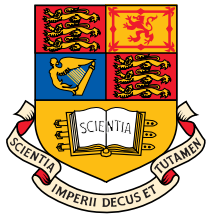
Adaptive SP & Machine Intelligence
Recursive Least Squares and Advanced Learning Systems

Danilo Mandic

room 813, ext: 46271



Department of Electrical and Electronic Engineering
Imperial College London, UK

d.mandic@imperial.ac.uk, URL: www.commsp.ee.ic.ac.uk/∼mandic

Outline:

Really, to de-mystify several concepts in Learning Systems

- Moving from a "stochastic" cost function to a "deterministic" least-squares based cost function
- Recursive 'deterministic' solution to the Wiener filter $\looparrowright$ Recursive Least Squares (RLS) (as opposed to the 'stochastic' LMS solution)
- Need for a matrix-valued stepsize, towards Kalman filter
- Regularisation of learning algorithms
- Incorporation of constraints (sparsity, smoothness, non-negativity)
- Ridge regression, LASSO, elastic net
- A general framework for training learning systems, white-, black-and grey-box modelling
- Connection between LMS, RLS, and Kalman filter
- Applications

The big picture of learning machines
two main families of algorithms

- **Gradient descent methods** (e.g. the Least Mean Square (LMS)) provide an **approximate minimisation** of the mean square error (MSE)

$$J \sim E\{e^2(n)\} \quad \text{for LMS} \quad J(n) \sim e^2(n) \qquad \textbf{stochastic}$$

  ◦ Knowledge of the autocorrelation of the input process and cross-correlation between the input an teaching signal required.

  ◦ Rapid convergence or sufficiently small excess MSE not guaranteed.

- **Least squares (LS) techniques** are based on the **exact minimisation** of a sum of instantaneous squared errors

$$J(n) = \sum_{i=0}^{n} e^2(i) = \sum_{i=0}^{n} \left| d(i) - \mathbf{x}^T(i)\mathbf{w}_n \right|^2 \qquad \textbf{deterministic}$$

  Deterministic cost function $\not\mapsto$ no statistical information about $\mathbf{x}$ and d!

☞ The 'deterministic' LS error depends on a particular combination of $x(n)$ and $d(n)$ $\not\mapsto$ for different signals we obtain different filters.

☞ The 'stochastic' MSE does not depend on a particular realisation of $x(n)$ and $d(n)$ $\not\mapsto$ produces equal weights for signals with the same statistics.

The Recursive Least Squares (RLS) algorithm

(recursive solution to the "deterministic" Wiener filtering problem)

Aim: For a filter of order N, find the weight vector $\mathbf{w}_n = \mathbf{w}_{opt}(n)$ which minimizes the sum of squares of output errors up until the time instant n

$$\mathbf{w}_n = \mathbf{w}_{opt}^{LS} \text{ over n time instants} \quad \Rightarrow \quad \mathbf{w}_n = arg \ \min_{\mathbf{w}} \sum_{i=0}^{n} \left| d(i) - \mathbf{x}^T(i)\mathbf{w}_n \right|^2$$

(the summation over i is performed for the **latest** set of coefficients $\mathbf{w}_n$)

☞ Notice, the weights $\mathbf{w}_n$ are **held constant** over the whole observation $[0, n]$ – similar to the Wiener setting (but a deterministic cost function)

To find $\mathbf{w}_n$: set $\nabla_{\mathbf{w}} J(n) = \mathbf{0}$ to solve for $\mathbf{w}_{opt}(n) = \mathbf{w}_n = \mathbf{R}^{-1}(n)\mathbf{p}(n)$

☞ **Careful here, as:** $\mathbf{R}(n) = \sum_{i=0}^{n} \mathbf{x}(i)\mathbf{x}^T(i), \quad \mathbf{p}(n) = \mathbf{r}_{d\mathbf{x}} = \sum_{i=0}^{n} d(i)\mathbf{x}(i)$

that is, $\mathbf{R}(n)$, $\mathbf{p}(n)$ are purely ***deterministic*** (*cf.* $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}, \mathbf{p} = E\{d\mathbf{x}\}$ in Wiener filt.)

Recursive least squares (RLS) summary. The aim is to recursively update:

(a) $J(n+1) = J(n) + |e(n+1)|^2$ (b) $\mathbf{p}(n+1) = \mathbf{p}(n) + d(n+1)\mathbf{x}(n+1)$

(c) $\mathbf{R}(n+1) = \mathbf{R}(n) + \mathbf{x}(n+1)\mathbf{x}^T(n+1)$

The weight update: $\mathbf{w}_{n+1} = \mathbf{w}_{opt}(n+1) = \mathbf{w}(n+1) = \mathbf{R}^{-1}(n+1)\mathbf{p}(n+1)$

The RLS formulation $\looparrowright$ $\mathbf{R}^{-1}$ calculation is $\mathcal{O}(N^3)$

(we use the standard matrix inversion lemma, also known as Woodbury's identity)

Key to RLS: all we have to do is to employ a recursive update of $\mathbf{R}^{-1}$

Start from the matrix inversion lemma (also known as ABCD lemma)

$$(A+BCD)^{-1} = A^{-1} - A^{-1}B\big(DA^{-1}B + C^{-1}\big)^{-1}DA^{-1}$$

and set $A = \mathbf{R}(n), B = \mathbf{x}(n+1), C = 1, D = \mathbf{x}^T(n+1)$, to give

$$\mathbf{R}(n+1) = \mathbf{R}(n) + \mathbf{x}(n+1)\mathbf{x}^T(n+1) = A + BCD$$

Then $\mathbf{R}^{-1}(n+1)$ is given by

$$\mathbf{R}^{-1}(n+1) = \mathbf{R}^{-1}(n) - \frac{\mathbf{R}^{-1}(n)\mathbf{x}(n+1)\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)}{\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)\mathbf{x}(n+1)+1} \quad \looparrowright \quad \mathcal{O}(N^2)$$

$\looparrowright$ **we never compute $\mathbf{R}(n+1)$ or $\mathbf{R}^{-1}(n)$ directly, but recursively**

The optimal RLS weight vector: $\mathbf{w}_{opt}(n+1) = \mathbf{w}(n+1) + \mathbf{R}^{-1}(n+1)\mathbf{p}(n+1)$

Minimum LSE: $J_{min}^{LS} = \| \mathbf{d}(n) \|_2^2 - \mathbf{r}_{dx}^T(n)\mathbf{w}_n$

where $\mathbf{r}_{dx}(n) = \mathbf{p}(n), \mathbf{d}(n) = [d(0), \ldots, d(n)]^T$ (also see the Appendix)

RLS: Adaptive noise cancellation

The system has two microphones (primary & reference) *`rlsdemo` in Matlab

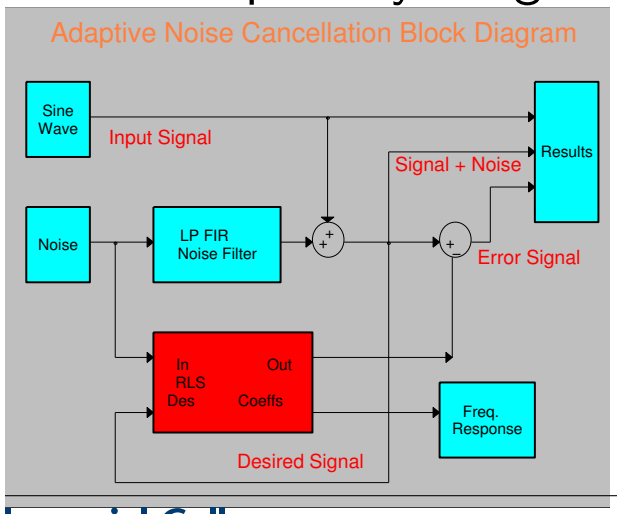
Similar scenario to noise-cancelling headphones.

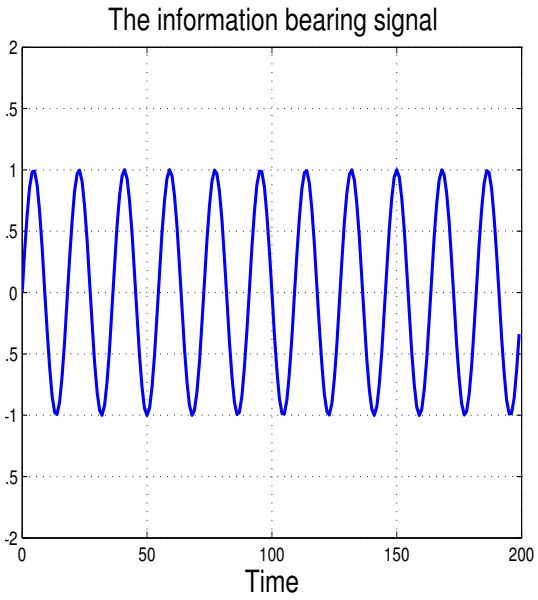
Primary signal:

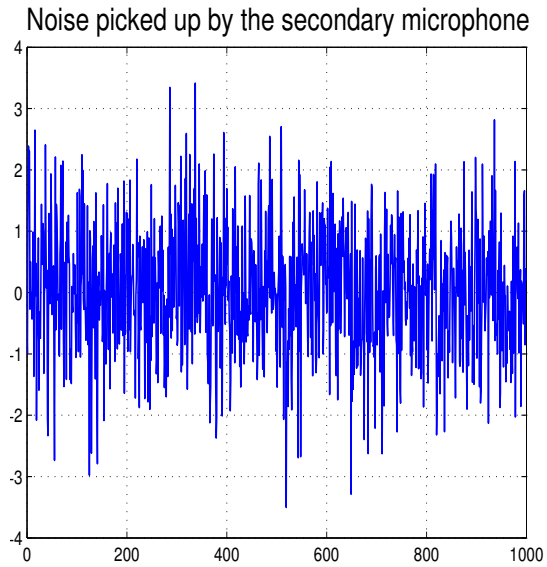
$$\sin(n) + q(n)$$

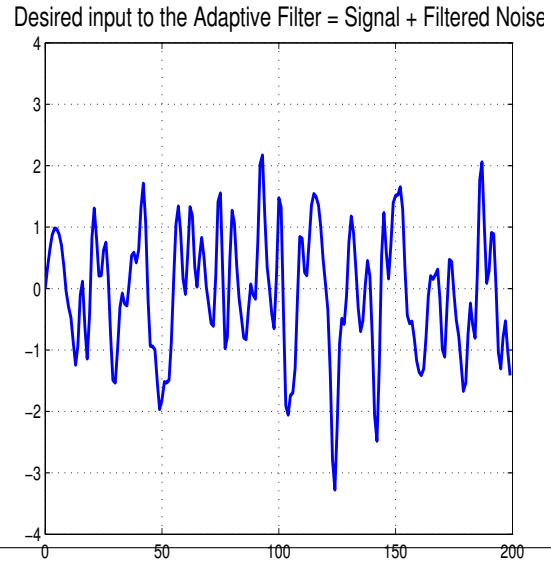
Reference signal:

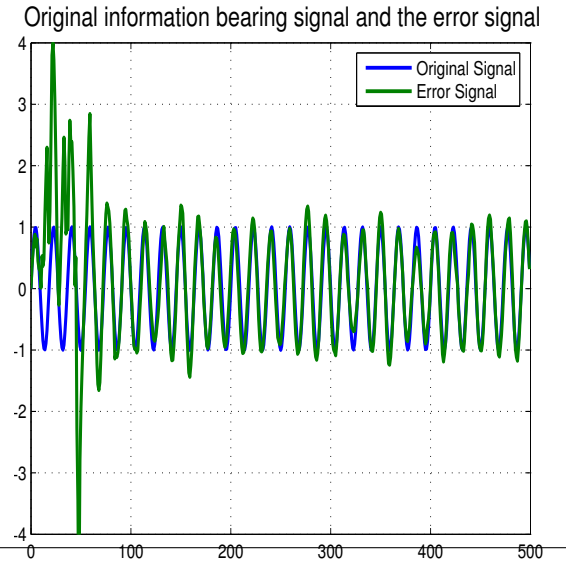
any noise correlated with the noise $q(n)$ in the primary signal











RLS with a forgetting factor

(a way to introduce weighting and fading memory into RLS)

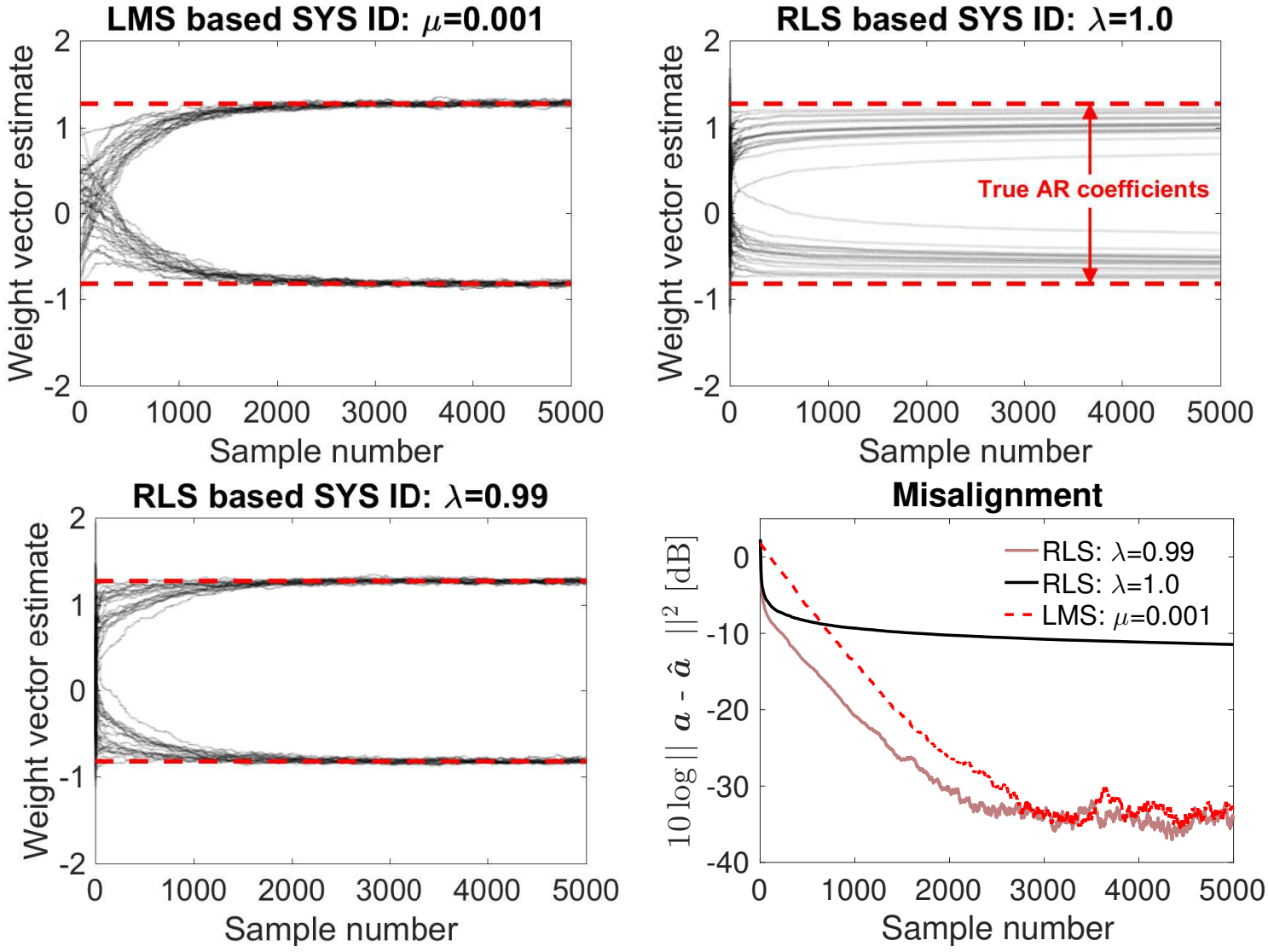
- The basic cost function for the RLS algorithm assumes a statistically stationary environment
- In the original cost function all the errors are weighted equally, which is wrong in the statistically nonstationary environment where distant past is not contributing to learning
- In order to deal with a nonstationary environment, modify the LS error criterion (recall the weighted least squares (WLS))

$$J(n) = \sum_{i=0}^{n} \lambda^{n-i} e^2(i) = [e_n, e_{n-1}, \ldots, e_0] \begin{bmatrix} \lambda^0 & 0 & \cdots \\ 0 & \lambda^1 & \cdots \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda^n \end{bmatrix} \begin{bmatrix} e_n \\ e_{n-1} \\ \vdots \\ e_0 \end{bmatrix} = \mathbf{e}^T \mathbf{W} \mathbf{e}$$

- In this way, the 'old' information is gradually forgotten
- Forgetting factor $\lambda \in (0, 1]$, but typically > 0.95
- The forgetting factor introduces an effective window length of $\frac{1}{1-\lambda}$

Comparison between LMS and RLS

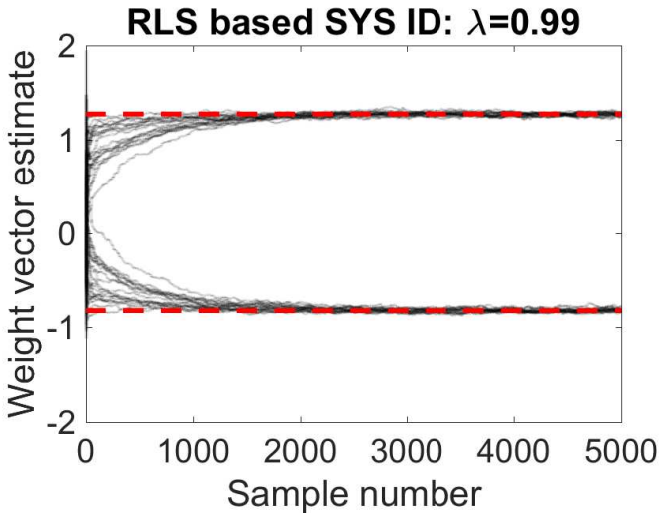
Consider SYS ID of an AR(2) process with coefficients $\mathbf{a} = [1.2728, -0.82]^\top$, driven by $w \sim \mathcal{N}(0, 1)$. Simulations performed over 10 independent realisations.

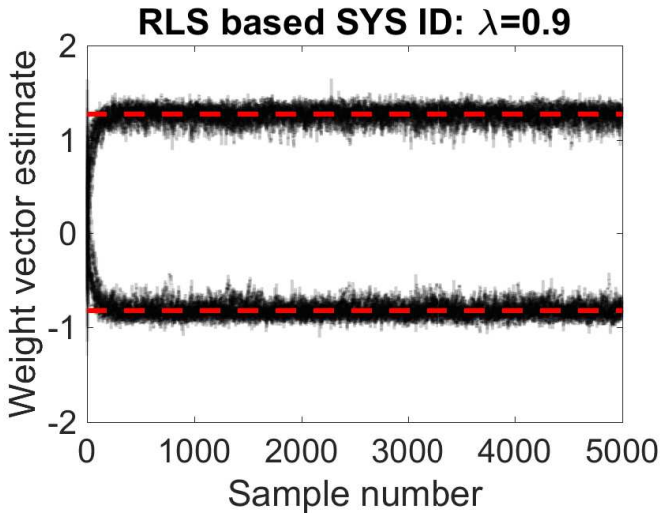


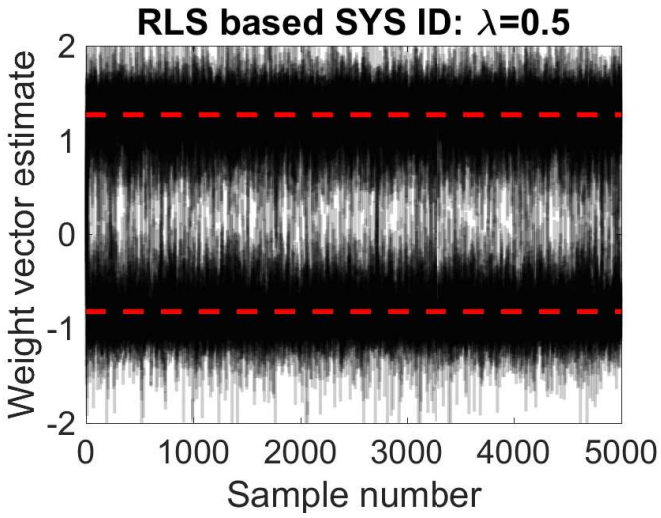
RLS with forgetting factor λ

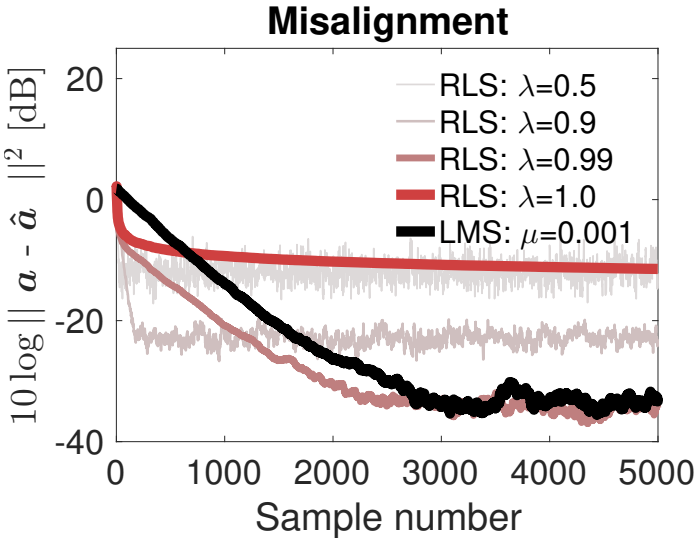
RLS_LMS_RUN_LIVE

with a careful choice of λ, RLS can outperform LMS









A compact form of RLS and geometry of learning

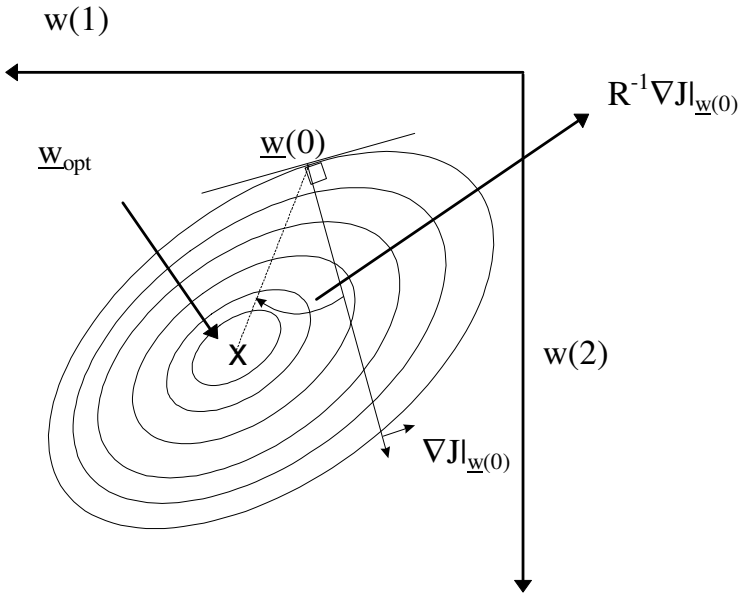
From $\mathbf{w}(n+1) = \mathbf{R}^{-1}(n+1)\mathbf{p}(n+1)$, we have

$$\mathbf{w}(n+1) = \left[\mathbf{R}^{-1}(n) - \frac{\mathbf{R}^{-1}(n)\mathbf{x}(n+1)\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)}{\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)\mathbf{x}(n+1)+1}\right]\left[\mathbf{p}(n) + d(n)\mathbf{x}(n)\right]$$

After some grouping of the terms above, we arrive at

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mathbf{R}^{-1}(n)}{1+\mathbf{x}^T(n)\mathbf{R}^{-1}(n)\mathbf{x}(n)}\, e(n)\mathbf{x}(n)$$

- the term $\mathbf{R}^{-1}$ "filters" the direction and length of the data vector $\mathbf{x}$
- The term $1+\mathbf{x}^T(n)\mathbf{R}^{-1}(n)\mathbf{x}(n)$ is a measure of input signal power, normalised by $\mathbf{R}^{-1}$.
- This normalisation makes the power proportional to the data length N and not to the actual signal level $\hookrightarrow$ it also decorrelates the data.



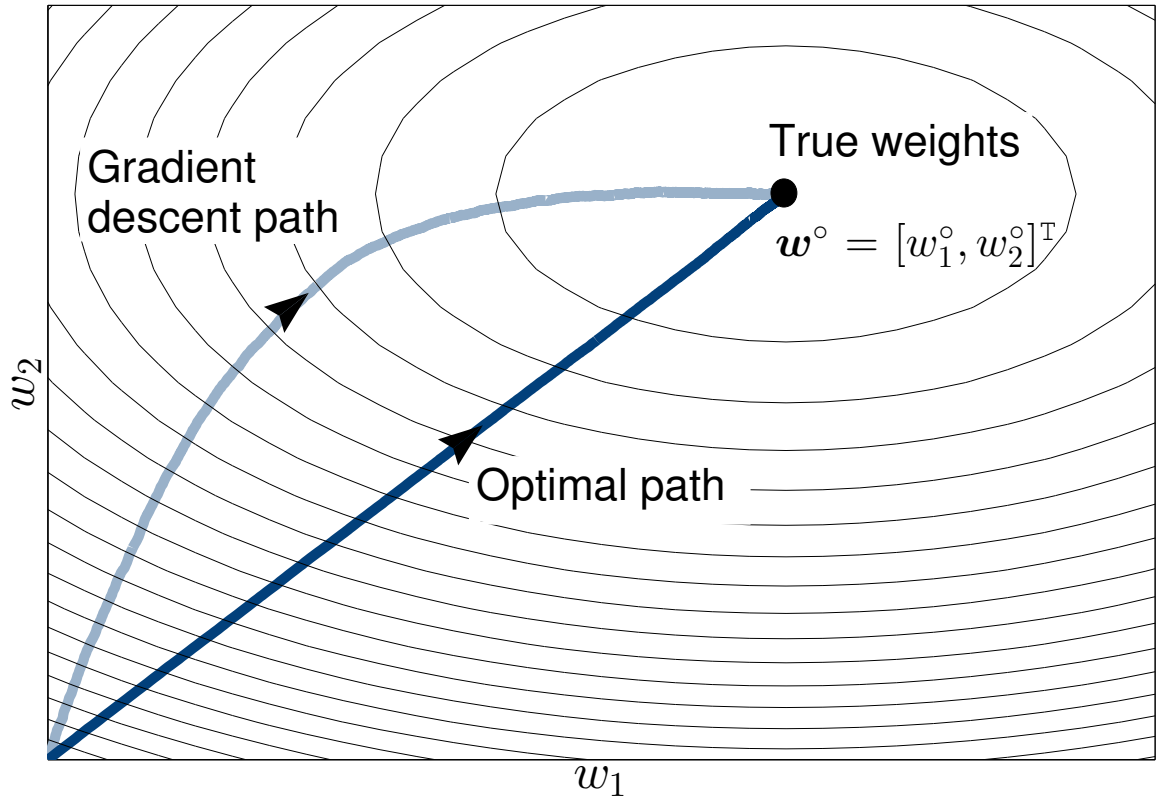
Things to remember about the RLS

- Results are exactly the same as for the standard least squares $\looparrowright$ no approximation involved
- RLS does not perform matrix inversion at any stage $\looparrowright$ it calculates the matrix inverse recursively
- The role of $\mathbf{R}^{-1}$ in RLS is to rotate the direction of descent of the LMS algorithm towards the minimum of the cost function independent of the nature, or colouration, of the input
- The operation $\mathbf{R}^{-1}\mathbf{x}[k]$ is effectively a **pre-whitening operation**, compare with transform domain adaptive filtering
- Unlike gradient algorithms, there is no explicit notion of learning rate
- RLS guarantees convergence to the optimal weight vector
- Variants of RLS (sliding window, forgetting factor) deal with pragmatic issues (nonstationarity etc)
- The price to pay is increased computational complexity compared to the LMS
- The convergence of the RLS in a high SNR environment ($> 10dB$) is of an order $\mathcal{O}(2p)$, whereas for LMS $\mathcal{O}(10p)$
- The misadjustment performance of RLS is essentially zero because it is deterministic and matches the data where $\lambda = 1$.

The need for a matrix stepsize

Optimal learning gain for stochastic gradient algorithms

Mean trajectories of an ensemble of noisy single-realisation gradient descent paths for correlated data.



The LMS path is locally optimal but globally slower converging than the optimal path (see DM *at al.*, IEEE Sig. Proc. Magazine, March 2015).

The proportionate NLMS (PNLMS) algorithm

(here g_i are based on thresholding filter weights, to promote zero weights)

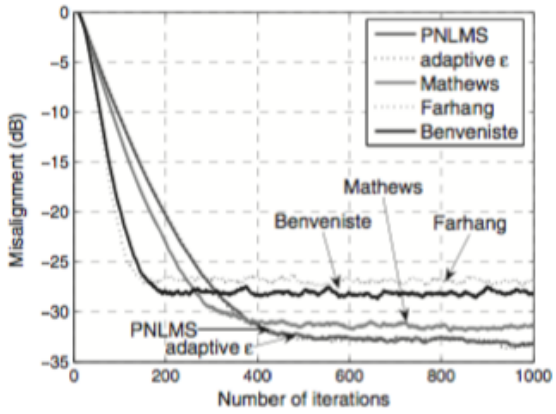
One such, ad hoc, matrix step-size is used in the PNLMS algorithm for system identification for long channels (echo cancellation), given by

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \frac{\mathbf{G}(k)e(k)\mathbf{x}(k)}{\|\mathbf{x}(k)\|_2^2},$$

where

$$\mathbf{G}(k) = \text{diag}[g_1(k), \ldots, g_N(k)].$$

The diagonal elements of $\mathbf{G}(k)$ define the proportionate amounts that each coefficient is updated



Performance comparison of the GASS PNLMS algorithms with the standard PNLMS for $\mu = 0.1$

Performance when identifying a 100-tap channel, with only 4 non-zero coefficients.

From LMS to Kalman Filter

Kalman filter $\Leftrightarrow$ LMS with an optimal learning "gain"

Recall the LMS algorithm: $\mathbf{w}_k = \mathbf{w}_{k-1} + \mu_k \mathbf{x}_k(d_k - \mathbf{w}_{k-1}^T \mathbf{x}_k)$ for the desired signal given by $d_k = \mathbf{x}_k^T \mathbf{w}^\circ + q_k$.

- Introduce more degrees of freedom by replacing the scalar step-size, μ_k, with a positive definite learning gain matrix, $\mathbf{G}_k$, to give

$$\mathbf{w}_k = \mathbf{w}_{k-1} + \mathbf{G}_k \mathbf{x}_k(d_k - \mathbf{w}_{k-1}^T \mathbf{x}_k).$$

- Now, we are able to control both the **magnitude and direction** of the gradient descent adaptation.
- To find $\mathbf{G}_k$, minimise the mean square deviation (MSD) in the form:

$$J_k = E\{\|\mathbf{w}^\circ - \mathbf{w}_k\|^2 \ = E\{\| \mathbf{v}_k \|^2\}$$

$$\partial J_k / \partial \mathbf{G}_k = \mathbf{0} \implies \mathbf{G}_k = \frac{\mathbf{P}_{k-1}}{\mathbf{x}_k^T \mathbf{P}_{k-1} \mathbf{x}_k + \sigma_q^2}$$

with $\mathbf{P}_k = E\{\mathbf{v}_k \mathbf{v}_k^T\} = \mathbf{P}_{k-1} - \mathbf{G}_k \mathbf{x}_k \mathbf{x}_k^T \mathbf{P}_{k-1}$ and $\sigma_q^2 = E\{q_k^2\}$.

- The optimal gain vector $\mathbf{g}_k \triangleq \mathbf{G}_k \mathbf{x}_k$ is known as the **Kalman gain.**

A general framework for multiple constraints

We start from the general L_p–norm family of penalty functions, defined by

$$L_p(\mathbf{x}) = \| \mathbf{x} \|_p = \Big(\sum_{n=0}^{N-1} |x(n)|^p \Big)^{\frac{1}{p}}$$

Then, the typical optimisations with a penalty are

Ridge regression: $$\mathcal{J}_k(\mathbf{w}) = \underbrace{(d_k - \mathbf{w}_k^T \mathbf{x}_k)^2}_{\text{Cost}} + \underbrace{\lambda_1 \|\mathbf{w}_k\|_2^2}_{\text{Penalty}} = e_k^2 + \lambda_1 \mathbf{w}_k^T \mathbf{w}_k$$

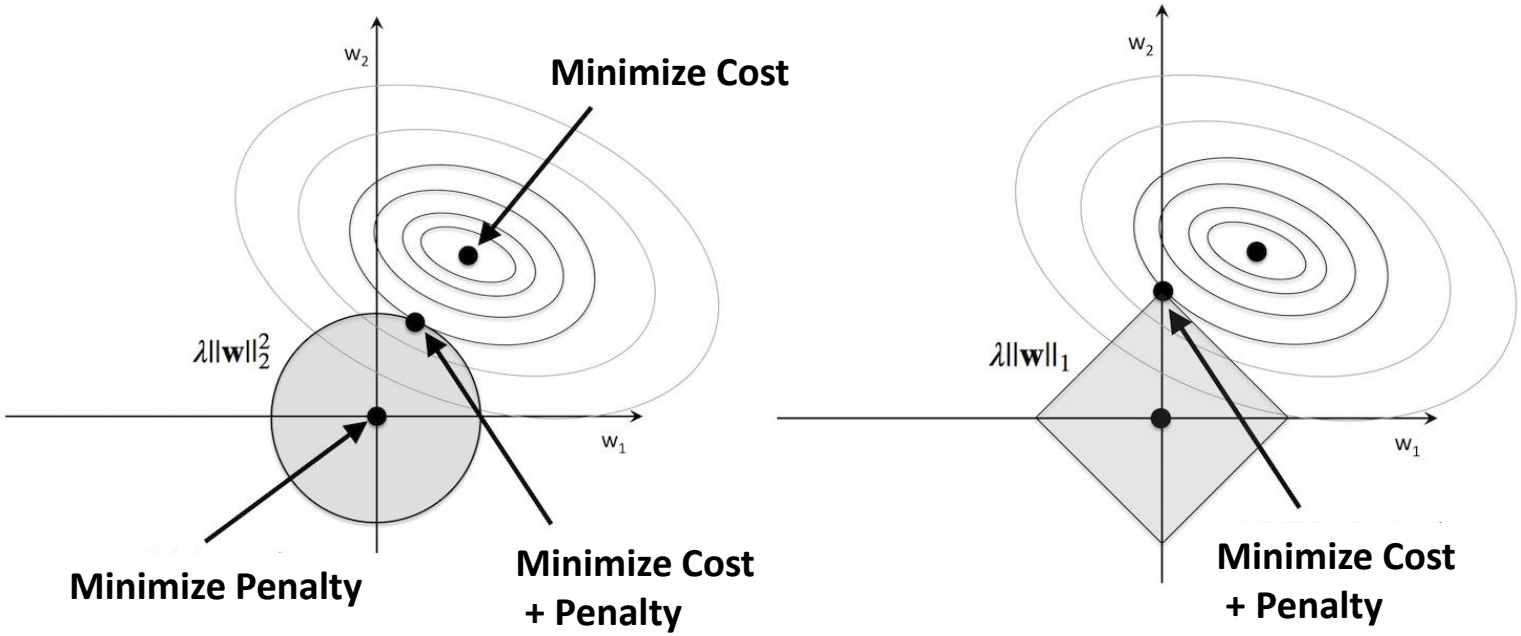
LASSO (sparsity promoting): $$\mathcal{J}_k(\mathbf{w}) = \underbrace{(d_k - \mathbf{w}_k^T \mathbf{x}_k)^2}_{\text{Cost}} + \underbrace{\lambda_2 \|\mathbf{w}_k\|_1}_{\text{Penalty}}$$

- Ridge regression penalises for large weights
- Least absolute shrinkage and selection operator (LASSO) enforces insignificant weights to go to zero, and in this way aids **interpretability**

Exploiting regularisation and sparsity

Regularised LMS cost function with ℓ_2-norm (Tikhonov) or ℓ_1-norm (Lasso)

$\mathcal{J}_k(\mathbf{w}) = \underbrace{(d_k - \mathbf{w}^T\mathbf{x}_k)^2}_{\text{Cost}} + \underbrace{\lambda\|\mathbf{w}\|_p}_{\text{Penalty}}$ with Regularisation Parameter: λ.



Tikhonov: $\ell = 2$

Lasso: $\ell = 1$

(Figure credit: Mlxtend, S. Raschka, 2016)

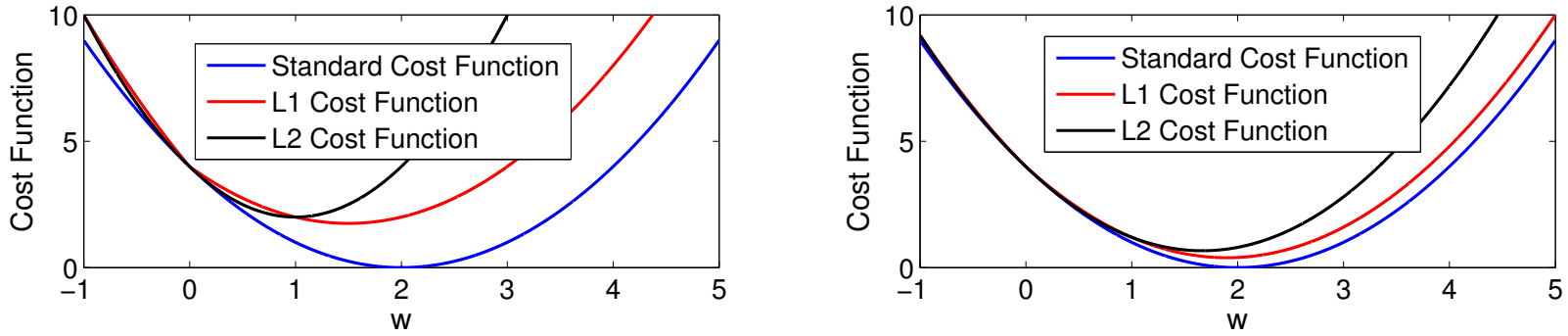
Sparsity promoting

For $\lambda \neq 0$ the **minima of the regularised cost functions do not correspond** to the squared error cost.

Regularised LMS aims to avoid overfitting by a penalty for complexity

The effect of the regularisation parameter

Consider a simple case of: $d_k = w^\circ x_k + q_k$ with $w_\circ = 2$.



$\lambda = 1$ $\lambda = 0.2$

Comparison between the ℓ_1- and ℓ_2-norm regularised cost functions

- For $\lambda \neq 0$ the minima of the regularised cost functions do not correspond to the optimum weight $w^\circ = 2$
- The regularisation parameter λ controls the **trade-off** between minimising the cost (squared error) and penalty.
- **Smaller values of** λ correspond to favours **minimising the cost** (squared error) while **large** λ **values** strongly **penalises the norm of** $\mathbf{w}$.

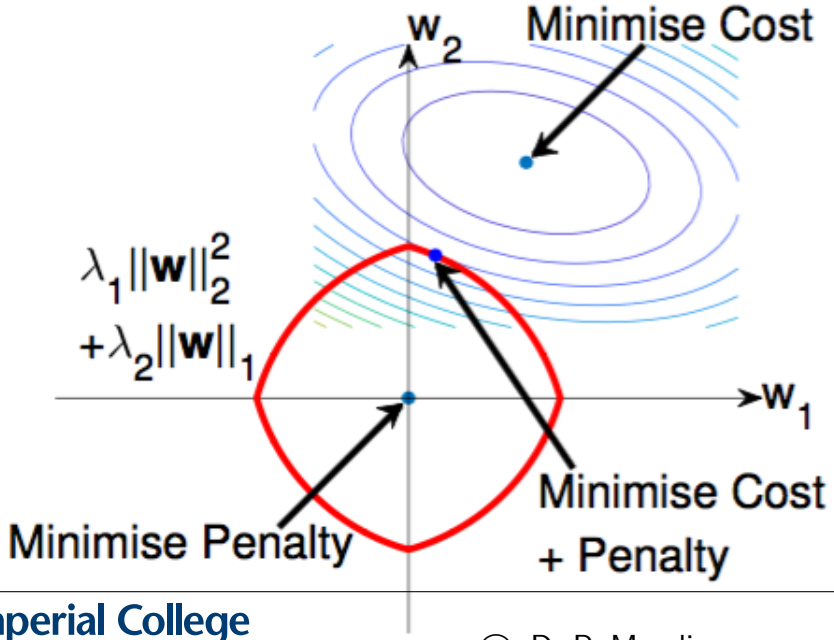
Elastic net: Getting the best of both worlds

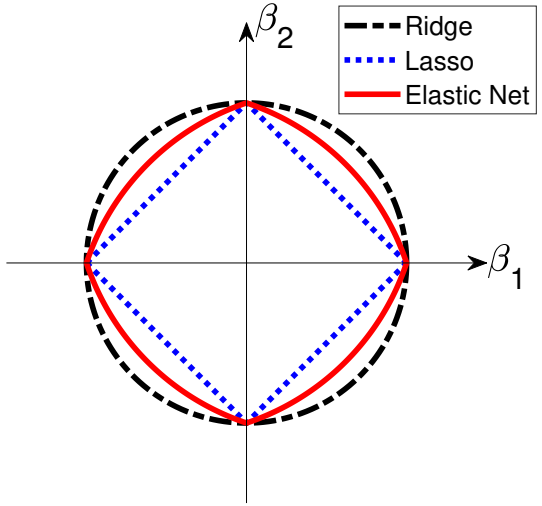
Elastic net aims to combine the benefits of Ridge Regression (penalising for large weights) and LASSO (enforcing sparsity), and employs the cost

Elastic net:
$$\mathcal{J}_k(\mathbf{w}) = \underbrace{(d_k - \mathbf{w}_k^T \mathbf{x}_k)^2}_{\text{Cost}} + \underbrace{\lambda_1 \|\mathbf{w}_k\|_2^2}_{\text{L2 penalty}} + \underbrace{\lambda_2 \|\mathbf{w}_k\|_1}_{\text{L1 penalty}}$$

In other words, Elastic Net aims to minimize

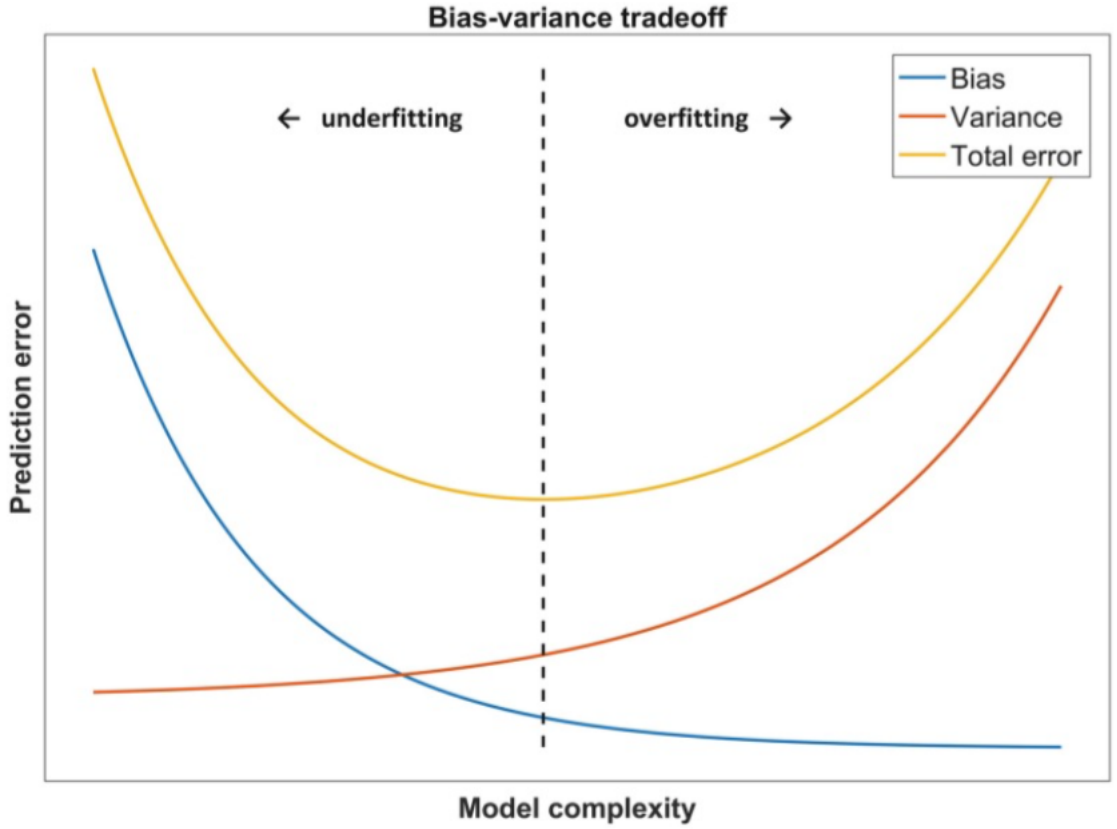
$$\mathcal{J}(\mathbf{w}(k)) = e^2(k) + \lambda_1 \sum_{i=0}^{N-1} w_i^2(k) + \lambda_2 \sum_{i=0}^{N-1} |w_i(k)|$$





Over– and under–fitting and the Bias–Variance tradeoff

☞ Recall the expression for Mean Square Error: $\text{MSE} = \text{Bias}^2 + \text{variance}$



Bias: An error due to (erroneous) assumptions in the learning algorithm

Variance: An error from sensitivity to fluctuations in the training set

Let us get more specific: A framework to involve sparsity into Adaptive Learning Systems

The optimisation criterion becomes:

$$\begin{aligned}
\hat{\mathbf{w}} &= arg\min_{\mathbf{w}} \left\{ E\left[\parallel y(n) - \mathbf{w}^T(n)\mathbf{x}(n) \parallel_2^2 \right] + \lambda \parallel \mathbf{w}(n) \parallel_1 \right\} \\
&\equiv \min_{\mathbf{w}} \left\{ E\left[\parallel y(n) - \mathbf{w}^T(n)\mathbf{x}(n) \parallel_2^2 \right] \right\} \quad \text{s.t.} \quad \parallel \mathbf{w}(n) \parallel_1 \leq \rho \\
&\equiv \min_{\mathbf{w}} \left\{ \parallel \mathbf{w}(n) \parallel_1 \right\} \quad \text{s.t.} \quad E\left[\parallel y(n) - \mathbf{w}^T(n)\mathbf{x}(n) \parallel_2^2 \right] \leq \varepsilon
\end{aligned}$$

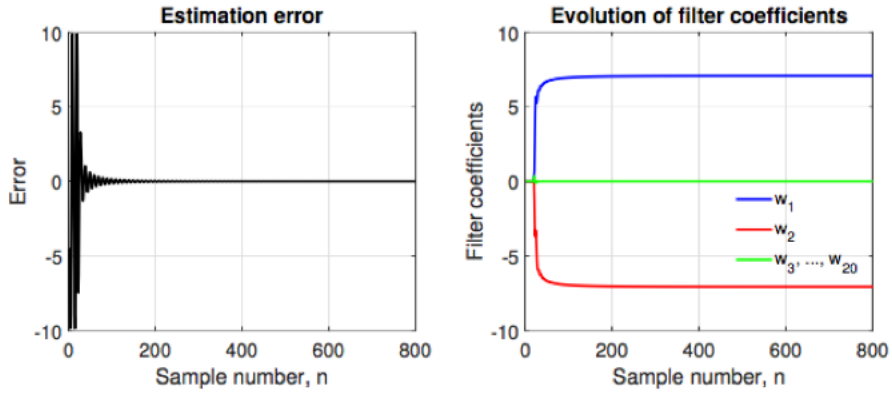
For LMS-type stochastic gradient descent, for a filter with length L we then have

$$\hat{\mathbf{w}} = arg\min_{\mathbf{w}} \left\{ \frac{1}{2}e^2(n) + \lambda \parallel \mathbf{w}(n) \parallel_1 \right\} = arg\min_{\mathbf{w}} \left\{ \frac{1}{2}e^2(n) + \lambda \sum_{i=1}^{L} |w_i(n)| \right\}$$

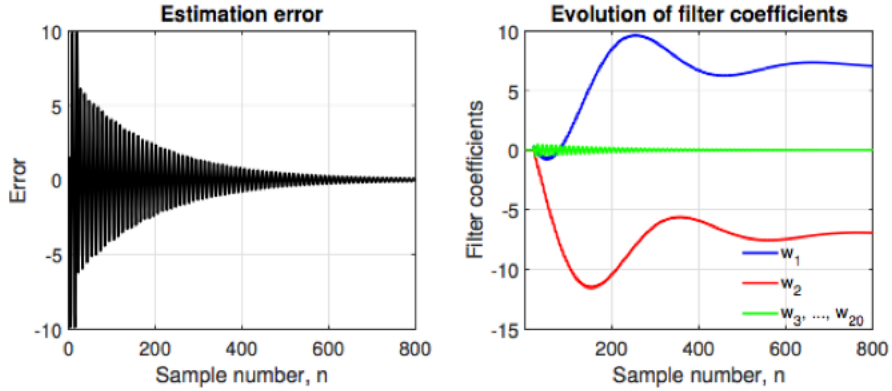
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) - \mu\lambda sign(\mathbf{w}(n))$$

Other constraints can also be included (e.g. smoothness)

Performance of standard RLS and LMS on a sparse problem



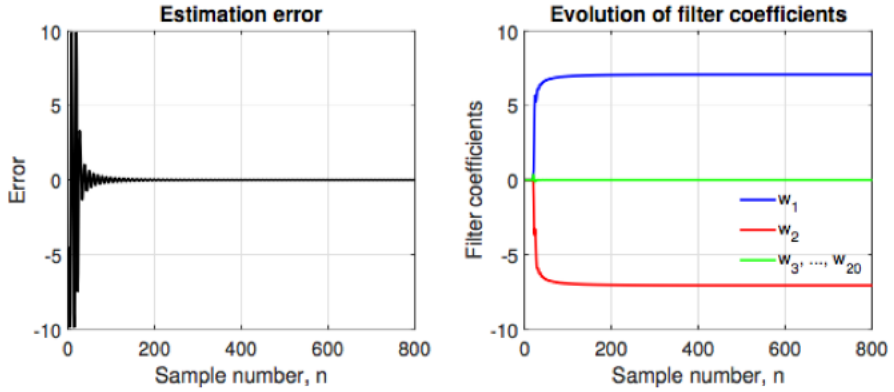
(a) Nonsparse RLS



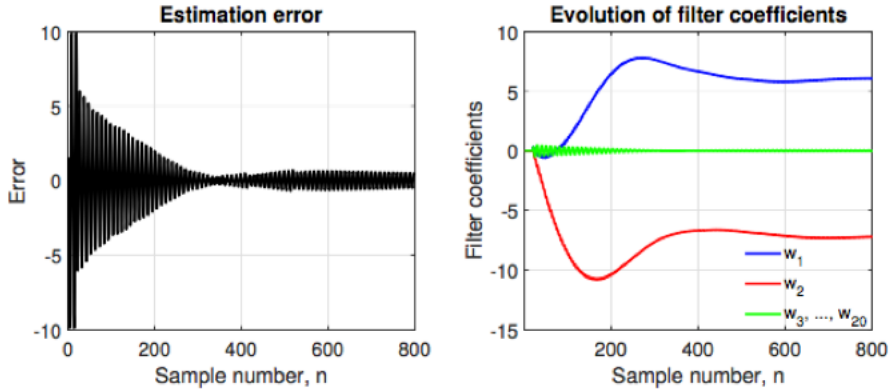
(b) Nonsparse LMS

Figure: Standard RLS and LMS for estimation of $y(n) = 10\cos(0.2\pi n + \pi/4)$ based on $\mathbf{x}(n) = [\sin(0.2\pi n); \cos(0.2\pi n); 1; 1; ..., 1]$, where the length of $\mathbf{x}$ is 20. For both algorithms, the initial weight is zero. For the RLS, the initial error covariance matrix is identity matrix, and the forgetting factor is 0.99. The step size of LMS is 0.05.

Performance of sparse RLS and LMS on a sparse problem



(a) Sparse RLS



(b) Sparse LMS

Figure: Sparse RLS and LMS for estimation of $y(n) = 10\cos(0.2\pi n + \pi/4)$ based on $\mathbf{x}(n) = [\sin(0.2\pi n); \cos(0.2\pi n); 1; 1; ...; 1]$, where the length of $\mathbf{x}$ is 20. For both algorithms, the sparsity coefficient is 0.2, and the initial weight is zero. For the RLS, the initial error covariance matrix is identity matrix, and the forgetting factor is 0.99. The step size of LMS is 0.05.

Performance of sparse RLS and LMS ↬ Misalignment

Recall that the misalignment $\mathcal{M} = J_{ex}(\infty)/J_{min}$

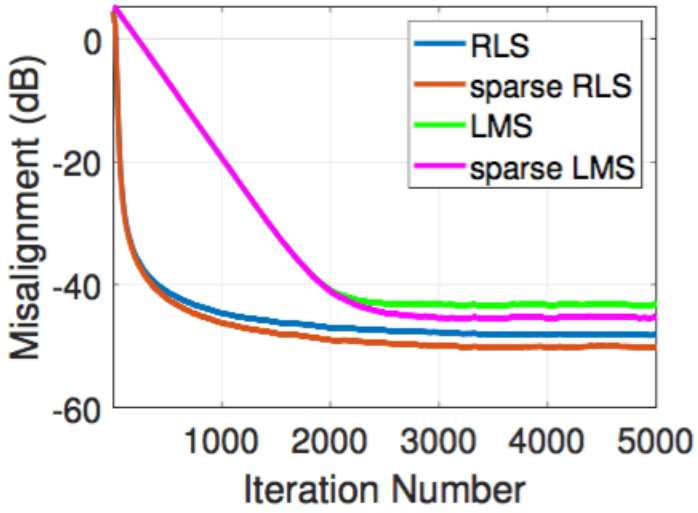
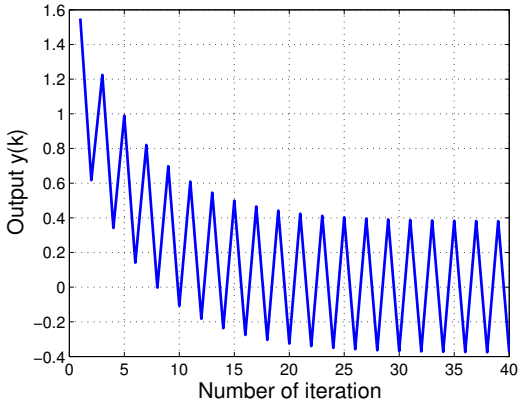
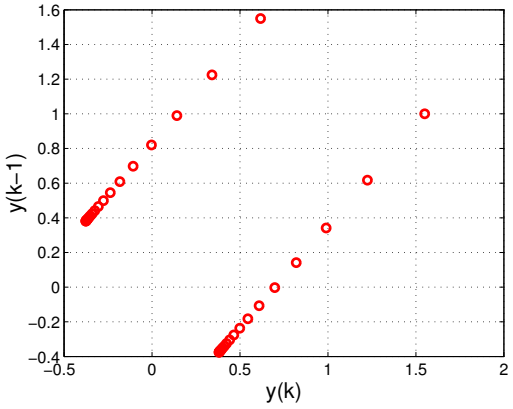


Figure: Average misalignment curves over 200 trials of four adaptive filtering algorithms for estimating the sparse coefficients of the system $y(n) = \sum_{i=0}^{29} w_i x(n-i) + q(n)$, where $x(n)$ and $q(n)$ are white Gaussian, $q(n)$ is noise, and only three of the 30 weights, $w_i, i = 0, \ldots, 29$, are nonzero. For all the algorithms, the initial weight is zero. For the RLS and sparse RLS, the initial error covariance matrix is identity matrix, the forgetting factor is 0.999. The sparsity coefficient of sparse RLS is 0.8. For the LMS and sparse LMS, the step size is 0.003. The sparsity coefficient of sparse LMS is 0.0015. The curves show that the misalignment of sparse RLS/LMS is 2dB lower than that of the standard RLS/LMS.

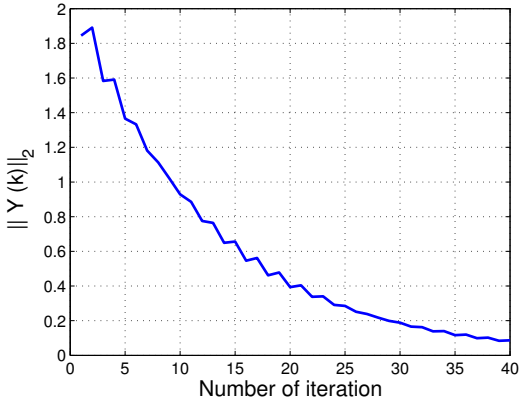
The need for state space models: Relaxation of autonomous systems



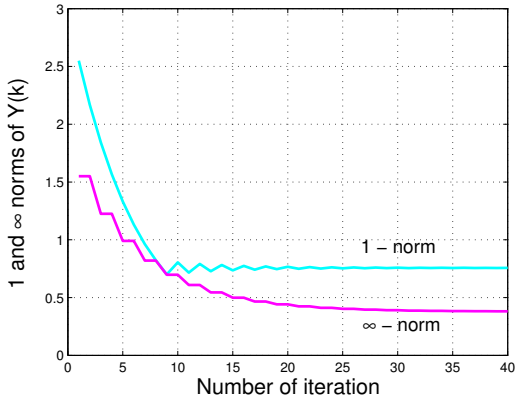
(a) Output $y(k)$



(b) Oscillatory limit cycle in state space



(c) Convergence in 2–norm



(d) Convergence in 1– and ∞–norm

- **We converge in the norm, but not in state space**
- The output oscillates, but the 'quantitative' performance measure shows convergence in the norm ($\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_\infty$)

The Need for state space estimation: State transition matrix

An autonomous system which is described by (the driving input is cut off)

$$y(k+1) = \mathbf{a}^T(k)\mathbf{y}(k) = a_1(k)y(k) + \cdots + a_N(k)y(k-N+1) \qquad (1)$$

and should ideally represent a relaxation. The vector–matrix form is

$$\begin{bmatrix} y(k+1) \\ y(k) \\ \vdots \\ y(k-N+1) \end{bmatrix} = \underbrace{\begin{bmatrix} a_1(k) & a_2(k) & \cdots & a_N(k) \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}}_{\text{companion matrix (Frobenius)}} \begin{bmatrix} y(k) \\ y(k-1) \\ \vdots \\ y(k-N) \end{bmatrix}$$

$$or$$

$$\mathbf{Y}(k+1) = \mathbf{A}(k)\mathbf{Y}(k)$$

$$with$$

$$y(k+1) = [1 \;\; 0 \;\; \cdots \;\; 0]\, \mathbf{Y}(k+1) \qquad (2)$$

Start iteration from any initial condition.

Intuitive example ↬ Towards the Kalman filter

Conisider a quantity x which is measured twice, the measurements are $x_1 \sim \mathcal{N}(\bar{x}_1, \sigma_1^2)$ and $x_2 \sim \mathcal{N}(\bar{x}_2, \sigma_2^2)$.

The two measurements are combined to give an estimate of the length:-

$$\hat{x} = wx_1 + (1-w)x_2$$

Due to the Gaussianity of the distributions of x_1 and x_2, the variance

$$\hat{\sigma}^2 = w^2\sigma_1^2 + (1-w)^2\sigma_2^2$$

To find a weight w which minimises the uncertainty $\hat{\sigma}^2$ apply $\frac{\partial}{\partial w}$, to have

$$w_{opt} = \frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2} \quad \Rightarrow \hat{x} = \frac{\sigma_2^2}{\sigma_1^2+\sigma_2^2}\,x_1 + \frac{\sigma_1^2}{\sigma_1^2+\sigma_2^2}\,x_2; \quad \hat{\sigma}^2 = \frac{\sigma_1^2\,\sigma_2^2}{\sigma_1^2+\sigma_2^2}$$

☞ **The estimate $\hat{x}$ minimises the sum of the distances to x_1 and x_2, weighted by the respective standard deviations.**

Intuitive example ↬ Towards the Kalman filter, cont.

Suppose the two measurements become available sequentially.

- Step 1: x_1 available $\Rightarrow \hat{x}_1 = x_1,\ \hat{\sigma}_1^2 = \sigma_1^2$
- Step 2: x_2 available $\Rightarrow$ rewrite the estimator $\hat{x}$ in a recursive form

$$\begin{aligned} \hat{x}_2 &= \hat{x}_1 + \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}\,(x_2 - \hat{x}_1) \\ \hat{\sigma}_2^2 &= \left(1 - \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}\right)\hat{\sigma}_1^2 \end{aligned}$$

☞ **Quantity $x_2 - \hat{x}_1$ represents the new information received at Time Step 2, and is called the innovation.**

Quantity $K = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_1^2 + \sigma_2^2}$ is called the update gain. Therefore:-

$$\begin{aligned} \hat{x}_2 &= \hat{x}_1 + K(x_2 - \hat{x}_1) \\ \hat{\sigma}_2^2 &= (1 - K)\hat{\sigma}_1^2 \end{aligned}$$

Kalman Filters – More specifically

- Generalisation of Wiener filter which can handle vector signals and noise with time-varying statistics

- It is a sequential MMSE estimator of a signal embedded in noise, where the signal is characterised by a dynamical or state model

- If the signal and noise are jointly Gaussian then the Kalman filter is an optimal MMSE estimator

- Signal is assumed to evolve according to a dynamical or state model of the form

$$s[n] = \underbrace{as[n-1]}_{state} + u[n] \quad n \geq 0$$

$u[n]$ - driving or excitation noise with variance σ_u^2

Initial value $s[-1] \sim \mathcal{N}(\mu_s, \sigma_s^2)$ independent of $u[n]$

The case of an AR(1) process

- The mean and variance can be expressed recursively

$$E\{s[n]\} = aE\{s[n-1]\} + E\{u[n]\}$$

$$= aE\{s[n-1]\}$$

$$Var(s[n]) = a^2 Var(s[n-1]) + \sigma_\mu^2$$

- N.B. $\lim_{n\to\infty} E\{s[n]\} \to 0$

- Generalising to a p-th order Gauss-Markov process

$$s[n] = \sum_{k-1}^{p} a[k]s[n-k] + u[n]$$

For the AR(p) case $\hookrightarrow$ The vector Gauss–Markov model

- State vector

$$\begin{bmatrix} s[n-p+1] \\ s[n-p+2] \\ \vdots \\ s[n-1] \\ s[n] \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ a(p) & a(p-1) & \dots & a(1) \end{bmatrix}}_{State\ transition\ matrix\ A} \begin{bmatrix} s[n-p] \\ s[n-p+1] \\ \vdots \\ s[n-2] \\ s[n-1] \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}}_{B} u[n]$$

- $u[n]$ can be a $r \times 1$ vector - models a vector signal output with vector input

- Notice the "companion matrix" structure

Kalman filter: Summary

- The Kalman filter becomes
  - PREDICTION

$$\hat{s}[n|n-1] = a\hat{s}[n-1|n-1]$$

  - MINIMUM PREDICTION MSE

$$M[n|n-1] = a^2 M[n-1|n-1] + \sigma_\mu^2$$

  - Kalman Gain

$$K[n] = \frac{M[n|n-1]}{\sigma_n^2 + M[n|n-1]}$$

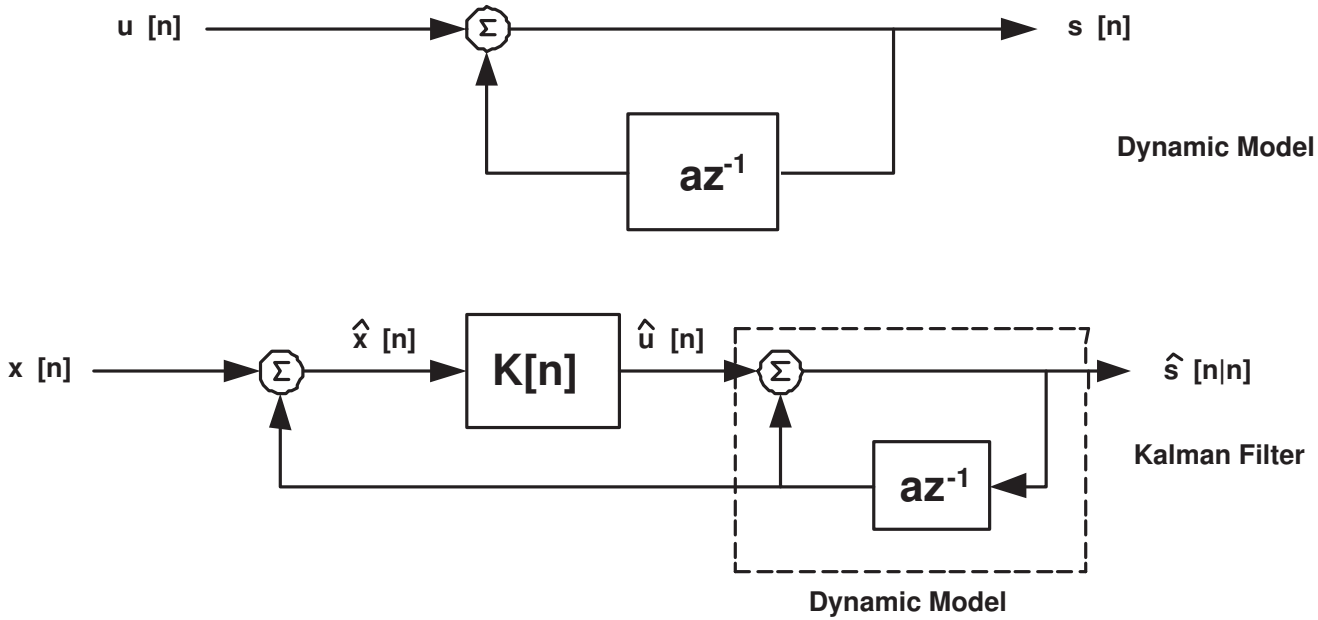
  - Correction

$$\hat{s}[n|n] = \hat{s}[n|n-1] + K[n]\left(x[n] - \hat{s}[n|n-1]\right)$$

Kalman filter: Summary, contd.

- Minimum MSE

$$M[n|n] = (1 - K[n])M[n|n-1]$$



Equivalence between Kalman filter and adaptive stepsize LMS

Algorithm 2: A hybrid Kalman-LMS algorithm.

At each time instant $k > 0$, based on measurements $\{d_k, \boldsymbol{x}_k\}$

1) Compute the confidence level (regularisation parameter):

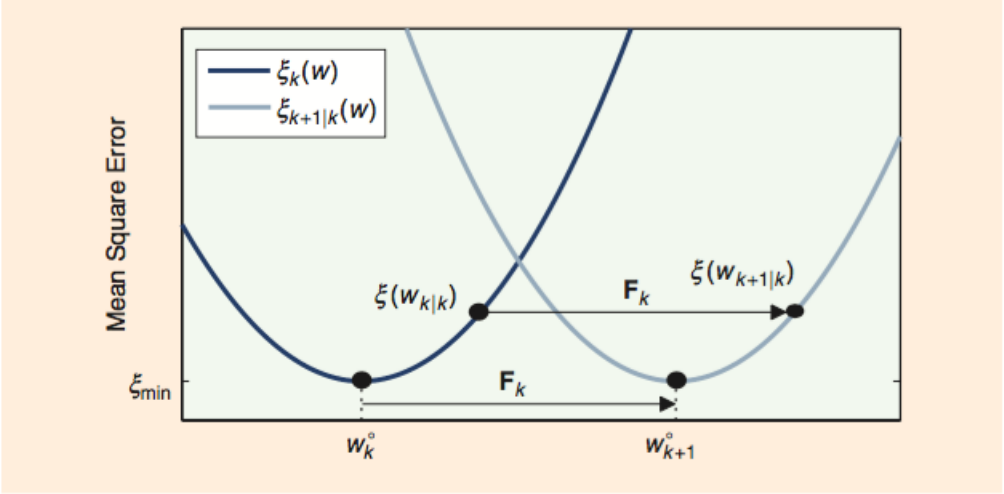
$$\varepsilon_k = \sigma_n^2 / \sigma_{P,k-1}^2$$

2) Update the weight vector estimate:

$$\boldsymbol{w}_k = \boldsymbol{w}_{k-1} + \frac{\boldsymbol{x}_k}{\|\boldsymbol{x}_k\|^2 + \varepsilon_k}(d_k - \boldsymbol{x}_k^T \boldsymbol{w}_{k-1})$$

3) Update the weight error variance:

$$\sigma_{P,k}^2 = \sigma_{P,k-1}^2 - \frac{\|\boldsymbol{x}_k\|^2}{M(\|\boldsymbol{x}_k\|^2 + \varepsilon_k)}\sigma_{P,k-1}^2$$



[FIG2] The time-varying state transition in (22a) results in a time-varying MSE surface. For clarity, the figure considers a scalar case without state noise. Within the Kalman filter, the prediction step in (23b) preserves the relative position of $w_{k+1|k}$ with respect to the evolved true state, w_{k+1}°.

Some notions from estimation theory

- **System:** Actual underlying physics[1] that generate the data;
- **Model:** A mathematical description of the system;
- Many variations of mathematical models can be postulated on the basis of datasets collected from observations of a system, and their suitability assessed by various performance metrics;
- Since it is not possible to characterise nonlinear systems by their impulse response, we may resort to less general models, e.g. polynomial filters
- Some of the most frequently used polynomial filters are based upon
  - Volterra series, a nonlinear analogue of the linear impulse response
  - threshold autoregressive models (TAR)
  - Hammerstein and Wiener models

[1] Technically, the notions of *system* and *process* are equivalent.

Basic modes of modelling

- **Parametric** modelling assumes a fixed structure for the model. The model identification problem then simplifies to estimating a finite set of parameters of this fixed model. An example of this technique is the broad class of ARIMA/NARMA models.

- **Nonparametric** modelling seeks a particular model structure from the input data. The actual model is not known beforehand.
  We look for a model in the form of $y(k) = f\left(x(k)\right)$ without knowing the function $f(\cdot)$.

- **Semiparametric** modelling is the combination of the above. Part of the model structure is completely specified and known beforehand, whereas the other part of the model is either not known or loosely specified.

Neural networks, especially recurrent neural networks, can be employed within estimators of all of the above classes of models. Closely related to the above concepts are white, grey, and black box modelling techniques.

White–, Grey–, and Black–Box modelling

- The idea is to distinguish between three levels of prior knowledge, which have been "colour–coded"

- **White box** $\looparrowright$ the underlying physical process has to be understood (planetary motions, Kepler's laws, gravitation)

- **Black box** $\looparrowright$ no previous knowledge about data production system. The aim is to find a function F that approximates[2] the process y based on the previous observation of the process y_{PAST} and input u, as

$$y = F\left(y_{PAST}, u\right)$$

- **Grey box** $\looparrowright$ obtained from black box modelling if some information about the system is known **a priori**

[2]The downside is that it is generally not possible to learn about te true physical process that generates the data, especially if a linear model is used

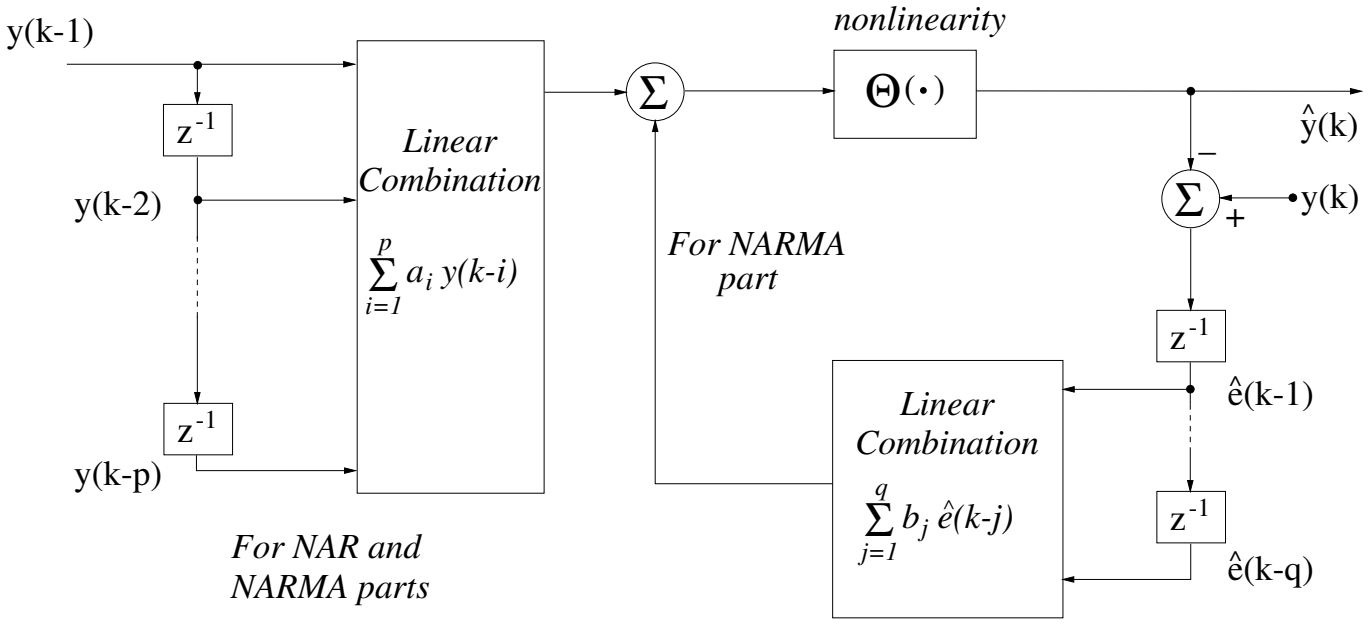
From a Black Box to a White Box

Recurrent neural networks as NARMA(p,q) models

A **NARMA(p,q)** model is given by

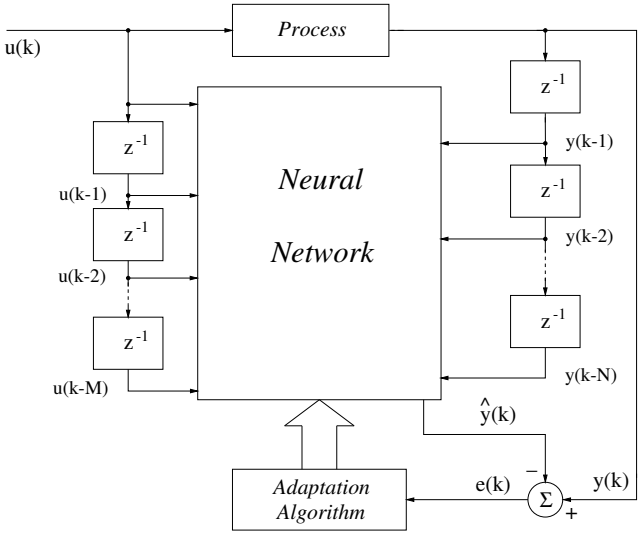
$$\hat{y}(k) = \Theta\left(\sum_{i=1}^{p} a_i y(k-i) + \sum_{j=1}^{q} b_j \hat{e}(k-j)\right) \qquad \Theta(\cdot) - nonlinear$$

where the residuals $\hat{e}(k-j) = y(k-j) - \hat{y}(k-j)$, $j = 1, 2, \ldots, q$.

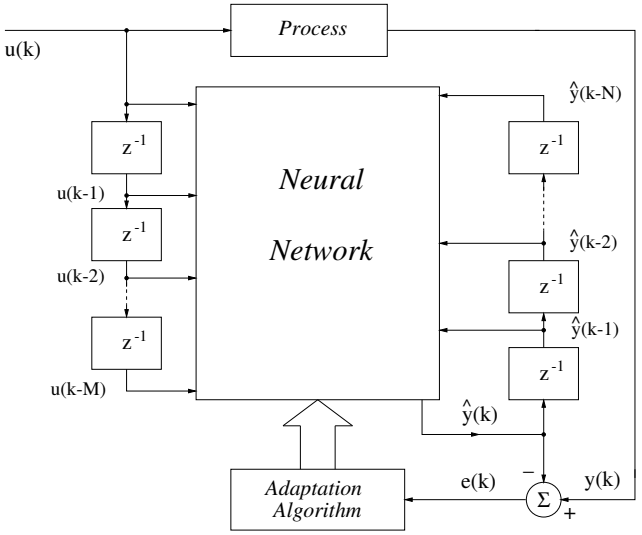


Learning strategies

Teacher forced



Supervised



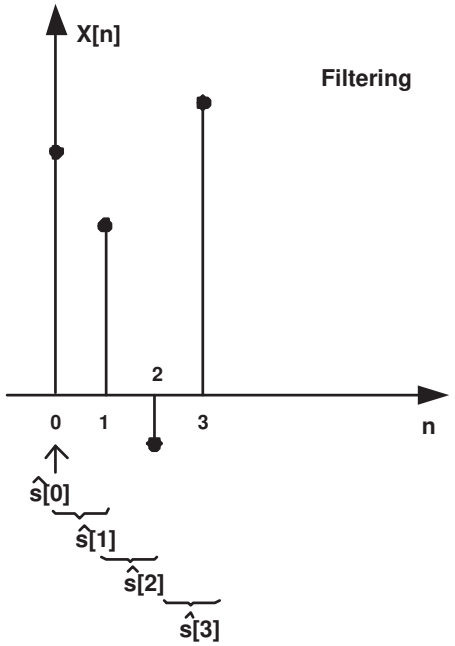
- **Teacher forced** ↬ introduces bias into estimation

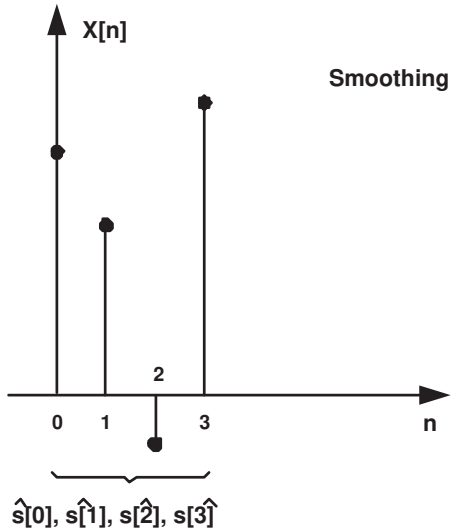
$$\hat{y}(k) = f\left(u(k), \ldots, u(k-M), y(k-1), \ldots, y(k-N)\right)$$

- **Supervised** ↬ unbiased (suitable for recursive estimation)

$$\hat{y}(k) = f\left(u(k), \ldots, u(k-M), \hat{y}(k-1), \ldots, \hat{y}(k-N)\right)$$

Problems to be solved: Filtering and Smoothing

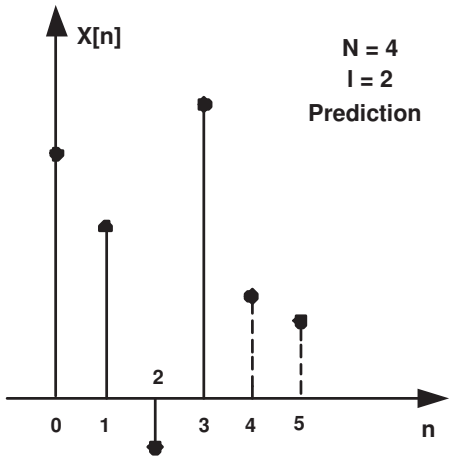




$s[n]$ to be estimated based on
$x[m] = s[m] + w[m]$ $m = 0, 1, ..., n$
Filter signal from noise, based on current and past data, i.e. casual filtering

$s[n]$ to be estimated based on entire dataset $\{x[0], x[1], ..., x[N-1]\}$.
Requires all data to be collected

Problems to be solved: Prediction



Estimates $x[N-1+l]$ for l a positive integer based on $\{x[0], x[1], ..., x[N-1]\}$ "l-step" forward prediction.

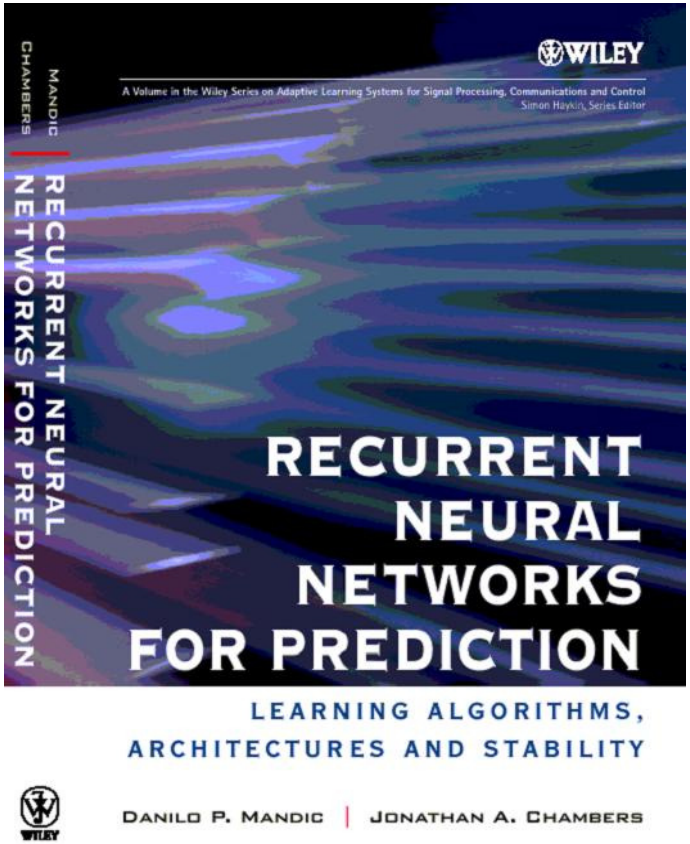
Prediction horizon: How far ahead can we predict.

Conclusions

- Adaptive processing of signals observed through possibly nonlinear sensors, and at multiple scales
- The recursive least squares (RLS), a 'deterministic' solution to the estimation of streaming data
- A general form of learning algorithms, optimality of the gain factor
- Black box, grey box and white box modelling
- Parametric, non-parametric, and semi-parametric modelling
- How can we 'make sense' of similar data classes which are observed in different lighting conditions, shades, and viewing angles
- Perceptron, neural network, recurrent perceptron
- Neural networks as 'nonlinear ARMA', that is, NARMA systems
- State space models and Kalman filter – relation with adaptive stepsize LMS
- This promises enhanced practical solutions in a variety of applications

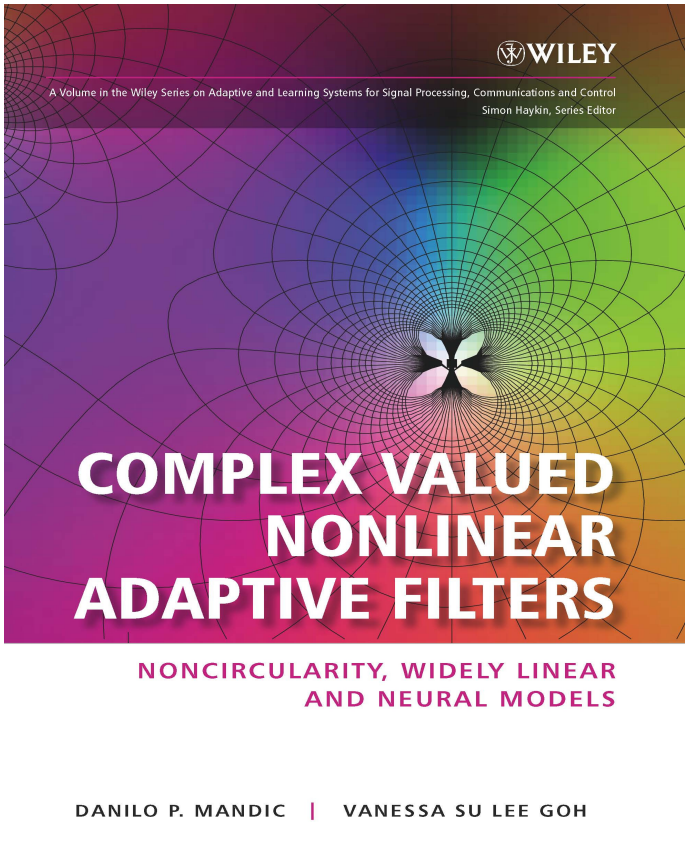
Additional reading

D. Mandic & J. Chambers (*RNNs for Prediction*, Wiley 2001).



neural network architectures

D. Mandic and S. Goh (*Complex Adaptive Filters*, Wiley 2009).



real, complex, and neural adaptive filters

Some back–up material

Appendix: Inverse of a matrix

We need to calculate those inverses for our main models

The structure of an inverse matrix is also important!

- A symmetric matrix has a symmetric inverse
- A **Toeplitz** matrix has a persymmetric inverse (symmetric about the cross diagonal)
- A Hankel matrix has a symmetric inverse (useful in harmonic retrieval)

A useful formula for matrix inversion is **Woodbury's identity** (matrix inversion Lemma)

$$\left(\mathbf{A}+\mathbf{U}\mathbf{V}^{\mathbf{T}}\right)^{-1}=\mathbf{A}^{-1}-\left[\mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I}+\mathbf{V}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^{\mathbf{T}}\mathbf{A}^{-1}\right]$$

If $\mathbf{u}$ and $\mathbf{v}$ are vectors, then this simplifies into

$$(\mathbf{A}+\mathbf{u}\mathbf{v}^{T})^{-1}=\mathbf{A}^{-1}-\frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^{T}\mathbf{A}^{-1}}{1+\mathbf{v}^{T}\mathbf{A}^{-1}\mathbf{u}}$$

which will be important in the derivation of adaptive algorithms.

In a special case $\mathbf{A}=\mathbf{I}$ and we have

$$(\mathbf{I}+\mathbf{u}\mathbf{v}^{T})^{-1}=\mathbf{I}-\frac{\mathbf{u}\mathbf{v}^{T}}{1+\mathbf{v}^{T}\mathbf{u}}$$

Ways to compute a matrix inverse

while direct, this technique is computationally wasteful, $N^3 + 2N^2 + N$ multiplic.

There are many ways (Neumann series, geometric series). For $|a| < 1$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}, \qquad \text{now swap} \quad \mathrm{b} = 1 - \mathrm{a} \quad \text{to get} \quad \sum_{\mathrm{i}=1}^{\infty} (1-\mathrm{b})^{\mathrm{i}} = \frac{1}{\mathrm{b}}$$

☞ **we have found the inverse of b via a series in $1 - b$.**

Generalization to matrices: Consider a nonsingular matrix $\mathbf{A}$, to yield

$$\lim_{i=\to\infty} (\mathbf{I} - \mathbf{A})^i = \mathbf{0} \quad \Rightarrow \quad \mathbf{A}^{-1} = \sum_{i=0}^{\infty} (\mathbf{I} - \mathbf{A})^i$$

Then, for an invertible matrix $\mathbf{X}$, and its inverse $\mathbf{A}$, we can write

$$\lim_{i\to\infty} \left(\mathbf{I}-\mathbf{X}^{-1}\mathbf{A}\right)^i = \lim_{i\to\infty} \left(\mathbf{I}-\mathbf{A}\mathbf{X}^{-1}\right)^i = \mathbf{0} \quad \Rightarrow \quad \mathbf{A}^{-1} = \sum_{i=0}^{\infty} \Big(\mathbf{X}^{-1}\big(\mathbf{X}-\mathbf{A}\big)\Big)^i \mathbf{X}^{-1}$$

If $(\mathbf{A} - \mathbf{X})$ has rank 1, we finally have: $\mathbf{A}^{-1} = \mathbf{X}^{-1} + \dfrac{\mathbf{X}^{-1}\big(\mathbf{X}-\mathbf{A}\big)\mathbf{X}^{-1}}{1-\mathrm{tr}\Big(\mathbf{X}^{-1}\big(\mathbf{X}-\mathbf{A}\big)\Big)}$

Example: Matrix inversion lemma ↬ another approach

Construct an 'augmented' matrix $[A, B; C, D]$ and its inverse $[E, F; G, H]$, so that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Consider the following two products

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+HC & CF+DH \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

and

$$\begin{bmatrix} E & F \\ G & H \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} EA+FC & EB+FD \\ GA+HC & GB+HD \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

Combine the corresponding terms to get another form

$$\left(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{C}\mathbf{A}^{-1}$$

Appendix: The RLS algorithms and its variants

Accelerated LMS:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{C}(n)\mathbf{x}(n), \quad \mathbf{C} \text{ a chosen matrix}$$

Normalised LMS:

$$\mu(n) = \frac{\mu}{\varepsilon + \mathbf{x}^T(n)\mathbf{x}(n)}, \quad 0 < \mu < 2$$

Exponentially weighted RLS (RLS with the forgetting factor)

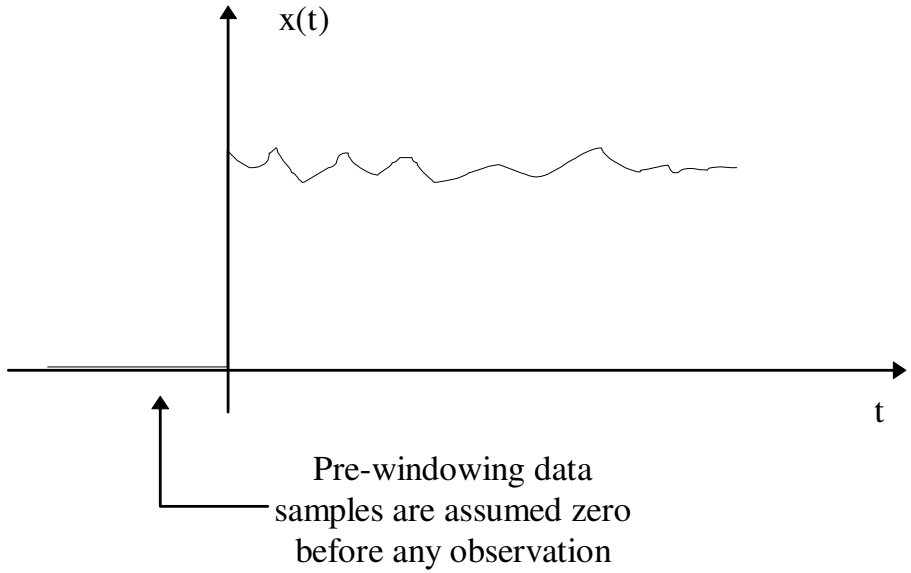
$$J(n) = \sum_{i=0}^{n} \lambda^{n-i} \big| d(i) - y(i) \big|^2, \quad \lambda \text{ is a forgetting factor}$$

- Forgetting factor $\lambda \in (0, 1]$, but typically > 0.95
- The forgetting factor introduces an effective window length of $\frac{1}{1-\lambda}$

Appendix: Pre–windowing

- To initialise RLS we need to make some assumptions about the input data

- The most common one is that of prewindowing, that is $\forall k < 0, x(k) = 0$



Appendix: Complete RLS algorithm

- Initialisation $\mathbf{w}(0) = \mathbf{0}$, $\mathbf{P}(0) = \delta\mathbf{I}$, $\mathbf{P}(k) = \mathbf{R}^{-1}$

- For each k

$$\begin{aligned}
\gamma(k+1) &= \mathbf{x}^T(k+1)\mathbf{P}(k)\mathbf{x}(k+1) \\
\mathbf{k}(k+1) &= \frac{\mathbf{P}(k)\mathbf{x}(k+1)}{1+\gamma(k+1)} \\
\mathbf{e}(k+1) &= \mathbf{d}(k+1) - \mathbf{x}^T(k+1)\mathbf{w}(k+1) \\
\mathbf{w}(k+1) &= \mathbf{w}(k)+\mathbf{k}(k+1)\mathbf{e}(k+1) = \mathbf{w}(k)+\mathbf{R}^{-1}(k+1)\mathbf{x}(k+1)\mathbf{e}(k+1) \\
\mathbf{P}(k+1) &= \mathbf{P}(k) - \mathbf{k}(k+1)\mathbf{x}^T(k)\mathbf{P}(k)
\end{aligned}$$

- Computational complexity of RLS is an $\mathcal{O}(p^2)$ in terms of multiplications and additions as compared to LMS and NLMS which are $\mathcal{O}(2N)$

A general form of recursive learning algorithms

Let us introduce a new 'rotated' or 'prewhitened' vector

$$\mathbf{k}(k) = \mathbf{R}^{-1}(k)\mathbf{x}(k)$$

and

$$\mathbf{R}^{-1}(k+1) = \mathbf{R}^{-1}(k) - \mathbf{k}(k+1)\mathbf{x}^T(k+1)\mathbf{R}^{-1}(k)$$

A general form of the RLS algorithm:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{k}(k+1)\alpha(k+1)$$

where

$$\alpha(k+1) = d(k+1) - \mathbf{x}^T(k+1)\mathbf{w}(k)$$

Cf. LMS - the term $\mu\mathbf{x}(k)$ has been replaced by a time–varying matrix $\mathbf{k}(k)$

An even more general framework
from LMS to Kalman filter

- Gradient algorithms can track the weights only approximately $\looparrowright$ we use the symbol $\hat{\mathbf{w}}$, to give a general expression for a weight update

$$\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \eta \mathbf{\Gamma}(k) \left(y(k) - \mathbf{x}^T(k) \hat{\mathbf{w}}(k) \right)$$

  where $\mathbf{\Gamma}(k)$ is a gain matrix which determines 'adaptation direction'

- To asses how far an adaptive algorithm is from the optimal solution, consier the weight error vector, $\tilde{\mathbf{w}}(k)$, and sample input matrix $\mathbf{\Sigma}(k)$

$$\tilde{\mathbf{w}}(k) = \mathbf{w}(k) - \hat{\mathbf{w}}(k), \quad \mathbf{\Sigma}(k) = \mathbf{\Gamma}(k) \mathbf{x}^T(k)$$

- We now have a new weight error equation

$$\tilde{\mathbf{w}}(k+1) = \left(\mathbf{I} - \eta \mathbf{\Sigma}(k) \right) \tilde{\mathbf{w}}(k) - \eta \mathbf{\Gamma}(k) \nu(k) + \Delta \mathbf{w}(k+1)$$

LMS, RLS and Kalman filter

For different gains different learning algorithms can be obtained

The Least Mean Square (LMS) algorithm

$$\mathbf{\Sigma}(k) = \mathbf{x}(k)\mathbf{x}^T(k)$$

The Recursive Least Squares (RLS) algorithm

$$\begin{aligned}
\mathbf{\Sigma}(k) &= P(k)\mathbf{x}(k)\mathbf{x}^T(k) \\
P(k) &= \frac{1}{1-\eta}\left[P(k-1) - \eta\frac{P(k-1)\mathbf{x}(k)\mathbf{x}^T(k)P(k-1)}{1-\eta+\eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)}\right]
\end{aligned}$$

The Kalman Filter (KF) algorithm

$$\begin{aligned}
\mathbf{\Sigma}(k) &= \frac{P(k-1)\mathbf{x}(k)\mathbf{x}^T(k)}{R+\eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)} \\
P(k) &= P(k-1) - \frac{\eta P(k-1)\mathbf{x}(k)\mathbf{x}^{\mathbf{T}}(k)P(k-1)}{R+\eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)} + \eta Q
\end{aligned}$$

Appendix: Towards RLS - Observations and notation

- No assumptions are made for the statistics of the input and desired response

- Parameter **w** is found to be the best solution in terms of minimising $J(n)$ for the observation interval $\{1, \ldots, n\}$.
  - Desired signal $\mathbf{d}(n) = [d(n), d(n-1), \ldots, d(1)]^T$
  - Input signal $\mathbf{x}(n) = [x(n), x(n-1), \ldots, x(1)]^T$
  - Data matrix $\mathbf{X}(n) = [\mathbf{x}(n), \mathbf{x}(n-1), \ldots, \mathbf{x}(n-p+1)]$
  - Filter weights $\mathbf{w}(n) = [w_1(n), \ldots, w_p(n)]^T$
  - Error vector $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)$

Appendix: Towards RLS - Optimal Least Squares solution

- If $\mathbf{w}(n)$ is chosen so that $\mathbf{X}(n)\mathbf{w}(n) = d(n)$ then the error $\mathbf{e}(n) = 0$
- We assume an over–determined system $(N > p)$
- The block least squares solution is found by pre–multiplying with $\mathbf{X}^T n)$ to give

$$\left(\mathbf{X}^T(n)\mathbf{X}(n)\right)\mathbf{w}_{opt}(n) = \mathbf{X}(n)\mathbf{d}(n)$$

$$\mathbf{w}_{opt}(n) = \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{d}(n)$$

provided the columns of $\mathbf{X}(n)$ are linearly independent

Appendix: Towards RLS - Orthogonality

- Notice that $\mathbf{P}(n) = \mathbf{X}(n)\left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{X}(n)$ is a projection matrix
- It projects an arbitrary $n \times 1$ dimensional vector onto the column space spanned by $\mathbf{X}(n)$
- The orthogonality states that at $\mathbf{w}_{opt}$ the error os orthogonal to the data on which the prediction is made

$$\left(\mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)\right)^T \mathbf{X}(n) = \mathbf{0}^T$$

- Taking transpose on both sides

$$\mathbf{w}_{opt}(n) = \left(\mathbf{X}^T(n)\mathbf{X}(n)\right)^{-1}\mathbf{X}^T(n)\mathbf{d}(n)$$

Appendix: Towards RLS - 'Recursive Least Squares (RLS)

- We desire

$$\mathbf{w}(n+1) = f\left[\mathbf{w}(n), \mathbf{x}(n+1)\right]$$

- To achieve this, we partition the data matrix

$$\mathbf{X}(n+1) = \begin{pmatrix} \mathbf{x}^T(n+1) \\ \dots \\ \mathbf{X}(n) \end{pmatrix}$$

- A sample correlation matrix $\mathbf{R}$ can be written as

$$\mathbf{R}(n+1) = \mathbf{X}^T(n+1)\mathbf{X}(n+1) = \mathbf{R}(n) + \mathbf{x}(n+1)\mathbf{x}^T(n+1)$$

Notes

○

Notes

○

Notes

○

Notes

○