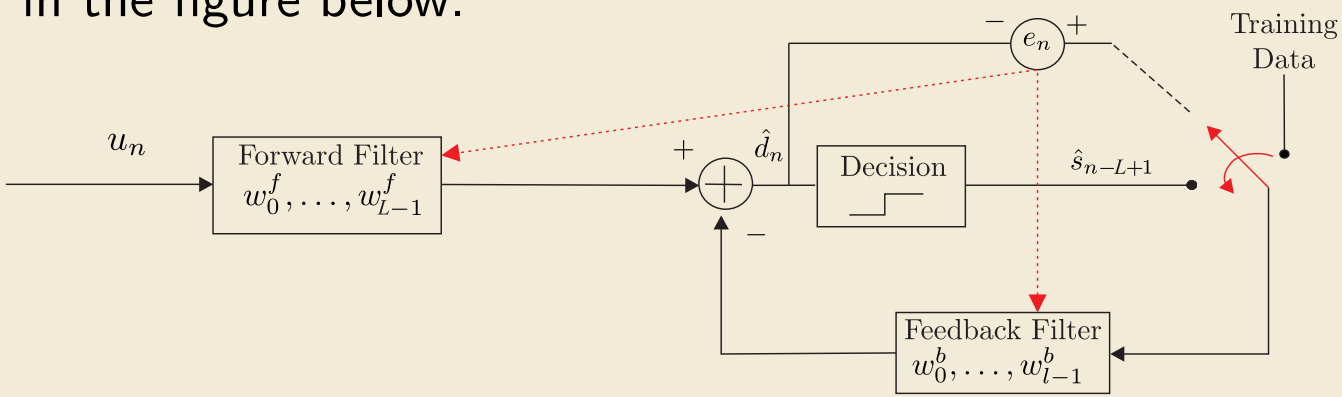
- This idea is explored in the Decision Feedback Equalizer. The equalizer's output is now written as

$$\hat{d}_n = \sum_{i=0}^{L-1} w_i^f u_{n-i} + \sum_{i=0}^{l-1} w_i^b s_{n-L-i} = \boldsymbol{w}^T \boldsymbol{u}_{e,n}$$

where,

$$\boldsymbol{w} := \begin{bmatrix} \boldsymbol{w}^f \\ \boldsymbol{w}^b \end{bmatrix} \in \mathbb{R}^{L+l}, \quad \boldsymbol{u}_{e,n} := \begin{bmatrix} \boldsymbol{u}_n \\ \boldsymbol{s}_n \end{bmatrix} \in \mathbb{R}^{L+l},$$

and the desired response process is $d_n = s_{n-L+1}$. In practice, after the initial training period, the information samples, s_{n-L-i} are replaced by their computed estimates, $\hat{s}_{n-L-i},\ i = 0, 1, \ldots, l-1$, which are available from decisions taken in previous time instants. It is said that the equalizer operates in the decision directed mode. The structure is shown in the figure below.



Adaptive Decision Feedback Equalization

- Note that during the training period, the parameter vector, $\boldsymbol{w}$, is trained so that to minimize the power of the error,

$$e_n = d_n - \hat{d}_n = s_{n-L+1} - \hat{d}_n.$$

  Once all the available training samples have been used, training carries on using the estimates $\hat{s}_{n-L+1}$. For example, for a binary information sequence $s_n \in \{1, -1\}$, the decision concerning the estimate at time n, is obtained by passing $\hat{d}_n$ through a threshold device and $\hat{s}_{n-L+1}$ is obtained.

- Note that DFE is one of the early examples of semisupervised learning, where training data is not enough, and then the estimates are used for training. In this way, assuming that at the end of the training phase, $\hat{s}_{n-L+1} = s_{n-L+1}$, and assuming that time variations are slow so that to guarantee that $\hat{d}_n \simeq d_n$, then we expect, with good enough probability, that $\hat{s}_{n-L+1}$ will remain equal to s_{n-L+1}, so that the equalizer can track the changes.

- Anyone of the online schemes, treated so far, can be used in a DFE scenario by replacing in the input vector, $\boldsymbol{u}_e$, the term $\boldsymbol{s}_n$ by $\hat{s}_n$, when operating in the decision directed mode.

Adaptive Decision Feedback Equalization

- Note that during the training period, the parameter vector, $\boldsymbol{w}$, is trained so that to minimize the power of the error,

$$e_n = d_n - \hat{d}_n = s_{n-L+1} - \hat{d}_n.$$

  Once all the available training samples have been used, training carries on using the estimates $\hat{s}_{n-L+1}$. For example, for a binary information sequence $s_n \in \{1, -1\}$, the decision concerning the estimate at time n, is obtained by passing $\hat{d}_n$ through a threshold device and $\hat{s}_{n-L+1}$ is obtained.

- Note that DFE is one of the early examples of semisupervised learning, where training data is not enough, and then the estimates are used for training. In this way, assuming that at the end of the training phase, $\hat{s}_{n-L+1} = s_{n-L+1}$, and assuming that time variations are slow so that to guarantee that $\hat{d}_n \simeq d_n$, then we expect, with good enough probability, that $\hat{s}_{n-L+1}$ will remain equal to s_{n-L+1}, so that the equalizer can track the changes.

- Anyone of the online schemes, treated so far, can be used in a DFE scenario by replacing in the input vector, $\boldsymbol{u}_e$, the term $\boldsymbol{s}_n$ by $\hat{s}_n$, when operating in the decision directed mode.

Adaptive Decision Feedback Equalization

- Note that during the training period, the parameter vector, $\boldsymbol{w}$, is trained so that to minimize the power of the error,

$$e_n = d_n - \hat{d}_n = s_{n-L+1} - \hat{d}_n.$$

  Once all the available training samples have been used, training carries on using the estimates $\hat{s}_{n-L+1}$. For example, for a binary information sequence $s_n \in \{1, -1\}$, the decision concerning the estimate at time n, is obtained by passing $\hat{d}_n$ through a threshold device and $\hat{s}_{n-L+1}$ is obtained.

- Note that DFE is one of the early examples of semisupervised learning, where training data is not enough, and then the estimates are used for training. In this way, assuming that at the end of the training phase, $\hat{s}_{n-L+1} = s_{n-L+1}$, and assuming that time variations are slow so that to guarantee that $\hat{d}_n \simeq d_n$, then we expect, with good enough probability, that $\hat{s}_{n-L+1}$ will remain equal to s_{n-L+1}, so that the equalizer can track the changes.

- Anyone of the online schemes, treated so far, can be used in a DFE scenario by replacing in the input vector, $\boldsymbol{u}_e$, the term $\boldsymbol{s}_n$ by $\hat{\boldsymbol{s}}_n$, when operating in the decision directed mode.

Adaptive Decision Feedback Equalization: An Example

- Let us consider a communication system where the input information sequence comprises a stream of randomly generated symbols $s_n = \pm 1$, with equal probability. This sequence is sent to a channel with impulse response,

$$\boldsymbol{h} = [0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0.21, 0.03, 0.07]^T.$$

  The output of the channel is contaminated by white Gaussian noise at the 11dB level. A DFE is used with $L = 21$ and $l = 10$. The DFE was trained with 250 symbols; then, it was switched on to the decision mode and it was run for 10000 iterations. At each iteration, the decision $(\text{sgn}(\hat{d}_n))$ was compared with the true transmitted symbol s_{n-L+1}.

- If $T[\cdot]$ denotes the thresholding operation, the LMS recursion for the linear DFE becomes,

$$\begin{aligned}
\hat{d}_n &= \boldsymbol{w}_{n-1}^T \boldsymbol{u}_{e,n} \\
d_n &= s_{n-L+1}; \quad \text{in the training mode, or} \\
d_n &= T\left[\hat{d}_n\right]; \quad \text{in the decision directed mode,} \\
e_n &= d_n - \boldsymbol{w}_{n-1}^T \boldsymbol{u}_{e,n} \\
\boldsymbol{w}_n &= \boldsymbol{w}_{n-1} + \mu \boldsymbol{u}_{e,n} e_n.
\end{aligned}$$

Adaptive Decision Feedback Equalization: An Example

- Let us consider a communication system where the input information sequence comprises a stream of randomly generated symbols $s_n = \pm 1$, with equal probability. This sequence is sent to a channel with impulse response,

$$\boldsymbol{h} = [0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0.21, 0.03, 0.07]^T.$$

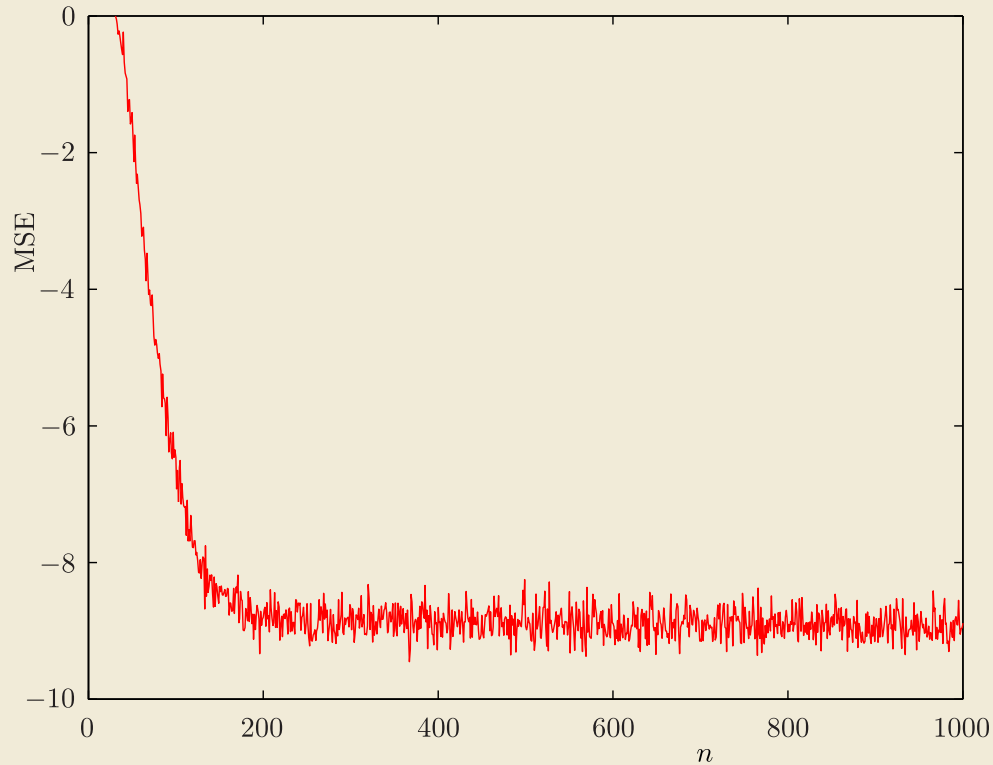
  The output of the channel is contaminated by white Gaussian noise at the 11dB level. A DFE is used with $L = 21$ and $l = 10$. The DFE was trained with 250 symbols; then, it was switched on to the decision mode and it was run for 10000 iterations. At each iteration, the decision $(\text{sgn}(\hat{d}_n))$ was compared with the true transmitted symbol s_{n-L+1}.

- If $T[\cdot]$ denotes the thresholding operation, the LMS recursion for the linear DFE becomes,

$$\begin{aligned}
\hat{d}_n &= \boldsymbol{w}_{n-1}^T \boldsymbol{u}_{e,n} \\
d_n &= s_{n-L+1}; \quad \text{in the training mode, or} \\
d_n &= T\left[\hat{d}_n\right]; \quad \text{in the decision directed mode,} \\
e_n &= d_n - \boldsymbol{w}_{n-1}^T \boldsymbol{u}_{e,n} \\
\boldsymbol{w}_n &= \boldsymbol{w}_{n-1} + \mu \boldsymbol{u}_{e,n} e_n.
\end{aligned}$$

Adaptive Decision Feedback Equalization: An Example

- The figure below shows the MSE curve as a function of iterations. For the LMS, we used $\mu = 0.025$.

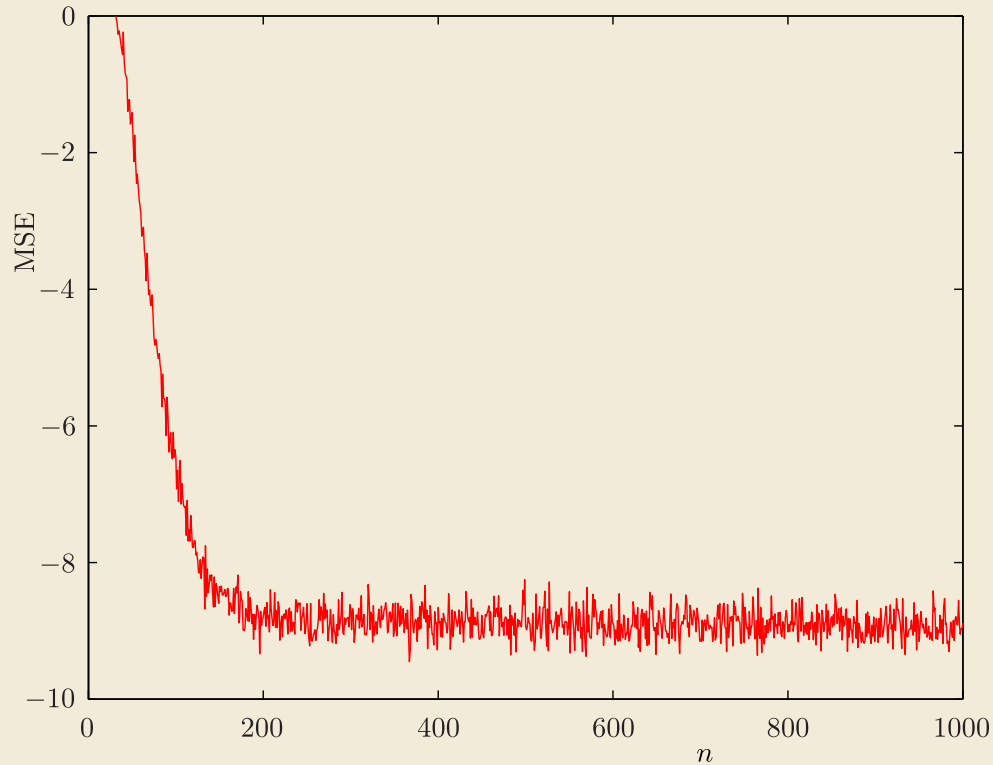


The MSE in dBs for the DFE. After the time instant $n = 250$, the LMS is trained with the decisions $\hat{s}_{n-L+1}$.

- The error rate (total number of errors over the corresponding number of symbols) was approximately equal 1%.

Adaptive Decision Feedback Equalization: An Example

- The figure below shows the MSE curve as a function of iterations. For the LMS, we used $\mu = 0.025$.



The MSE in dBs for the DFE. After the time instant $n = 250$, the LMS is trained with the decisions $\hat{s}_{n-L+1}$.

- The error rate (total number of errors over the corresponding number of symbols) was approximately equal 1%.

Distributed Learning

- There is an increasing number of applications where data are received/reside in different sensors/data bases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal; that is, to perform a common inference task. We refer to such tasks as distributed or decentralized learning.
- At the heart of this problem lies the concept of cooperation, which is another name for the process of exchanging learning experience/ information in order to reach a common goal/decision.
- Distributed learning is common in many biological systems, where no individual/agent is in charge yet the group exhibits a high degree of intelligence (we humans refer to it as instinct). Look at the way birds fly in formation and bees swarm in a new hive.
- Wireless Sensor Networks (WSN) is another typical example in engineering applications. Each sensor node is equipped with an onboard processor, in order to perform locally some simple processing and transmit the required and partially processed data. Sensors/nodes are characterized by low processing, memory and communication capabilities due to low energy and bandwidth constraints.

Distributed Learning

- There is an increasing number of applications where data are received/reside in different sensors/data bases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal; that is, to perform a common inference task. We refer to such tasks as distributed or decentralized learning.
- At the heart of this problem lies the concept of cooperation, which is another name for the process of exchanging learning experience/ information in order to reach a common goal/decision.
- Distributed learning is common in many biological systems, where no individual/agent is in charge yet the group exhibits a high degree of intelligence (we humans refer to it as instinct). Look at the way birds fly in formation and bees swarm in a new hive.
- Wireless Sensor Networks (WSN) is another typical example in engineering applications. Each sensor node is equipped with an onboard processor, in order to perform locally some simple processing and transmit the required and partially processed data. Sensors/nodes are characterized by low processing, memory and communication capabilities due to low energy and bandwidth constraints.

Distributed Learning

- There is an increasing number of applications where data are received/reside in different sensors/data bases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal; that is, to perform a common inference task. We refer to such tasks as distributed or decentralized learning.
- At the heart of this problem lies the concept of cooperation, which is another name for the process of exchanging learning experience/ information in order to reach a common goal/decision.
- Distributed learning is common in many biological systems, where no individual/agent is in charge yet the group exhibits a high degree of intelligence (we humans refer to it as instinct). Look at the way birds fly in formation and bees swarm in a new hive.
- Wireless Sensor Networks (WSN) is another typical example in engineering applications. Each sensor node is equipped with an onboard processor, in order to perform locally some simple processing and transmit the required and partially processed data. Sensors/nodes are characterized by low processing, memory and communication capabilities due to low energy and bandwidth constraints.

Distributed Learning

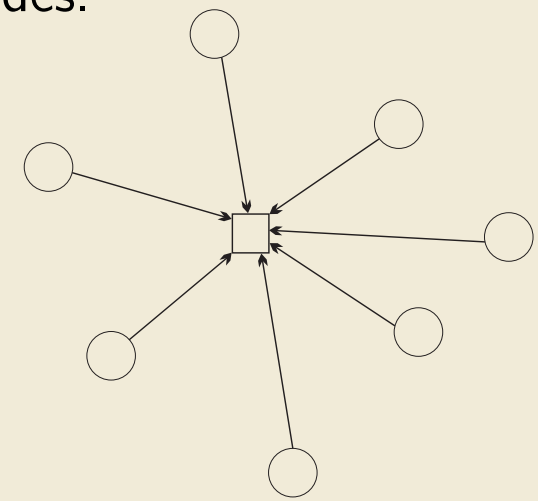
- There is an increasing number of applications where data are received/reside in different sensors/data bases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal; that is, to perform a common inference task. We refer to such tasks as distributed or decentralized learning.
- At the heart of this problem lies the concept of cooperation, which is another name for the process of exchanging learning experience/ information in order to reach a common goal/decision.
- Distributed learning is common in many biological systems, where no individual/agent is in charge yet the group exhibits a high degree of intelligence (we humans refer to it as instinct). Look at the way birds fly in formation and bees swarm in a new hive.
- Wireless Sensor Networks (WSN) is another typical example in engineering applications. Each sensor node is equipped with an onboard processor, in order to perform locally some simple processing and transmit the required and partially processed data. Sensors/nodes are characterized by low processing, memory and communication capabilities due to low energy and bandwidth constraints.

Cooperation Strategies

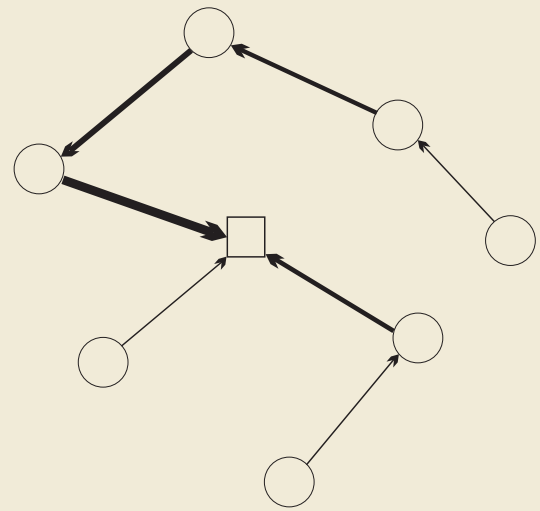
- In distributed learning, each individual agent is represented as a node in a graph. Edges between nodes indicate that the respective agents can exchange information. Undirected edges indicate that information can be exchanged in both directions, while directed edges indicate the allowed direction of information flow.

Cooperation Strategies

- In distributed learning, each individual agent is represented as a node in a graph. Edges between nodes indicate that the respective agents can exchange information. Undirected edges indicate that information can be exchanged in both directions, while directed edges indicate the allowed direction of information flow.
- Centralized Networks: Under this scenario of cooperation, nodes communicate their measurements to a central fusion unit for processing. The obtained estimate can be communicated back to each one of the nodes.



All nodes communicate directly to the fusion center



Some nodes are connected directly to the fusion center. Others, communicate their own data to a neighboring node and so on, till the information reaches the fusion center. The bolder a connection is drawn, the higher the amount of data transmitted via the corresponding link.

Cooperation Strategies

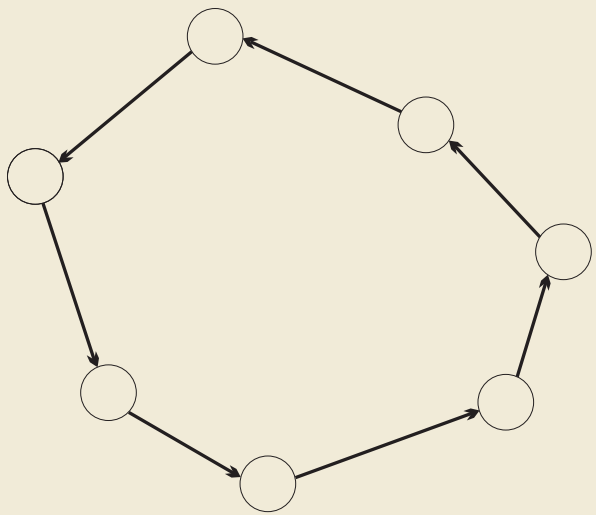
- The major advantage in this cooperation strategy is that the fusion center can compute optimal estimates, since it has access to all the available information. However, the optimality is obtained under a number of drawbacks, such as: demand for increased communication costs and delays, especially for large networks. In addition, when the fusion center breaks down, the whole network collapses. Moreover, in certain applications, privacy issues are involved. To overcome the drawbacks of centralized processing scenario, different distributed processing schemes have been proposed.
- Decentralized Networks: Under this scenario, there is not a central fusion center. Processing is performed locally at each node, employing the locally received measurements and in the sequel each node communicates the locally obtained estimates to its neighbors; that is, to the nodes which is linked with. These links are denoted as edges in the respective graph. There are different decentralized schemes.

Cooperation Strategies

- The major advantage in this cooperation strategy is that the fusion center can compute optimal estimates, since it has access to all the available information. However, the optimality is obtained under a number of drawbacks, such as: demand for increased communication costs and delays, especially for large networks. In addition, when the fusion center breaks down, the whole network collapses. Moreover, in certain applications, privacy issues are involved. To overcome the drawbacks of centralized processing scenario, different distributed processing schemes have been proposed.
- Decentralized Networks: Under this scenario, there is not a central fusion center. Processing is performed locally at each node, employing the locally received measurements and in the sequel each node communicates the locally obtained estimates to its neighbors; that is, to the nodes which is linked with. These links are denoted as edges in the respective graph. There are different decentralized schemes.

Incremental/Ring Networks

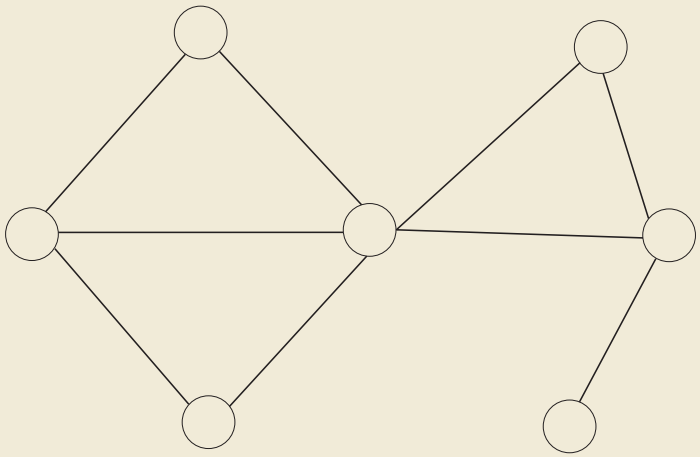
- Ring networks require the existence of a cyclic path following the edges through the network. Starting from a node, such a cycle has to visit every node, at least once, and then return to the first one. Such a topology implements an iterative computational scheme. At each iteration, every node performs its data acquisition and **processing locally** and communicates the required information to its neighbor in the cyclic path. It has been shown that incremental schemes achieve global performance. The main disadvantage of this mode of cooperation is that cycling information around at each iteration is a problem in large networks. The construction and maintenance of a cyclic graph, visiting each node, is an NP–hard task.



In the incremental or ring topology, the information flow follows a cyclic path. Note that the whole network **collapses** if one node is **malfunctioning.**

Ad Hoc Networks

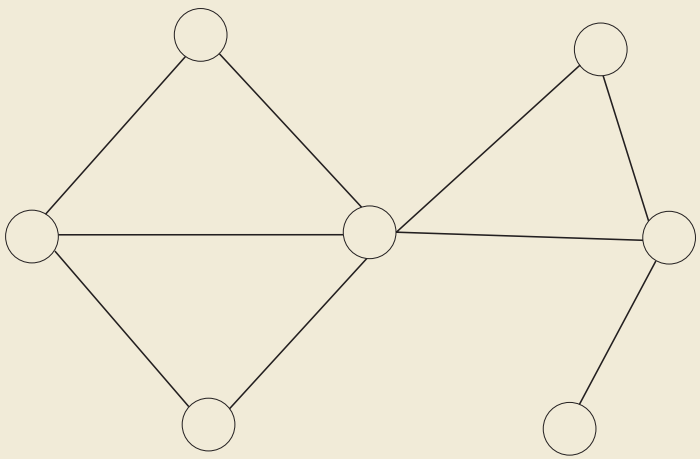
- According to this philosophy of cooperation, nodes perform **locally** data acquisition as well as processing, at each iteration. Each node communicates information to its neighboring nodes with which it shares an edge; in this way, information is diffused across the whole network.



Topology corresponding to a diffusion strategy. Each node communicates information with the nodes with which it shares an edge.

Ad Hoc Networks

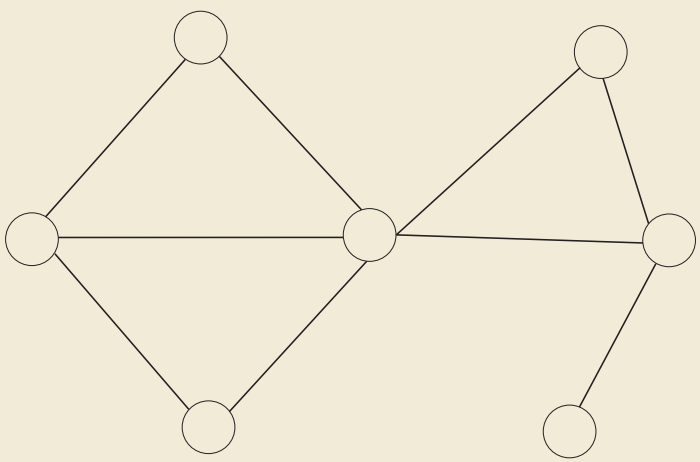
- According to this philosophy of cooperation, nodes perform **locally** data acquisition as well as processing, at each iteration. Each node communicates information to its neighboring nodes with which it shares an edge; in this way, information is diffused across the whole network.
- An advantage of such schemes is that operation is not seized if some nodes are malfunctioning. Also, the topology of the network may not be fixed.



Topology corresponding to a diffusion strategy. Each node communicates information with the nodes with which it shares an edge.

Ad Hoc Networks

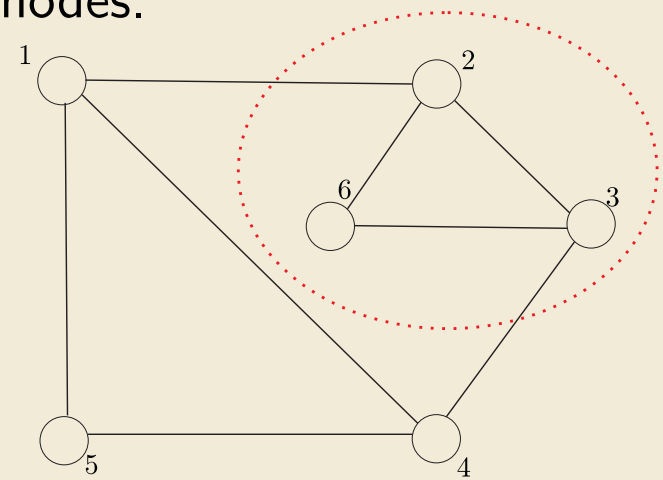
- According to this philosophy of cooperation, nodes perform **locally** data acquisition as well as processing, at each iteration. Each node communicates information to its neighboring nodes with which it shares an edge; in this way, information is diffused across the whole network.
- An advantage of such schemes is that operation is not seized if some nodes are malfunctioning. Also, the topology of the network may not be fixed.
- The price one pays for such "extras" is that the final obtained performance, after convergence, is inferior to those obtained by its incremental counterpart and by the centralized processing. This is natural, since at each iteration every node has access to only limited amount of information.



Topology corresponding to a diffusion strategy. Each node communicates information with the nodes with which it shares an edge.

Diffusion Gradient Descent

- Let us consider a network of K agents/nodes. Each node exchanges information with the nodes in its neighborhood. Given a node, k, in a graph, let $\mathcal{N}_k$ be the set of nodes with which this node shares an edge; moreover, node k is also included in $\mathcal{N}_k$. This comprises the neighborhood set of k. We will denote the cardinality of this set as n_k. For the needs of the section, we assume that the graph is a connected one; that is, there is at least one path of edges that connects any pair of nodes.

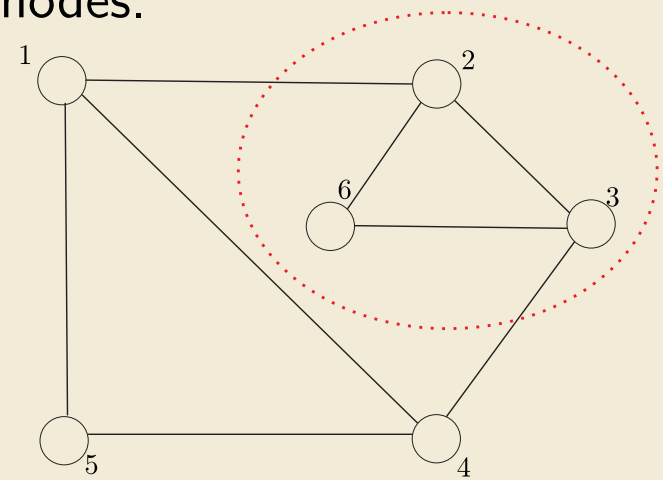


A graph corresponding to a network operating under a diffusion strategy. The red dotted line encircles the nodes comprising the neighborhood of node $k = 6$. As an example, the neighborhood of node $k = 6$ is $\mathcal{N}_6 = \{2, 3, 6\}$, with cardinality $n_6 = 3$. Also, On the contrary, nodes $k = 6$ and $k = 5$ are not neighbors.

- Each node in the network has access to a local data acquisition process, that provides the pair of training data $(y_k(n),\ \boldsymbol{x}_k(n))$, $k = 1, 2, \ldots, K$, $n = 0, 1, \ldots$. We further assume that, in all cases, the pairs of the input-output variables are associated with a common to all (unknown) parameter vector $\boldsymbol{\theta}_o$.

Diffusion Gradient Descent

- Let us consider a network of K agents/nodes. Each node exchanges information with the nodes in its neighborhood. Given a node, k, in a graph, let $\mathcal{N}_k$ be the set of nodes with which this node shares an edge; moreover, node k is also included in $\mathcal{N}_k$. This comprises the neighborhood set of k. We will denote the cardinality of this set as n_k. For the needs of the section, we assume that the graph is a connected one; that is, there is at least one path of edges that connects any pair of nodes.



A graph corresponding to a network operating under a diffusion strategy. The red dotted line encircles the nodes comprising the neighborhood of node $k = 6$. As an example, the neighborhood of node $k = 6$ is $\mathcal{N}_6 = \{2, 3, 6\}$, with cardinality $n_6 = 3$. Also, On the contrary, nodes $k = 6$ and $k = 5$ are not neighbors.

- Each node in the network has access to a local data acquisition process, that provides the pair of training data $(y_k(n),\ \boldsymbol{x}_k(n))$, $k = 1, 2, \ldots, K$, $n = 0, 1, \ldots$. We further assume that, in all cases, the pairs of the input-output variables are associated with a common to all (unknown) parameter vector $\boldsymbol{\theta}_o$.

Diffusion Gradient Descent

- Let us consider that, in every node, the data are generated by a corresponding regression model

$$\mathrm{y}_k = \boldsymbol{\theta}_o^T \mathbf{x}_k + \eta_k, \;\; k = 1, 2, \ldots, K,$$

where $\mathbf{x}_k$, as well as the zero mean noise variable, η_k, obey **different** statistical properties in each node.

- Treating each node individually, the MSE optimal solution which minimizes the local cost function

$$J_k(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y}_k - \boldsymbol{\theta}^T \mathbf{x}_k|^2\right],$$

will be given by the respective normal equations, involving the respective covariance matrix and cross-correlation vector, i.e.,

$$\Sigma_{x_k} \boldsymbol{\theta}_* = \boldsymbol{p}_k.$$

Recall that for the case of a regression model, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$ and the same solution results from all nodes. No doubt, if the statistics $\Sigma_{x_k}, \; \boldsymbol{p}_k, \; k = 1, 2, \ldots, K$, were known we could stop here.

- However, in practice, they have to be estimated and one has to resort to iterative techniques to learn the statistics as well as the unknown parameters. Thus, one has to consider all nodes, in order to benefit from all the observations, which are distributed across the network.

Diffusion Gradient Descent

- Let us consider that, in every node, the data are generated by a corresponding regression model

$$\mathrm{y}_k = \boldsymbol{\theta}_o^T \mathbf{x}_k + \eta_k, \;\; k = 1, 2, \ldots, K,$$

where $\mathbf{x}_k$, as well as the zero mean noise variable, η_k, obey **different** statistical properties in each node.

- Treating each node individually, the MSE optimal solution which minimizes the local cost function

$$J_k(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y}_k - \boldsymbol{\theta}^T \mathbf{x}_k|^2\right],$$

will be given by the respective normal equations, involving the respective covariance matrix and cross-correlation vector, i.e.,

$$\Sigma_{x_k} \boldsymbol{\theta}_* = \boldsymbol{p}_k.$$

Recall that for the case of a regression model, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$ and the same solution results from all nodes. No doubt, if the statistics $\Sigma_{x_k}, \; \boldsymbol{p}_k, \; k = 1, 2, \ldots, K$, were known we could stop here.

- However, in practice, they have to be estimated and one has to resort to iterative techniques to learn the statistics as well as the unknown parameters. Thus, one has to consider all nodes, in order to benefit from all the observations, which are distributed across the network.

Diffusion Gradient Descent

- Let us consider that, in every node, the data are generated by a corresponding regression model

$$\mathrm{y}_k = \boldsymbol{\theta}_o^T \mathbf{x}_k + \eta_k, \;\; k = 1, 2, \ldots, K,$$

where $\mathbf{x}_k$, as well as the zero mean noise variable, η_k, obey **different** statistical properties in each node.

- Treating each node individually, the MSE optimal solution which minimizes the local cost function

$$J_k(\boldsymbol{\theta}) = \mathbb{E}\left[|\mathrm{y}_k - \boldsymbol{\theta}^T \mathbf{x}_k|^2\right],$$

will be given by the respective normal equations, involving the respective covariance matrix and cross-correlation vector, i.e.,

$$\Sigma_{x_k} \boldsymbol{\theta}_* = \boldsymbol{p}_k.$$

Recall that for the case of a regression model, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$ and the same solution results from all nodes. No doubt, if the statistics $\Sigma_{x_k}, \; \boldsymbol{p}_k, \; k = 1, 2, \ldots, K$, were known we could stop here.

- However, in practice, they have to be estimated and one has to resort to iterative techniques to learn the statistics as well as the unknown parameters. Thus, one has to consider all nodes, in order to benefit from all the observations, which are distributed across the network.

Diffusion Gradient Descent

- Thus, a more natural criterion to adopt is

$$J(\boldsymbol{\theta}) = \sum_{k=1}^{K} J_k(\boldsymbol{\theta}) = \sum_{k=1}^{K} \mathbb{E}\left[\left|\mathrm{y}_k - \boldsymbol{\theta}^T \mathbf{x}_k\right|^2\right]. \tag{13}$$

  Using the standard arguments, it is readily seen that the (common) estimate of the unknown $\boldsymbol{\theta}_o$ will be provided as a solution of

$$\Big(\sum_{k=1}^{K} \Sigma_{x_k}\Big)\boldsymbol{\theta}_* = \sum_{k=1}^{K} \boldsymbol{p}_k.$$

- Let us use the global cost in (13) as our kick off point to apply a gradient descent optimization scheme

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu \sum_{k=1}^{K} \left(\boldsymbol{p}_k - \Sigma_{x_k}\boldsymbol{\theta}^{(i-1)}\right),$$

  from which a corresponding stochastic gradient scheme results, by replacing expectations with instantaneous observations and associating iteration steps with time updates, i.e.,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \sum_{k=1}^{K} \boldsymbol{x}_k(n) e_k(n), \quad e_k(n) = y_k(n) - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_k(n).$$

Diffusion Gradient Descent

- Thus, a more natural criterion to adopt is

$$J(\boldsymbol{\theta}) = \sum_{k=1}^{K} J_k(\boldsymbol{\theta}) = \sum_{k=1}^{K} \mathbb{E}\left[\left|\mathrm{y}_k - \boldsymbol{\theta}^T \mathbf{x}_k\right|^2\right]. \tag{13}$$

  Using the standard arguments, it is readily seen that the (common) estimate of the unknown $\boldsymbol{\theta}_o$ will be provided as a solution of

$$\Big(\sum_{k=1}^{K} \Sigma_{x_k}\Big)\boldsymbol{\theta}_* = \sum_{k=1}^{K} \boldsymbol{p}_k.$$

- Let us use the global cost in (13) as our kick off point to apply a gradient descent optimization scheme

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu \sum_{k=1}^{K} \left(\boldsymbol{p}_k - \Sigma_{x_k}\boldsymbol{\theta}^{(i-1)}\right),$$

  from which a corresponding stochastic gradient scheme results, by replacing expectations with instantaneous observations and associating iteration steps with time updates, i.e.,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu \sum_{k=1}^{K} \boldsymbol{x}_k(n) e_k(n), \quad e_k(n) = y_k(n) - \boldsymbol{\theta}_{n-1}^T \boldsymbol{x}_k(n).$$

Diffusion Gradient Descent

- Such an LMS-type recursion is perfect for a centralized scenario, where all data are transmitted to a fusion center. This is one of the extremes, having at its opposite end the scenario involving nodes acting individually without cooperation. However, there is an intermediate path, which will lead us to the distributed diffusion mode of operation.
- Instead of trying to minimize (13), let us select a specific node k, and construct a local cost as the weighted aggregate in $\mathcal{N}_k$, i.e.,

$$J_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}), \quad k = 1, 2, \ldots, K, \tag{14}$$

so that

$$\sum_{k=1}^{K} c_{mk} = 1, \; c_{mk} \geq 0, \text{ and } c_{mk} = 0 \text{ if } m \notin \mathcal{N}_k, \; m = 1, 2, \ldots, K.$$

- Stochastic matrices: Let C be the $K \times K$ matrix with entries $[C]_{mk} = c_{mk}$, then the summation condition above can be written as

$$C\mathbf{1} = \mathbf{1},$$

where $\mathbf{1}$ is the vector with all its entries being equal to 1. That is, all the entries across a row are summing to 1. Such matrices are known as right stochastic matrices. In contrast, a matrix is said to be left stochastic if

$$C^T\mathbf{1} = \mathbf{1}.$$

Also, a matrix that is both left and right stochastic is known as doubly stochastic.

Diffusion Gradient Descent

- Such an LMS-type recursion is perfect for a centralized scenario, where all data are transmitted to a fusion center. This is one of the extremes, having at its opposite end the scenario involving nodes acting individually without cooperation. However, there is an intermediate path, which will lead us to the distributed diffusion mode of operation.
- Instead of trying to minimize (13), let us select a specific node k, and construct a local cost as the weighted aggregate in $\mathcal{N}_k$, i.e.,

$$J_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}), \quad k = 1, 2, \ldots, K, \tag{14}$$

so that

$$\sum_{k=1}^{K} c_{mk} = 1, \; c_{mk} \geq 0, \text{ and } c_{mk} = 0 \text{ if } m \notin \mathcal{N}_k, \; m = 1, 2, \ldots, K.$$

- Stochastic matrices: Let C be the $K \times K$ matrix with entries $[C]_{mk} = c_{mk}$, then the summation condition above can be written as

$$C\mathbf{1} = \mathbf{1},$$

where $\mathbf{1}$ is the vector with all its entries being equal to 1. That is, all the entries across a row are summing to 1. Such matrices are known as right stochastic matrices. In contrast, a matrix is said to be left stochastic if

$$C^T\mathbf{1} = \mathbf{1}.$$

Also, a matrix that is both left and right stochastic is known as doubly stochastic.

Diffusion Gradient Descent

- Such an LMS-type recursion is perfect for a centralized scenario, where all data are transmitted to a fusion center. This is one of the extremes, having at its opposite end the scenario involving nodes acting individually without cooperation. However, there is an intermediate path, which will lead us to the distributed diffusion mode of operation.
- Instead of trying to minimize (13), let us select a specific node k, and construct a local cost as the weighted aggregate in $\mathcal{N}_k$, i.e.,

$$J_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}), \quad k = 1, 2, \ldots, K, \tag{14}$$

so that

$$\sum_{k=1}^{K} c_{mk} = 1, \; c_{mk} \geq 0, \text{ and } c_{mk} = 0 \text{ if } m \notin \mathcal{N}_k, \; m = 1, 2, \ldots, K.$$

- Stochastic matrices: Let C be the $K \times K$ matrix with entries $[C]_{mk} = c_{mk}$, then the summation condition above can be written as

$$C\mathbf{1} = \mathbf{1},$$

where $\mathbf{1}$ is the vector with all its entries being equal to 1. That is, all the entries across a row are summing to 1. Such matrices are known as right stochastic matrices. In contrast, a matrix is said to be left stochastic if

$$C^T\mathbf{1} = \mathbf{1}.$$

Also, a matrix that is both left and right stochastic is known as doubly stochastic.

Diffusion Gradient Descent

- Note that due to this matrix constraint, we still have that

$$\sum_{k=1}^{K} J_k^{loc}(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m=1}^{K} c_{mk} J_m(\boldsymbol{\theta}) = J(\boldsymbol{\theta})$$

  That is, summing all local costs, the global one results.

- Let us focus on minimizing (14). The gradient descent scheme results in

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right).$$

- However, since nodes in the neighborhood exchange information, they could also share their current estimates. This is justified by the fact that the ultimate goal is to reach a common estimate; thus, knowing each others current estimates could be used for the benefit of the algorithmic process in order to achieve this goal. To this end, we will modify the cost in (14) by regularizing it, i.e.,

$$\tilde{J}_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}||^2, \tag{15}$$

  where $\tilde{\theta}$ encodes information with respect to the unknown vector, which is obtained by the neighboring nodes and $\lambda > 0$.

Diffusion Gradient Descent

- Note that due to this matrix constraint, we still have that

$$\sum_{k=1}^{K} J_k^{loc}(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m=1}^{K} c_{mk} J_m(\boldsymbol{\theta}) = J(\boldsymbol{\theta})$$

  That is, summing all local costs, the global one results.

- Let us focus on minimizing (14). The gradient descent scheme results in

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right).$$

- However, since nodes in the neighborhood exchange information, they could also share their current estimates. This is justified by the fact that the ultimate goal is to reach a common estimate; thus, knowing each others current estimates could be used for the benefit of the algorithmic process in order to achieve this goal. To this end, we will modify the cost in (14) by regularizing it, i.e.,

$$\tilde{J}_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}||^2, \qquad (15)$$

  where $\tilde{\theta}$ encodes information with respect to the unknown vector, which is obtained by the neighboring nodes and $\lambda > 0$.

Diffusion Gradient Descent

- Note that due to this matrix constraint, we still have that

$$\sum_{k=1}^{K} J_k^{loc}(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{m=1}^{K} c_{mk} J_m(\boldsymbol{\theta}) = J(\boldsymbol{\theta})$$

  That is, summing all local costs, the global one results.

- Let us focus on minimizing (14). The gradient descent scheme results in

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right).$$

- However, since nodes in the neighborhood exchange information, they could also share their current estimates. This is justified by the fact that the ultimate goal is to reach a common estimate; thus, knowing each others current estimates could be used for the benefit of the algorithmic process in order to achieve this goal. To this end, we will modify the cost in (14) by regularizing it, i.e.,

$$\tilde{J}_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}||^2, \tag{15}$$

  where $\tilde{\boldsymbol{\theta}}$ encodes information with respect to the unknown vector, which is obtained by the neighboring nodes and $\lambda > 0$.

Diffusion Gradient Descent

- Applying the gradient descent scheme (and absorbing the factor "2", which comes from the exponents, into the step-size), we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right) + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}),$$

which can be broken into the following two steps:

$$\begin{aligned} \text{Step 1:} \quad \boldsymbol{\psi}_k^{(i)} &= \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right), \\ \text{Step 2:} \quad \boldsymbol{\theta}_k^{(i)} &= \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}). \end{aligned}$$

- Step 2 can slightly be modified and replace $\boldsymbol{\theta}_k^{(i-1)}$ by $\boldsymbol{\psi}_k^{(i)}$, since this encodes more recent information, and we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\psi}_k^{(i)}).$$

- Furthermore, a reasonable choice of $\tilde{\boldsymbol{\theta}}$, at each iteration step, would be

$$\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}^{(i)} := \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} \boldsymbol{\psi}_m^{(i)}, \text{ where } \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} = 1, \; b_{mk} \geq 0,$$

and $\mathcal{N}_{k \setminus k}$ denotes the elements in $\mathcal{N}_k$ excluding k.

Diffusion Gradient Descent

- Applying the gradient descent scheme (and absorbing the factor "2", which comes from the exponents, into the step-size), we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right) + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}),$$

  which can be broken into the following two steps:

$$\begin{aligned} \text{Step 1:} \quad \boldsymbol{\psi}_k^{(i)} &= \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right), \\ \text{Step 2:} \quad \boldsymbol{\theta}_k^{(i)} &= \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}). \end{aligned}$$

- Step 2 can slightly be modified and replace $\boldsymbol{\theta}_k^{(i-1)}$ by $\boldsymbol{\psi}_k^{(i)}$, since this encodes more recent information, and we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\psi}_k^{(i)}).$$

- Furthermore, a reasonable choice of $\tilde{\theta}$, at each iteration step, would be

$$\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}^{(i)} := \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} \boldsymbol{\psi}_m^{(i)}, \text{ where } \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} = 1, \; b_{mk} \geq 0,$$

  and $\mathcal{N}_{k \setminus k}$ denotes the elements in $\mathcal{N}_k$ excluding k.

Diffusion Gradient Descent

- Applying the gradient descent scheme (and absorbing the factor "2", which comes from the exponents, into the step-size), we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right) + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}),$$

  which can be broken into the following two steps:

$$\begin{aligned} \text{Step 1:} \quad \boldsymbol{\psi}_k^{(i)} &= \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right), \\ \text{Step 2:} \quad \boldsymbol{\theta}_k^{(i)} &= \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}). \end{aligned}$$

- Step 2 can slightly be modified and replace $\boldsymbol{\theta}_k^{(i-1)}$ by $\boldsymbol{\psi}_k^{(i)}$, since this encodes more recent information, and we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\psi}_k^{(i)}).$$

- Furthermore, a reasonable choice of $\tilde{\boldsymbol{\theta}}$, at each iteration step, would be

$$\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}^{(i)} := \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} \boldsymbol{\psi}_m^{(i)}, \text{ where } \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} = 1, \; b_{mk} \geq 0,$$

  and $\mathcal{N}_{k \setminus k}$ denotes the elements in $\mathcal{N}_k$ excluding k.

Diffusion Gradient Descent

- In other words, at each iteration, we update $\boldsymbol{\theta}_k$ so that to move it towards the descent direction of the local cost and at the same time we **constrain it to stay close to the convex combination** of the rest of the updates, which are obtained during the computations in step 1 from all the nodes in its neighborhood. Thus, we end up with the following recursions:

  Diffusion Gradient Descent

$$\text{Step 1:} \quad \boldsymbol{\psi}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} \left(\boldsymbol{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)} \right),$$

$$\text{Step 2:} \quad \boldsymbol{\theta}_k^{(i)} = \sum_{m \in \mathcal{N}_k} a_{mk} \boldsymbol{\psi}_m^{(i)}.$$

- Note that we have set

$$a_{kk} := 1 - \mu_k \lambda, \; a_{mk} := \mu_k \lambda b_{mk} \Longrightarrow \sum_{m \in \mathcal{N}_k} a_{mk} = 1 \; , \quad a_{mk} \geq 0,$$

  for small enough values of $\mu_k \lambda$. Setting $a_{mk} = 0$, $m \notin \mathcal{N}_k$ and defining A to be the matrix with entries $[A]_{mk} = a_{mk}$, we have that $A^T \mathbf{1} = 1$. That is, A is a left stochastic matrix. Note that, any left stochastic matrix A can also be used.

Diffusion LMS

- To state the diffusion LMS, we only have to replace in gradient descent scheme the expectations with instantaneous observations and interpreting iterations as time updates.

  The Adapt-then-Combine Diffusion LMS

  - Initialize
    - **For** $k = 1, 2, \ldots, K$, **Do**
      - $\boldsymbol{\theta}_k(-1) = \mathbf{0} \in \mathbb{R}^l$; or any other value.
    - **End For**
    - Select $\mu_k,\ k = 1, 2, \ldots, K$; a small positive number.
    - Select $C: \quad C\mathbf{1} = \mathbf{1}$
    - Select $A: \quad A^T\mathbf{1} = \mathbf{1}$
  - **For** $n = 0, 1, \ldots,$ **Do**
    - **For** $k = 1, 2, \ldots, K$, **Do**
      - **For** $m \in \mathcal{N}_k$, **Do**
      - $e_{k,m}(n) = y_m(n) - \boldsymbol{\theta}_k^T(n-1)\boldsymbol{x}_m(n)$
      - **End For**
      - $\boldsymbol{\psi}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk}\boldsymbol{x}_m(n)e_{k,m}(n)$;
    - **End For**
    - **For** $k = 1, 2, \ldots, K$
      - $\boldsymbol{\theta}_k(n) = \sum_{m \in \mathcal{N}_k} a_{mk}\boldsymbol{\psi}_m(n)$
    - **End For**
  - **End For**

Diffusion LMS

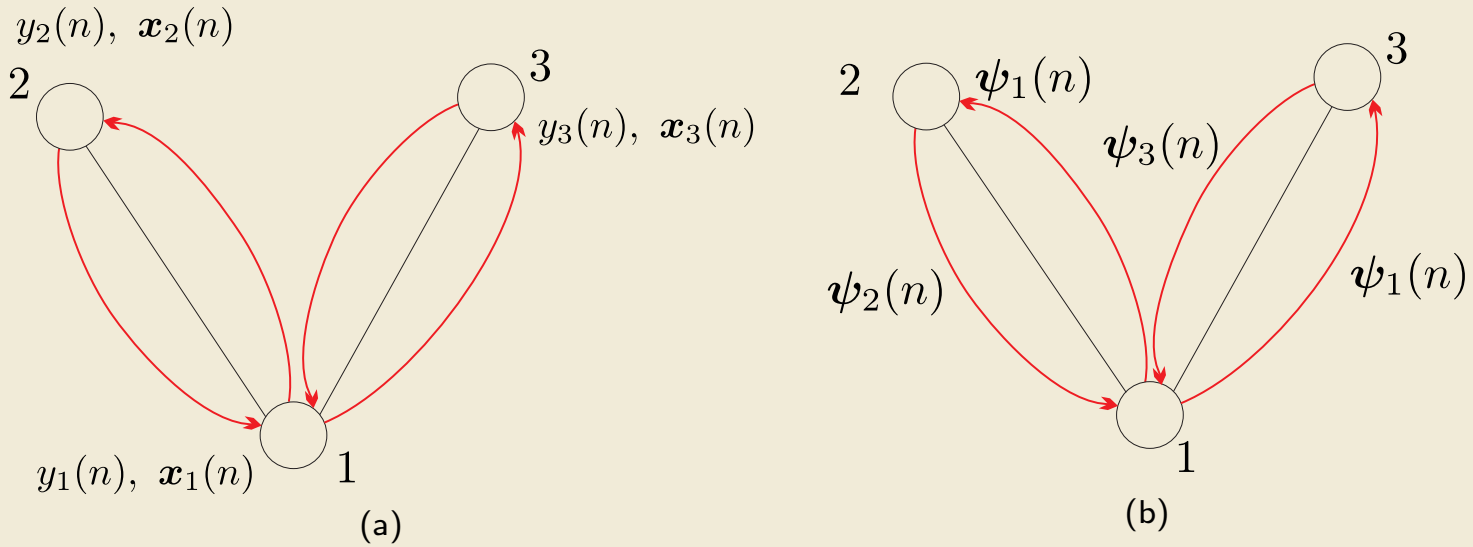
- The following comments are in order:
  - This form of diffusion LMS (DiLMS) is known as Adapt-then-Combine DiLMS (ATC) since the first step refers to the update and the combination step follows. If one reverts the orders of the two steps the version known as Combine-then-Adapt DiLMS (CTA) results.
  - In the special case of $C = I$, then the adaptation step becomes
    $$\boldsymbol{\psi}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu \boldsymbol{x}_k(n) e_k(n),$$
    and nodes need not exchange their observations/measuremnts.
  - Two popular paths for the choices of $C(A)$ are:

    Averaging Rule:
    $$c_{mk} = \begin{cases} \frac{1}{n_k}, & \text{if } k = m, \text{ or if nodes } k \text{ and } m \text{ are neighbors,} \\ 0, & \text{otherwise,} \end{cases}$$
    and the respective matrix is left stochastic.

    Metropolis Rule:
    $$c_{mk} = \begin{cases} \frac{1}{\max\{n_k, n_m\}}, & \text{if } k \neq m \text{ and } k, m \text{ are neighbors,} \\ 1 - \sum_{i \in \mathcal{N}_k \setminus k} c_{ik}, & m = k, \\ 0, & \text{otherwise,} \end{cases}$$
    which makes the respective matrix to be doubly stochastic.

Diffusion LMS

- At time n, all three neighbors exchange the received data. In case the input vector corresponds to a realization of a random signal, $\mathrm{u}_k(n)$, the exchange of information comprises two values, i.e., $(y_k(n), u_k(n))$ in each direction for each one of the links. In the more general case, where the input is a random vector of jointly distributed variables, then all l variables have to be exchanged. After this message passing, adaptation is taken place as shown in Figure (a). Then, the nodes exchange their obtained estimates, $\boldsymbol{\psi}_k(n),\ k = 1, 2, 3$, across the links, Figure (b).



a) In step 1, adaptation is carried out after the exchange of the received observations. b) In step 2, the nodes exchange their locally computed estimates to obtain the updated one.

Some Hints On Convergence and Steady State Performance

- The previously stated diffusion gradient descent scheme is guaranteed to converge, i.e.,

$$\boldsymbol{\theta}_k^{(i)} \xrightarrow[i\to\infty]{} \boldsymbol{\theta}_*,$$

provided that

$$\mu_k \leq \frac{2}{\lambda_{\max}\{\Sigma_k^{loc}\}}, \text{ where } \Sigma_k^{loc} = \sum_{m\in\mathcal{N}_k} c_{mk}\Sigma_{x_m}.$$

This is the counterpart of the condition that holds for the standard gradient descent scheme.

- For the LMS and under a number of assumptions, stated in the text, the following hold:
  Convergence in the Mean: Provided that

$$\mu_k < \frac{2}{\lambda_{\max}\{\Sigma_k^{loc}\}},$$

then

$$\mathbb{E}\big[\boldsymbol{\theta}_k(n)\big] \xrightarrow[n\to\infty]{} \boldsymbol{\theta}_*, \quad k = 1, 2, \ldots, K.$$

It is important to state here that the previous stability condition depends on C and not on A. If in addition to the previous assumption, C is chosen to be doubly stochastic, then the convergence in the mean, in any node under the distributed scenario, is **faster** than that obtained if the node is operating individually without cooperation, provided $\mu_k = \mu$ is the same and it is chosen so that to guarantee convergence.

Some Hints On Convergence and Steady State Performance

- The previously stated diffusion gradient descent scheme is guaranteed to converge, i.e.,

$$\boldsymbol{\theta}_k^{(i)} \xrightarrow[i\to\infty]{} \boldsymbol{\theta}_*,$$

provided that

$$\mu_k \leq \frac{2}{\lambda_{\max}\{\Sigma_k^{loc}\}}, \text{ where } \Sigma_k^{loc} = \sum_{m\in\mathcal{N}_k} c_{mk}\Sigma_{x_m}.$$

This is the counterpart of the condition that holds for the standard gradient descent scheme.

- For the LMS and under a number of assumptions, stated in the text, the following hold:
  Convergence in the Mean: Provided that

$$\mu_k < \frac{2}{\lambda_{\max}\{\Sigma_k^{\text{loc}}\}},$$

then

$$\mathbb{E}\big[\boldsymbol{\theta}_k(n)\big] \xrightarrow[n\to\infty]{} \boldsymbol{\theta}_*, \quad k = 1, 2, \ldots, K.$$

It is important to state here that the previous stability condition depends on C and not on A. If in addition to the previous assumption, C is chosen to be doubly stochastic, then the convergence in the mean, in any node under the distributed scenario, is **faster** than that obtained if the node is operating individually without cooperation, provided $\mu_k = \mu$ is the same and it is chosen so that to guarantee convergence.

Some Hints On Convergence and Steady State Performance

- Misadjustment: under the assumptions of C and A being doubly stochastic, the following are true:
  - The average misadjustment over all nodes in the steady-state for the Adapt-then-Combine strategy is always smaller than that of the Combine-then-Adapt one.
  - The average misadjustment over all the nodes of the network in the distributed operation is **always lower** than that obtained if nodes are adapted individually, without cooperation, by using the same $\mu_k = \mu$ in all cases. That is, cooperation does not only improve convergence speed but it also improves the steady-state performance.

A Simulation Example

- In this example, a network of $L = 10$ nodes is considered. The nodes were randomly connected with a total number of 32 connections; the resulting network was checked out that it was strongly connected. In each node, data are generated according to a regression model, using the same vector $\boldsymbol{\theta}_o \in \mathbb{R}^{30}$. The latter was randomly generated by a $\mathcal{N}(0, 1)$. The input vectors, $\boldsymbol{x}_k$, were i.i.d generated according to a $\mathcal{N}(0, 1)$ and the noise level was different for each node, varying from 20-25 dBs.
- Three experiments were carried out. The first involved the distributed LMS in its ATC form and the second one the CTA version. In the third experiment, the LMS algorithm was run independently for each node, without cooperation. In all cases, the step-size was chosen equal to $\mu = 0.01$.
- The average (over all nodes) MSD(n) : $\frac{1}{K}\sum_{k=1}^{K}||\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o||^2$ was computed obtained for each one of the experiments. As it is verified by the next figure, cooperation improves the performance significantly, both in terms of convergence as well as in steady-state error floor. Moreover, the ATC performs slightly better that the CTA version.

A Simulation Example

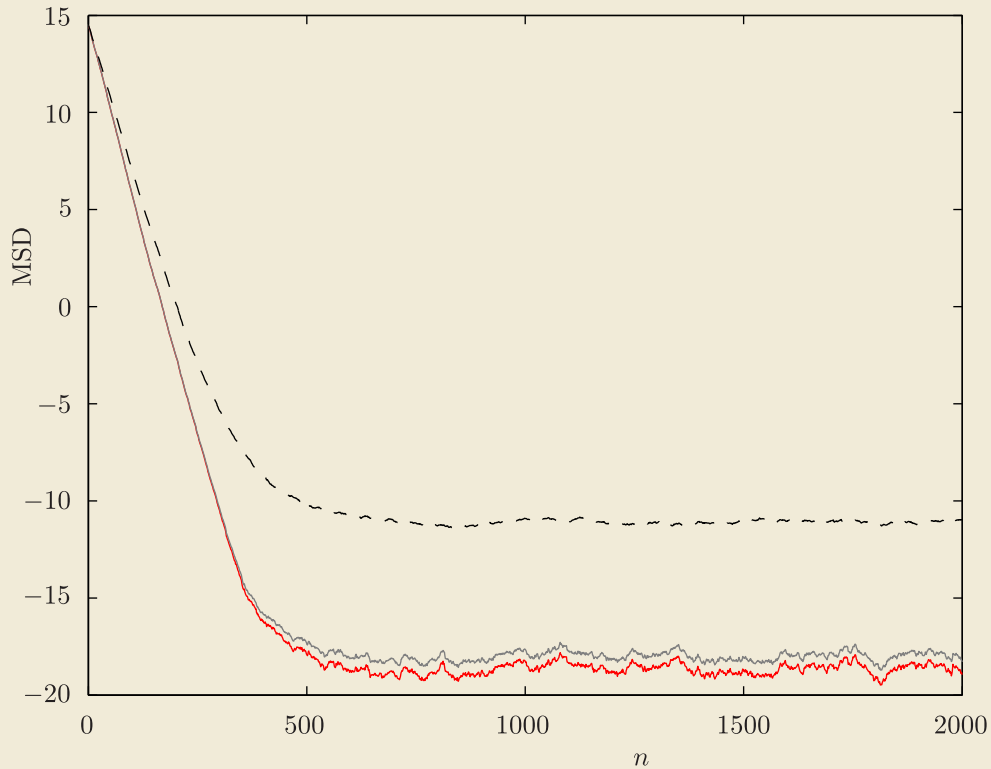
- In this example, a network of $L = 10$ nodes is considered. The nodes were randomly connected with a total number of 32 connections; the resulting network was checked out that it was strongly connected. In each node, data are generated according to a regression model, using the same vector $\boldsymbol{\theta}_o \in \mathbb{R}^{30}$. The latter was randomly generated by a $\mathcal{N}(0, 1)$. The input vectors, $\boldsymbol{x}_k$, were i.i.d generated according to a $\mathcal{N}(0, 1)$ and the noise level was different for each node, varying from 20-25 dBs.
- Three experiments were carried out. The first involved the distributed LMS in its ATC form and the second one the CTA version. In the third experiment, the LMS algorithm was run independently for each node, without cooperation. In all cases, the step-size was chosen equal to $\mu = 0.01$.
- The average (over all nodes) MSD(n) : $\frac{1}{K}\sum_{k=1}^{K}||\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o||^2$ was computed obtained for each one of the experiments. As it is verified by the next figure, cooperation improves the performance significantly, both in terms of convergence as well as in steady-state error floor. Moreover, the ATC performs slightly better that the CTA version.

A Simulation Example

- In this example, a network of $L = 10$ nodes is considered. The nodes were randomly connected with a total number of 32 connections; the resulting network was checked out that it was strongly connected. In each node, data are generated according to a regression model, using the same vector $\boldsymbol{\theta}_o \in \mathbb{R}^{30}$. The latter was randomly generated by a $\mathcal{N}(0, 1)$. The input vectors, $\boldsymbol{x}_k$, were i.i.d generated according to a $\mathcal{N}(0, 1)$ and the noise level was different for each node, varying from 20-25 dBs.
- Three experiments were carried out. The first involved the distributed LMS in its ATC form and the second one the CTA version. In the third experiment, the LMS algorithm was run independently for each node, without cooperation. In all cases, the step-size was chosen equal to $\mu = 0.01$.
- The average (over all nodes) MSD(n) : $\frac{1}{K}\sum_{k=1}^{K}||\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o||^2$ was computed obtained for each one of the experiments. As it is verified by the next figure, cooperation improves the performance significantly, both in terms of convergence as well as in steady-state error floor. Moreover, the ATC performs slightly better that the CTA version.

A Simulation Example

- The figure shows the average (over all nodes) $\text{MSD}(n) : \frac{1}{K}\sum_{k=1}^{K} ||\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o||^2$ obtained for each one of the experiments.



Average (over all the nodes) error convergence curves (MSD) for the LMS in non-cooperative mode of operation (dotted line) and for the case of the diffusion LMS, in the ATC mode (red line) and the CTA mode (gray line). The step size μ was the same in all three cases. Co-operation among modes significantly improves performance. For the case of the diffusion LMS, the ATC version results in slightly better performance compared to that of the CTA.