Least Squares, and Wiener Filters

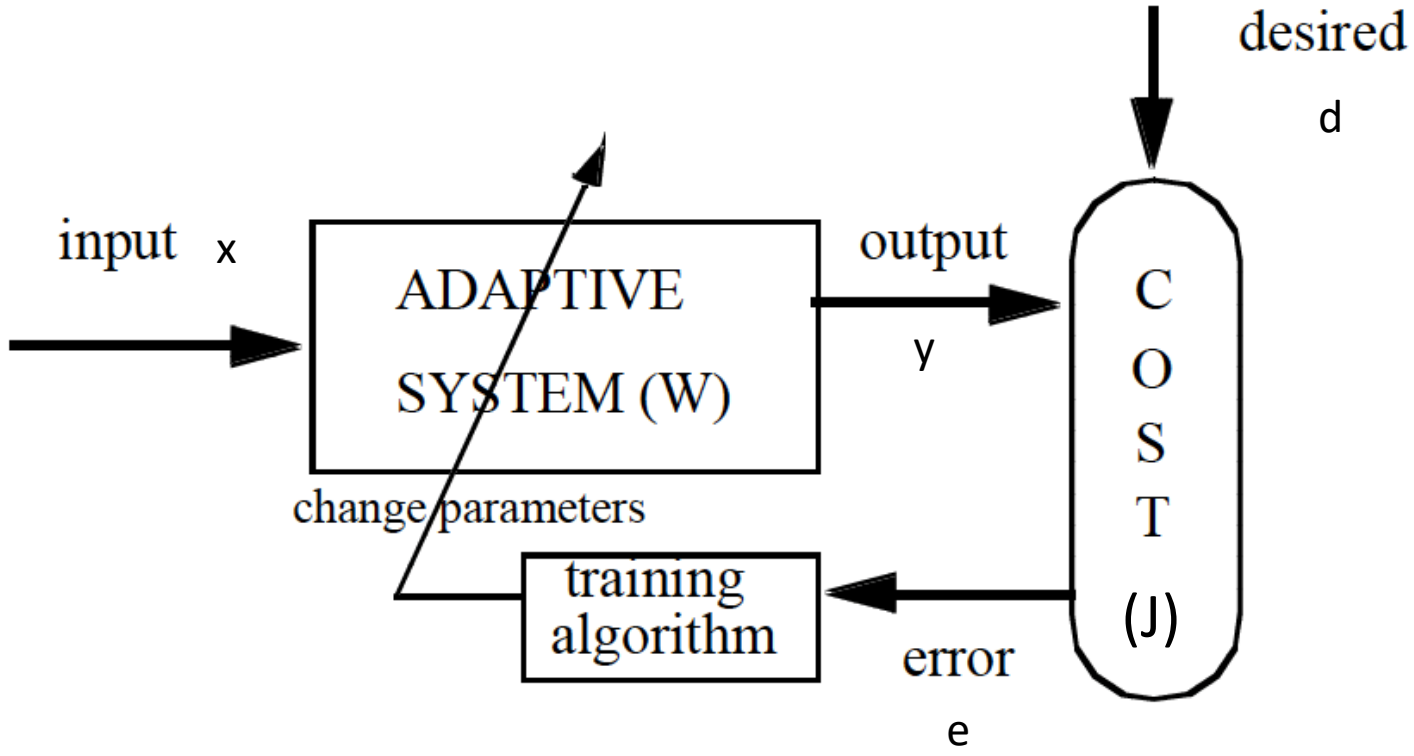
Jose C. Principe

Model Based Supervised Learning

- Assume that we obtain from the world pairs of N data samples {xi,di} , i=1,….N and we want to model their relationship. There are 3 basic components to an adaptive model: the Mapper, the Cost and the Training algorithm.
- The user must select the type of model, the function for the cost and the training algorithm (and some hyper-parameters related to the choices made).
- After these choices, the process runs independently of the user because of the intrinsic feedback, and the data CONTROLS the outcome.
- The choices are not equally good, so the user must understand the data structure as well as the limitations of the choices being made.

Model Based Supervised Learning

- Fundamental block diagram for supervised learning



$\{x_i, d_i\}$ are scalars

$e_i = d_i - y_i$

$$J = \frac{1}{2N}\sum_{i=1}^{N} e_i^2$$

NOTE: Parameters W are unknown, so goal is

$$\min_{w} J$$

Model Based Supervised Learning

- First choice is type of mapping function. We will start with the simplest: the linear model $y=wx+b$
- The goal is to have y approximate the data component d when we put x as the input to the model.
- Therefore we would like to make the error $e=d-y$ as small as possible across the data set.
- Since J is a function of the parameters (normally called weights) in this simple case we can solve the problem by hand.
- To simplify let us assume first that b is zero

Model Based Supervised Learning

- Substituting y in the error, we get

$$J = \frac{1}{2N}\sum_{i=1}^{N}(d_i - wx_i)^2$$

- The goal is to find a value of w that produces the least change in J. Just by looking at the square we see that point will correspond to the minimum of J, which will be the optimum w, denoted as w*.
- We can find this value mathematically by taking the partial derivative of J w.r.t. w, i.e. $\frac{\partial J}{\partial w} = 0$, which gives

$$\frac{1}{N}\sum_{i=1}^{N}(d_i - wx_i)\quad x_i = 0 \xrightarrow{\text{yields}} w^* = \frac{\sum_{i=1}^{N} d_i x_i}{\sum_{i=1}^{N} x_i^2} = \frac{\sum_{i=1}^{N} d_i x_i}{\lambda}$$

λ is eigenvalue

- This is the famous Least Squares procedure. Regression also shares the same solution, although it imposes a statistical model upon the data.

Model Based Supervised Learning

- In this special case the learning algorithm defaults to an analytic solution. Notice that if the data is one million samples this is time consuming, but the solution is very simple.
- If we include the bias b, what changes? J becomes

$$J = \frac{1}{2N}\sum_{i=1}^{N}(d_i - wx_i + b)^2$$

- But the procedure is the same to obtain the optimum w*, we just have two unknowns, w and b , i.e. $\frac{\partial J}{\partial w} = 0, \frac{\partial J}{\partial b} = 0$ so we will get a set of two equations in two unknowns

$$\begin{cases} \frac{1}{N}\sum_{i=1}^{N}(d_i - wx_i + b) \quad x_i = 0 \\ \frac{1}{N}\sum_{i=1}^{N}(d_i - wx_i + b) \quad = 0 \end{cases}$$

Model Based Supervised Learning

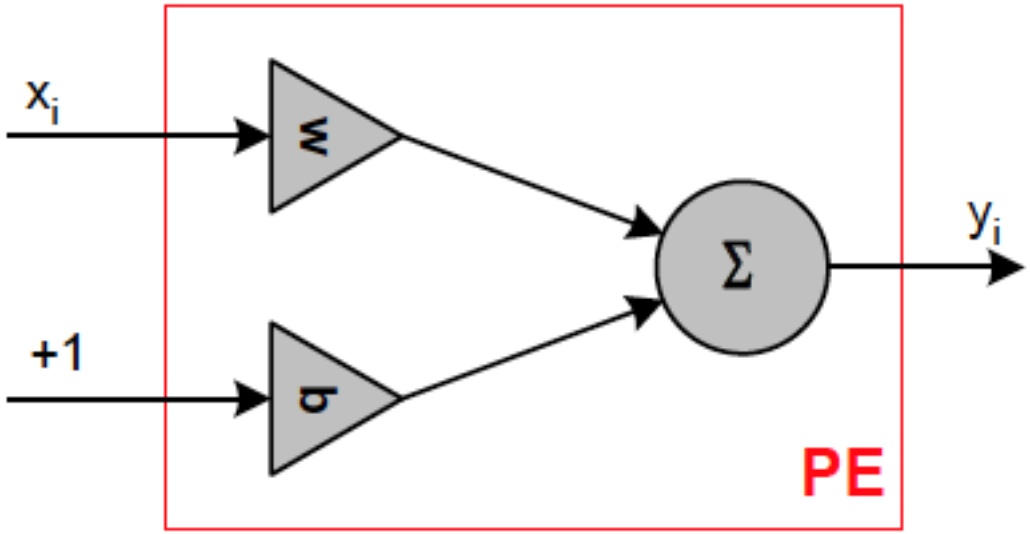
- Solution is still analytic as before, but now we have to further solve a set of equations with a cardinality given by the set of unknown parameters, which is more time consuming, but there are numerical algorithms for this.
- For this simple case we can get b of w from the second equation, and then substitute in the first to obtain w, and then obtain b. The solution is

$$\begin{cases} w = \dfrac{\sum_i {d_i x_i}/{N} - \left(\sum_i {d_i}/{N}\right)\left(\sum_i {x_i}/{N}\right)}{\left(\sum_i {x_i^2}/{N}\right) - \left(\sum_i {x_i}/{N}\right)^2} \\ b = -\sum_i {d_i}/{N} + \dfrac{\left(\sum_i {d_i x_i}/{N}\right)\left(\sum_i {x_i}/{N}\right) - \left(\sum_i {d_i}/{N}\right)\left(\sum_i {x_i}/{N}\right)^2}{\left(\sum_i {x_i^2}/{N}\right) - \left(\sum_i {x_i}/{N}\right)^2} \end{cases}$$

Model Based Supervised Learning

- Although these values are optimal under the linear model, it does not mean that the solution is ANY GOOD. This depends upon the data structure that we do not control.

- This is the essence of ML: we hypothesize a model for the data and we find the best possible parameters, but if the model we impose is weak, the results will not be good enough for the application.



Model Based Supervised Learning

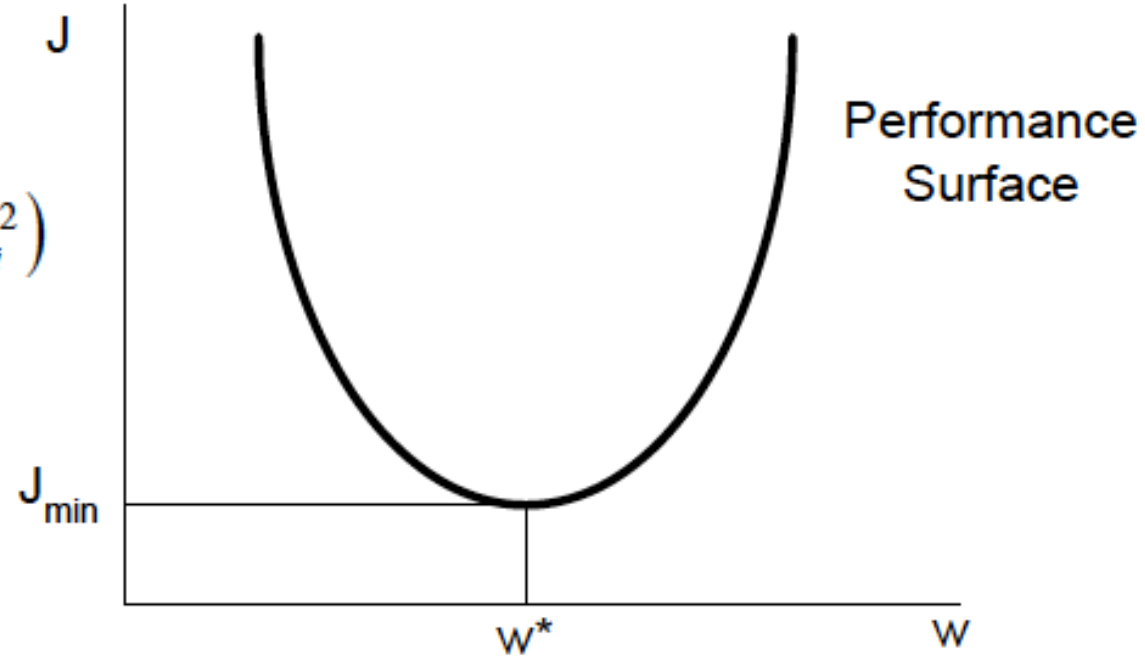
- The choice of the mapper is crucial for performance.
- But the choice of the cost is equally important because if the cost does not capture the full structure of the data, then the weights are not guided in the proper way, i.e. w* is a function of the cost.
- The advantage of MSE, is its simplicity and the intuition that minimizing the variance makes sense in optimization, but MSE only captures the second order statistics of the error variable, and when there is noise in the data it makes the solution not robust.

Learning Algorithm as a Search Procedure

- Another alternative for the learning algorithm is to use search.
- This is motivated by the shape of cost function in the space of the parameters, which we call the *Performance Surface*.
- If we expand J we get a parabola

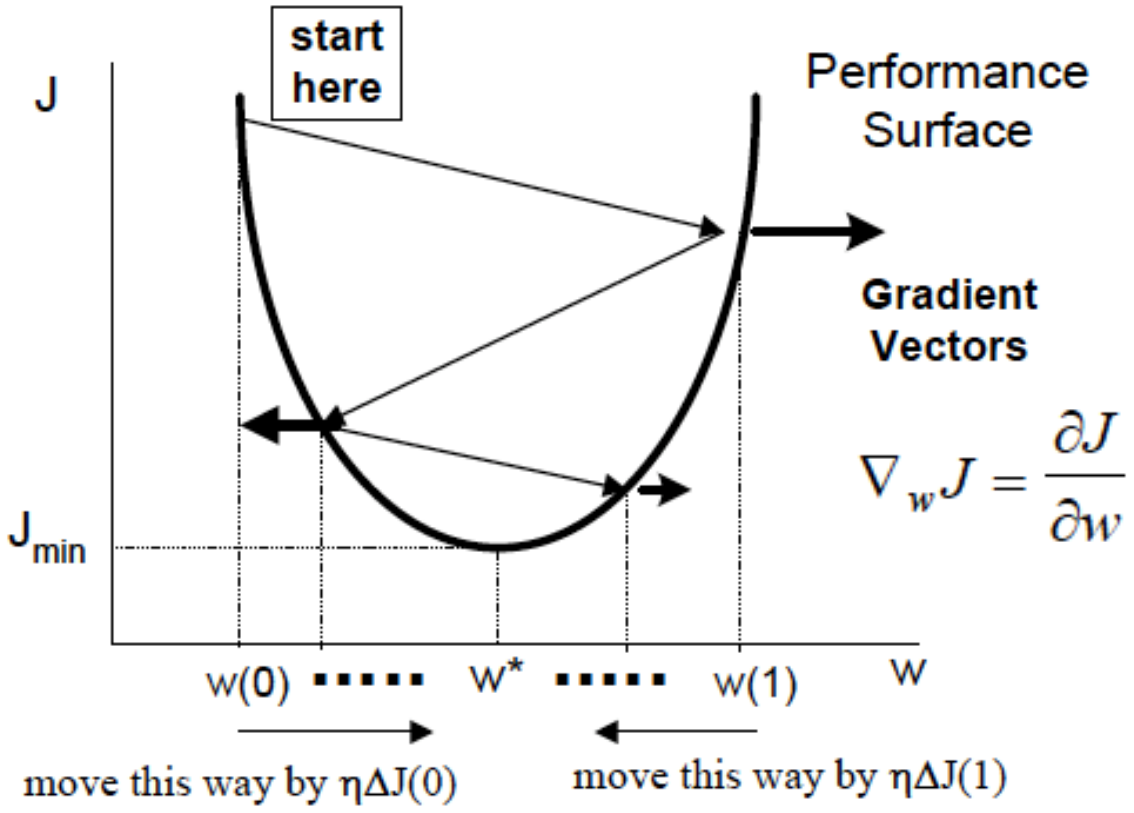
$$J = \frac{1}{2N}\sum_i \left(d_i - wx_i\right)^2 = \frac{1}{2N}\sum_i \left(x_i^2 w^2 - 2d_i x_i w + d_i^2\right)$$

- You should verify that W* is at the bottom of the bowl



Steepest Descent Learning

- One very old and simple search method is to use the information of the gradient of the performance surface, called *steepest descent*.



Start with a random initial condition and correct the current weight proportionally (η) to the gradient. Since we want to go to the bottom, we should go in the opposite direction of the gradient (gradient descent), i.e.

$$w(k+1) = w(k) - \eta \nabla J(k)$$

η is called the stepsize or learning rate

Steepest Descent Learning

- If the stepsize η is small enough, the picture tells us that the weights will gravitate towards the neighborhood of the optimum w*. In a naive way, η should be less than the inverse of the power (variance) of the input data.
- The **stepsize is an hyper-parameter** that is selected by the user and controls the dynamic of learning. The diameter of the neighborhood is controlled by η as well as the number of steps to reach the minimum, which creates a fundamental compromise because we want to quickly get to the neighborhood of w* (large η) but keep close to it (small η).
- Notice also that we can apply steepest descent at every sample i, to correct the weight, which is **called online learning**. This is different from the LS solution that uses ALL of the data (**batch learning**)

The Least Mean Square Algorithm (LMS)

- Notice that the gradient is still is a function of all the data samples.
- Bernie Widrow in the 50's proposed a very clever simplification that uses the current error as an approximation to the sum. Since the approximation is unbiased (preserves the average direction), and the process is iterative, it works!

$$\nabla J(k) = \frac{\partial}{\partial w} J(k) = \frac{\partial}{\partial w} \frac{1}{2N} \sum \varepsilon^2 \approx \frac{1}{2} \frac{\partial}{\partial w} \left(\varepsilon^2(k) \right) = -\varepsilon(k) x(k)$$

- With two multiplications per weight we can find the optimum!

$$w(k+1) = w(k) + \eta \varepsilon(k) x(k)$$

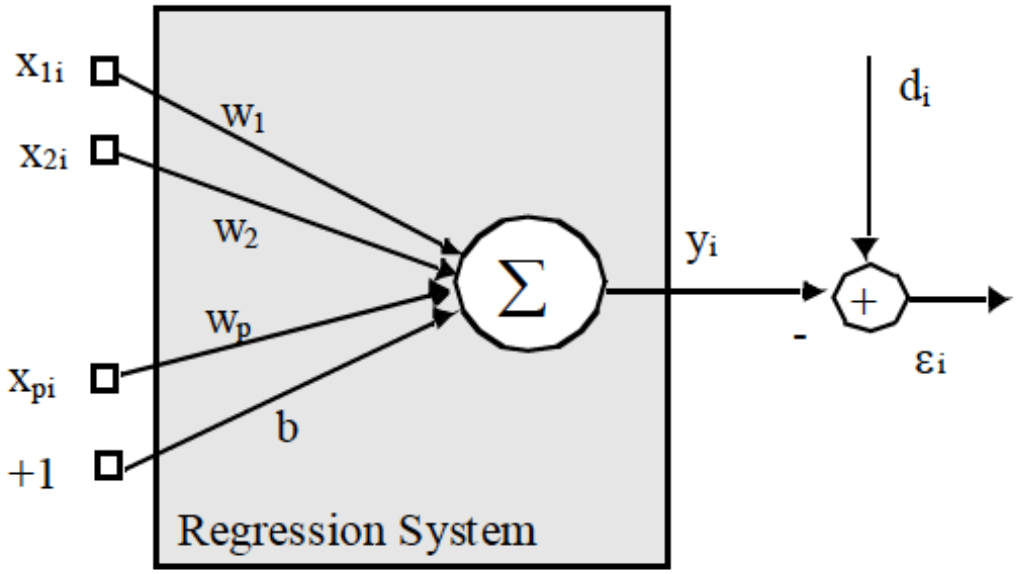
- So the LMS update only needs the current error and the current input to the weight. This is the **precursor to backpropagation** !!!!

Least Squares for Multiple Weights

- Assume that now we have collected N samples from a vector of inputs (size P) and a scalar desired, i.e. $\{x_{ik}, d_i\}$, i=1,….N; k=0,….P.
- We still use a linear mapper, but now we have P+1 free parameters w_k (x_0 is +1 for the bias b). Basically we would like to fit an hyper plane to our data.
- The cost remains the same

$$J = \frac{1}{2N}\sum_i \left(d_i - \sum_{k=0}^{p} w_{ik} x_{ik} \right)^2$$

- We just need to solve for LS



Least Squares for Multiple Weights

- Procedure is basically the same. Take the derivative of the cost with respect to each of the P+1 parameters and set each to zero. Be ready because this gets very detailed….

$$\frac{\partial J}{\partial w_j} = -\frac{1}{N}\sum_i x_{ij}\left(d_i - \sum_{k=0}^{p} w_k x_{ik}\right) = 0 \qquad \text{for j} = 0\ldots\text{p}$$

- We end up with a set of P+1 equations in P+1 unknowns (the weights)

leading to the *normal equations*

$$\sum_i x_{ij} d_i = \sum_{k=0}^{p} w_k \sum_i x_{ik} x_{ij} \qquad \text{j} = 0,1,\ldots\text{p}$$

expanding…..

$$\sum_i x_{i0} d_i = \sum_k w_k \sum_i x_{ik} x_{i0}$$
$$\sum_i x_{i1} d_i = \sum_k w_k \sum_i x_{ik} x_{i1}$$
$$\sum_i x_{ip} d_i = \sum_k w_k \sum_i x_{ik} x_{ip}$$

Least Squares for Multiple Weights

- The vector and matrix notation comes to our rescue (forget about writing scalar equations…)
- Let us define the autocorrelation function of the input between channels k and j as

$$R_{kj} = \frac{1}{N}\sum_i x_{ik}x_{ij}$$

- So we obtain the autocorrelation matrix which is square and symmetric. The size of this matrix is (P+1)x(P+1)
- But each element requires N multiplications so R is Nx(P+1)

$$\mathbf{R} = \begin{bmatrix} R_{00} & R_{01} & \cdots & R_{0p} \\ R_{10} & R_{11} & \cdots & R_{1p} \\ \cdots & \cdots & \cdots & \cdots \\ R_{P0} & R_{p1} & \cdots & R_{pp} \end{bmatrix}$$

Least Squares for Multiple Weights

- Let us also define the crosscorrelation function between the input j and desired data as

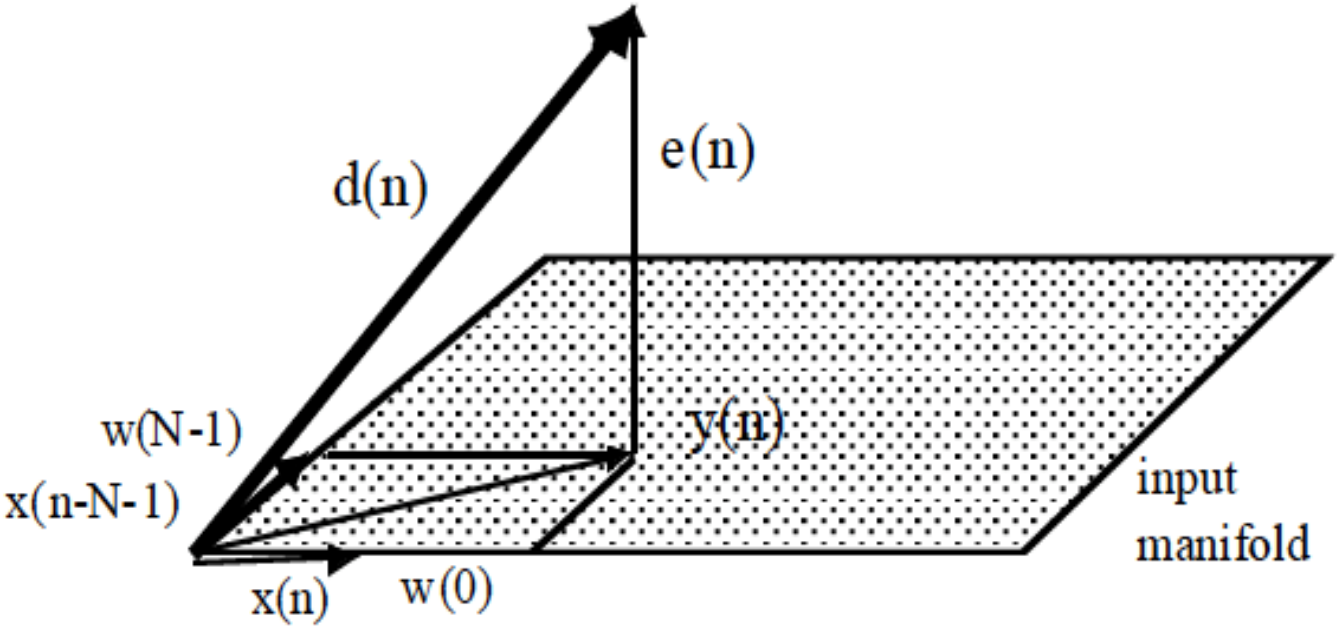
$$P_j = \frac{1}{N}\sum_i x_{ij} d_i$$

- We are going to define the weight vector and the cross correlation vector as
- $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_p \end{bmatrix}$ $\mathbf{p} = \begin{bmatrix} P_0 \\ P_1 \\ \dots \\ P_p \end{bmatrix}$ So $\begin{bmatrix} R_{00} & R_{01} & \dots & R_{0p} \\ R_{10} & R_{11} & \dots & R_{1p} \\ \dots & \dots & \dots & \dots \\ R_{P0} & R_{p1} & \dots & R_{pp} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_p \end{bmatrix} = \begin{bmatrix} P_0 \\ P_1 \\ \dots \\ P_p \end{bmatrix}$
- The normal equations in matrix form are **Rw=p** and so **w***=**R**$^{-1}$**p**

Geometric Interpretation of Least Squares

- Let each of the input data be represented by a vector $\mathbf{x}_k$, k=0…P. likewise for the desired data **d**.
- With this interpretation the output y of the linear model will always live in the space spanned by the input data, because it is a linear combination of inputs. Changing the weight vector **w** will change the point where y will be.

The Least Square solution finds the weight vector where *y is the orthogonal projection of the desired data on the input space*, because the norm of the error will be the minimum for this case.



Least Squares for Multiple Weights

- If you look at the single weight case, you see that it conforms with this solution, i.e. it is a special case. Remember division is not defined for matrices, so it is replaced by multiplication with the inverse.
- Solving the normal equation is expensive computationally. A general inverse is $O(P^3)$, but since **R** is symmetric there are $O(P^2)$ solutions. But for larger data sets the ACF may even be more expensive because it is O(NxP).
- Notice that in order to compute this solution R must be full rank, otherwise there will be infinite number of solutions. A useful procedure is to compute $\mathbf{w}^*=(\mathbf{R}+\lambda\mathbf{I})^{-1}\mathbf{p}$, where λ is a small regularization constant. This inverse is always full rank, but will distort slightly the information in R (it is an hyper-parameter).

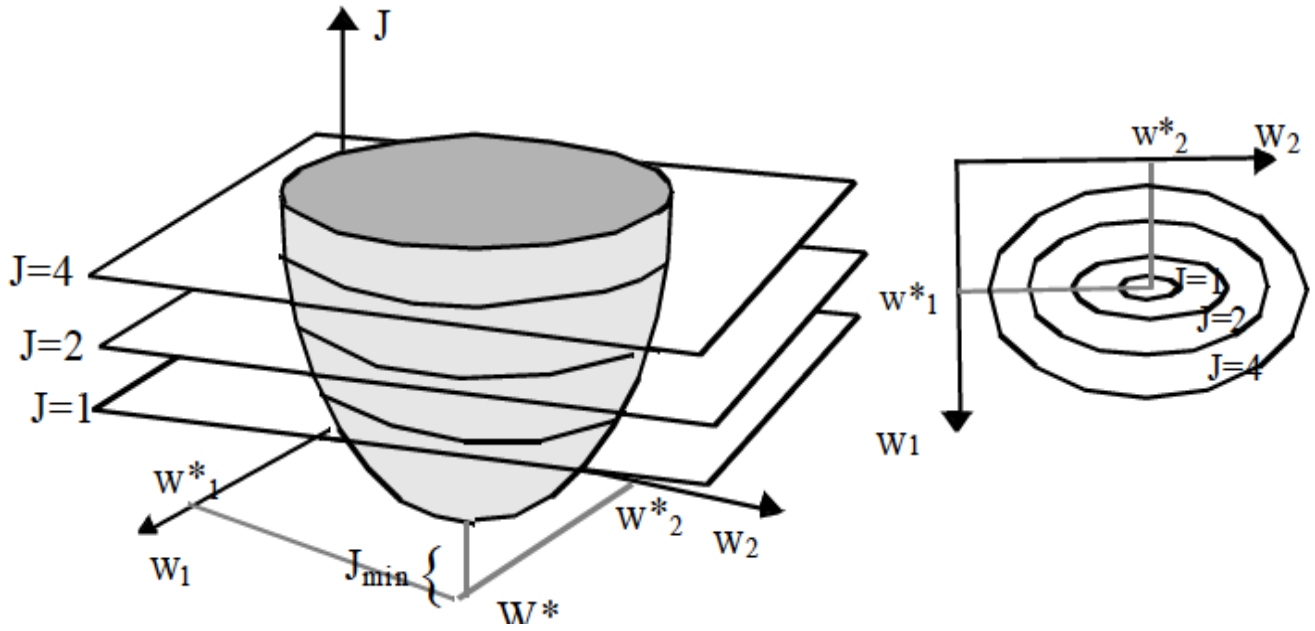
Steepest Descent for Multiple Weights

- The beautiful thing about the search methodology is that it can be extended to multiple dimensions and with LMS the solution can be written as set of P+1 equations for each weight, each requiring only 2 multiplications per weight per update (2x(P+1)xN). So it is a huge savings.
- The performance surface exists

in a space of P+1 dimensions,

$$J = \left[0.5\mathbf{w}^T\mathbf{R}\mathbf{w} - \mathbf{p}^T\mathbf{w} + \sum_i \frac{d_i^2}{2N} \right]$$

- It is a paraboloid facing upwards



Steepest Descent for Multiple Weights

- It is easy to verify that the minimum still is centered at **w***, but the paraboloid is not circularly symmetric (the contour plots are ellipsoids)
- The gradient can be written $\nabla \mathbf{J} = \left[\frac{\partial J}{\partial w_0}, \ldots, \frac{\partial J}{\partial w_p} \right]^T$ and the steepest descent equation becomes

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta \nabla \mathbf{J}(k)$$

- When we apply the LMS, the solution can be written as a set of scalar equations, one per each weight

$$w_p(k+1) = w_p(k) + \eta e(k) x_p(k) \qquad p = 0, \ldots P$$

where k is the iteration number (normally each new sample). We can still compute the weight update locally (its input and the error).

- The naïve stepsize for convergence should be smaller than the sum of the variances of each input.

Optimal Models for Time Series

- Up to now we discussed optimal linear models for multi-variate static data (i.e. weight and height of each individual in a population).
- We will address now the case of time series data. By definition, a time series is a real valued function of time, i.e. x(t) is real and t is continuous time.
- The *information in time series is contained in how signal amplitude varies across time*. This means that to model a time series we can not just use the current time, we have to use the past of the time series (since the future is unknown, i.e. causal generation system).
- This means that the mapping function in our model must change to incorporate new elements that remember the past. In signal processing we call them *memory elements*.

Optimal Models for Time Series

- In our studies we are not going to use continuous time but discrete time, i.e. we assume that the signal is sampled with an analog to digital converter (ADC), at twice the highest frequency present in the signal to preserve its amplitude information in the discrete domain.
- We further assume that the signal amplitude is quantized appropriately, i.e. with sufficient number of bits to resolve the fine detail of the signal amplitude (by selecting the appropriate number of bits in the converter).
- So our scalar digital signal is denotes {x(nT)} , n= 0, 1, …..N. we can also define multivariate time signals, but use the scalar case now.

Optimal Models for Time Series

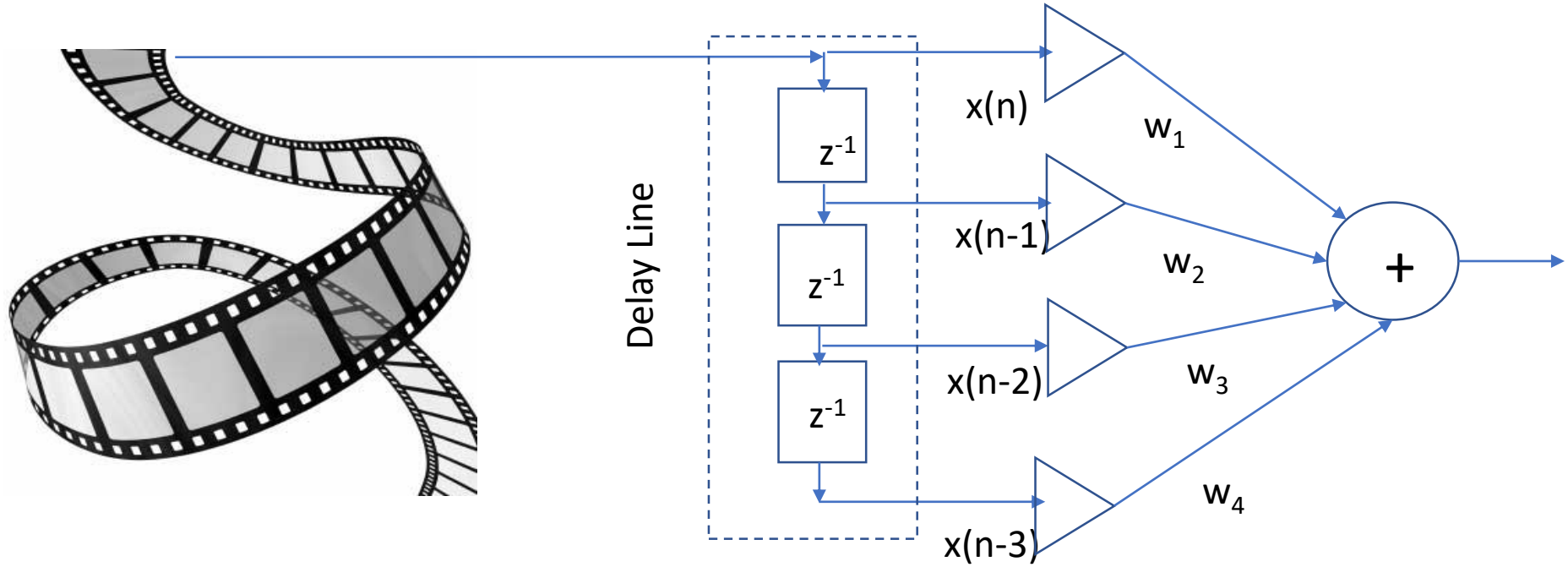
- We need to design a new element that stores its input without changing the amplitude but delays its output by one sample, which we call the ideal delay operator D[x(n)]. By definition $D[x(n)] = x(n-1)$. Normally the delay operator is denoted by z^{-1} because mathematically D(n) is simply $\delta(n-1)$, where δ is a delta function.
- It is interesting that we can write

$$\{x(n)\} = \mathbf{x} = \sum_{i=0}^{N-1} x(i)\delta(n-i)$$

Which tells us how fundamental D is

- If we cascade P delay operators, we create what is called *a delay line* that at time n is able to remember all samples between the current time and n-P, i.e. x(n), x(n-1), …. X(n-P)

Optimal Models for Time Series

- A time signal is like a one pixel movie, where we only see the current frame. But with the help of a delay line, we can see many frames at the same time, and this allows the modeling of time structure.
- 

- Delay lines are the integral part of Finite Impulse Response (FIR) filters

Optimal Models for Time Series

- An FIR filter computes an output that is the convolution of the input x(n) with the filter weights w_i (also called the impulse response h_i)

$$y(n) = x(n) \bullet h(n) = \sum_{i=-\infty}^{\infty} x(i)h(n-i)$$

- An FIR has an *extra hyper parameter* that needs to be set by the designer: *the number of delays*. This does not exist in static data.
- The important thing for us is that the delay line is a single input multiple output system that when coupled with a multivariate linear processing element that we have used for static data creates a linear model.

Optimal Models for Time Series

- The general rule is that the size of the delay line should be able to provide enough degrees of freedom to represent the dynamics of the times series. You may think that the time series is produced by a dynamical system that has certain, but unknown, degrees of freedom.
- Basically the filter order for our model should have twice the number of degrees of freedom of the system that created the time series. There are ways of estimating the size of the filter using the theory of delay embedding (Takens embedding theory).
- One practical way is to plot the autocorrelation function of the signal and look at the first zero crossing. The impulse response of the FIR (number of delays) should be similar to the lag at which the ACF goes to zero. But this is naïve… the embedding theory is more accurate.

Optimal Models for Time Series

- So how can we create an adaptive model of the time series with an FIR? We just use least squares…. as we did before.
- However, now the autocorrelation function exists in time, instead of quantifying the correlations across the multivariate inputs.
- We can expect that the optimal weight vector is given by $\mathbf{w}^*=\mathbf{R}^{-1}\mathbf{p}$
- The interpretation of this solution remains the same: It selects $\mathbf{w}^*$ to make the *error orthonormal to the input data space*, such that the norm of the error is minimized.
- This solution is normally called in the engineering literature the *Wiener solution*. Wiener was a famous mathematician that proved this in functional spaces, not in vector spaces as we do in DSP, and the solution is more general because it applies to functions (involves spectral factorization).

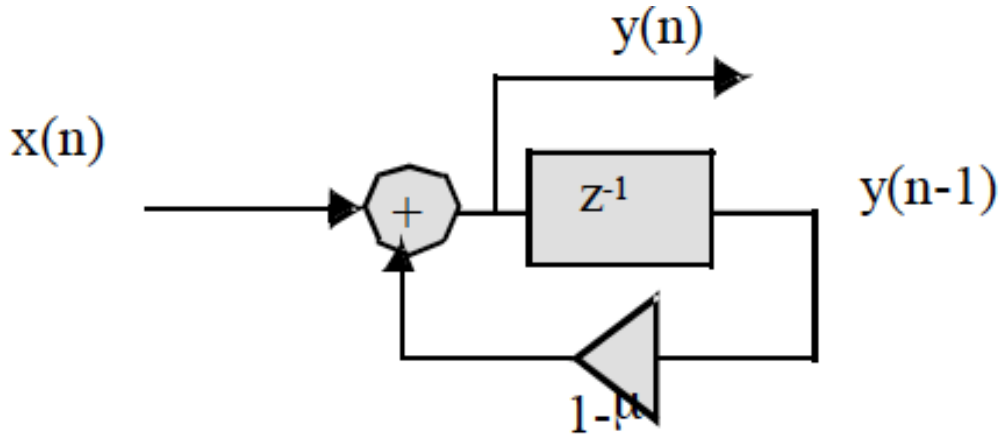
Another Linear Memory Processing Element

- There is a second memory element that is very useful in optimal modeling: the recursive first order filter.

$$y(n) = (1-\mu)y(n-1) + x(n)$$

- With impulse response

$$h(n) = (1-\mu)^n .$$

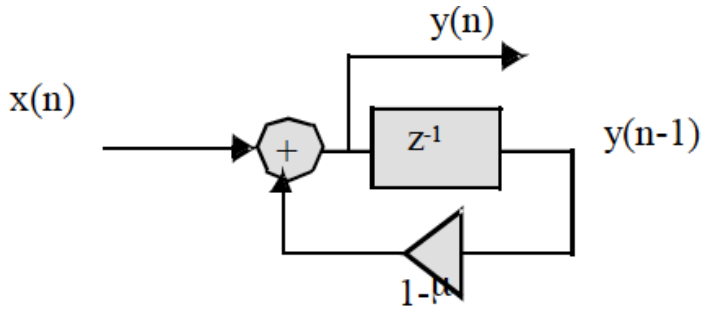


- The difference is that we are not storing the input but the previous value of the state variable. However, the system remembers something that happens in the input "for ever", i.e. the impulse response is of infinite extent and the decay is controlled by the hyper-parameter μ, with $0<\mu< 1$ for stability.
- Notice that there is a compromise, because the integrator does modify the input (can never recover an impulse in time).

Another Linear Memory Processing Element

- How can we adapt the hyper parameter μ?
- Need to take the sensitivity of y to μ, i.e.

$$\frac{\partial y(n)}{\partial \mu} = -y(n-1) + (1-\mu)\frac{\partial y(n-1)}{\partial \mu}$$



- This is very different from the FIR case where only the first term would exist, but in reality y(n-1) is ALSO a function of μ….. How can we practically compute this, since it goes on and on?
- We will do a small approximation. In adaptation μ will be changing, so it is μ(n). We will call the derivative $\partial y(n)/\partial \mu(n) = \alpha(n)$, so we can write approximately

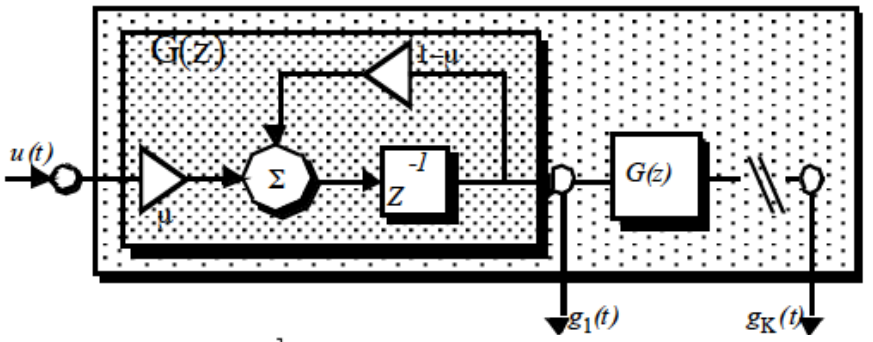
$$\alpha(n) = -y(n-1) + (1-\mu)\alpha(n-1)$$

- This can be computed recursively by the same block as above, if we substitute x(n) by y(n-1), with zero initial conditions.

The Gamma Memory

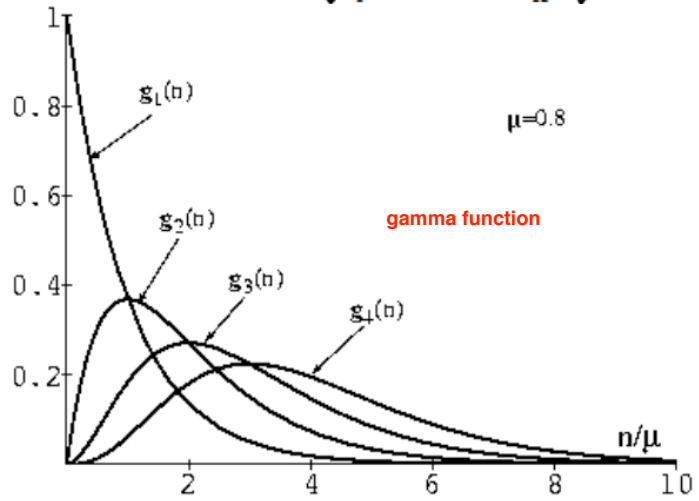
- We can cascade several integrators, and obtain a delay line that provides the optimal compromise memory depth resolution for linear models. We called this delay line the gamma memory.
- The impulse responses are

$$g_k(n) = \binom{n-1}{k-1} \mu^k (1-\mu)^{n-k}$$



which are the Gamma functions in discrete time and form a complete bases for L_2.

- The feedback parameter μ controls the scale of the time axis, which is remarkable



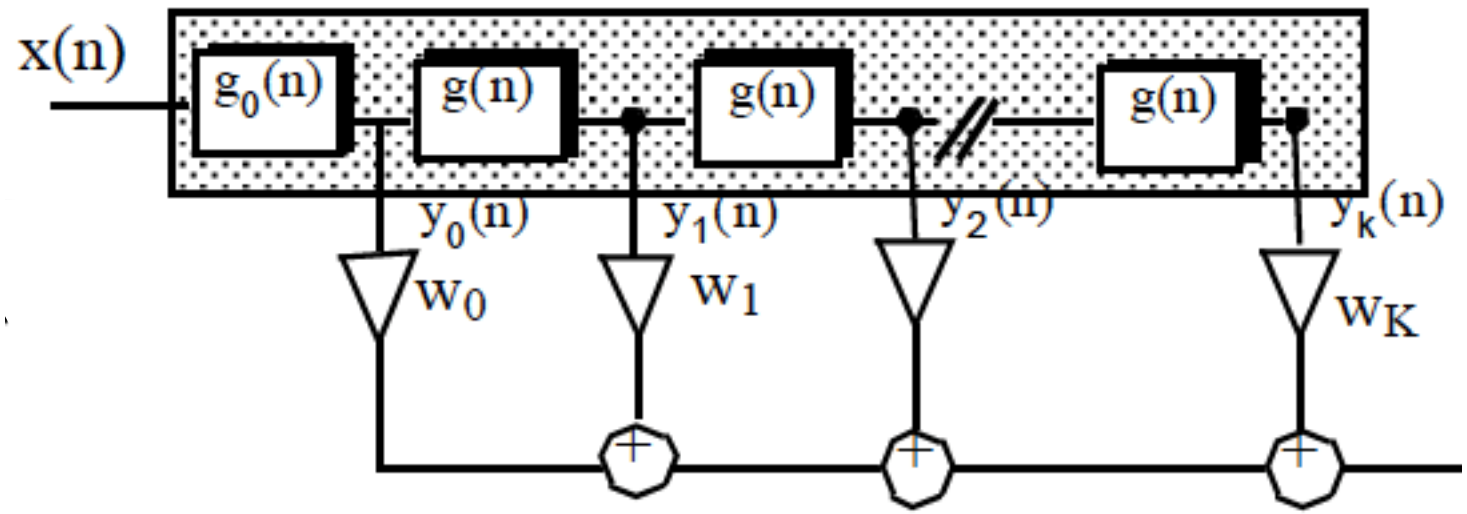
The Gamma Memory

- Let us define *memory depth* M as the center of mass of the last memory tap $g_K(n)$ as $M = \sum_{n=0}^{\infty} n g_K(n)$ and *memory resolution* R as number of taps per unit time.

  With this definition RM=D where D is the number of taps.
- This means that *any linear memory* presents a compromise between its memory depth and its resolution, that can be solved with the selection of D.
- With this definition the tap delay line has the smallest depth and the highest resolution, while the gamma memory trades resolution by memory depth.

The Gamma Filter

- From this discussion we can define the gamma filter that contains the FIR as a special case (μ=0)
- The free parameters are the K+1 weights and μ. If we decide not to adapt μ the least square solution applies without modification (but need to select μ by hand).



$y(n) = \sum_{k=0}^{K} w_k x_k(n)$; $x_k(n) = (1-\mu)x_k(n-1) + \mu x_{k-1}(n-1)$ k=1,…,K; x_0=x(n)

- But can also adapt

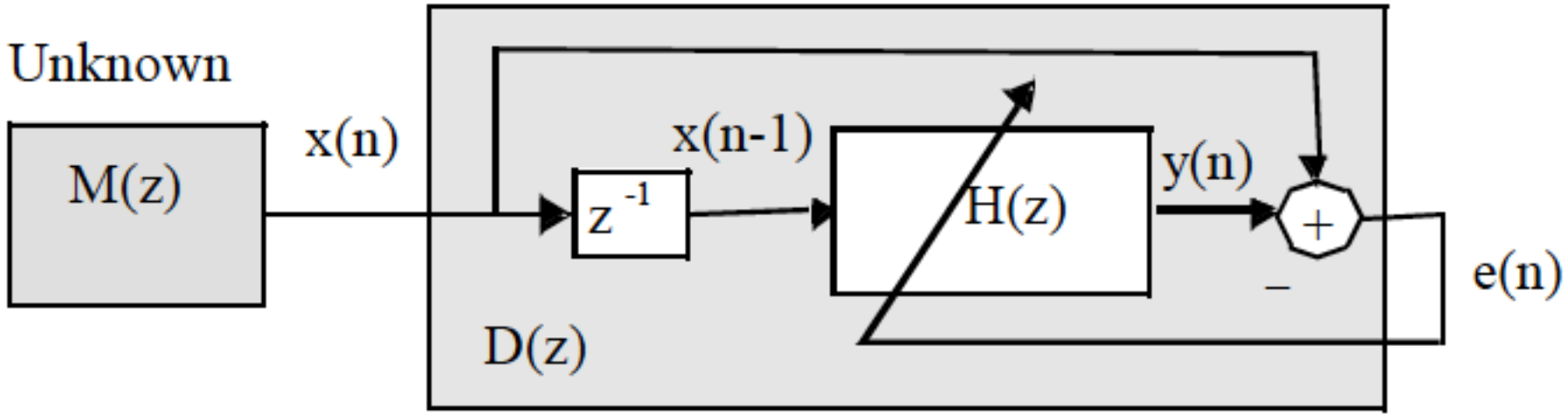
$$\Delta\mu = -\eta\frac{\partial J}{\partial \mu} = \eta\sum_{n=0}^{T} e(n)\sum_{k=0}^{K} w_k \alpha_k(n)$$

$$\alpha_0(n) = 0$$

$$\alpha_k(n) = (1-\mu)\alpha_k(n-1) + \mu\alpha_{k-1}(n-1) + [x_{k-1}(n-1) - x_k(n-1)]$$

Application of Adaptive Filters

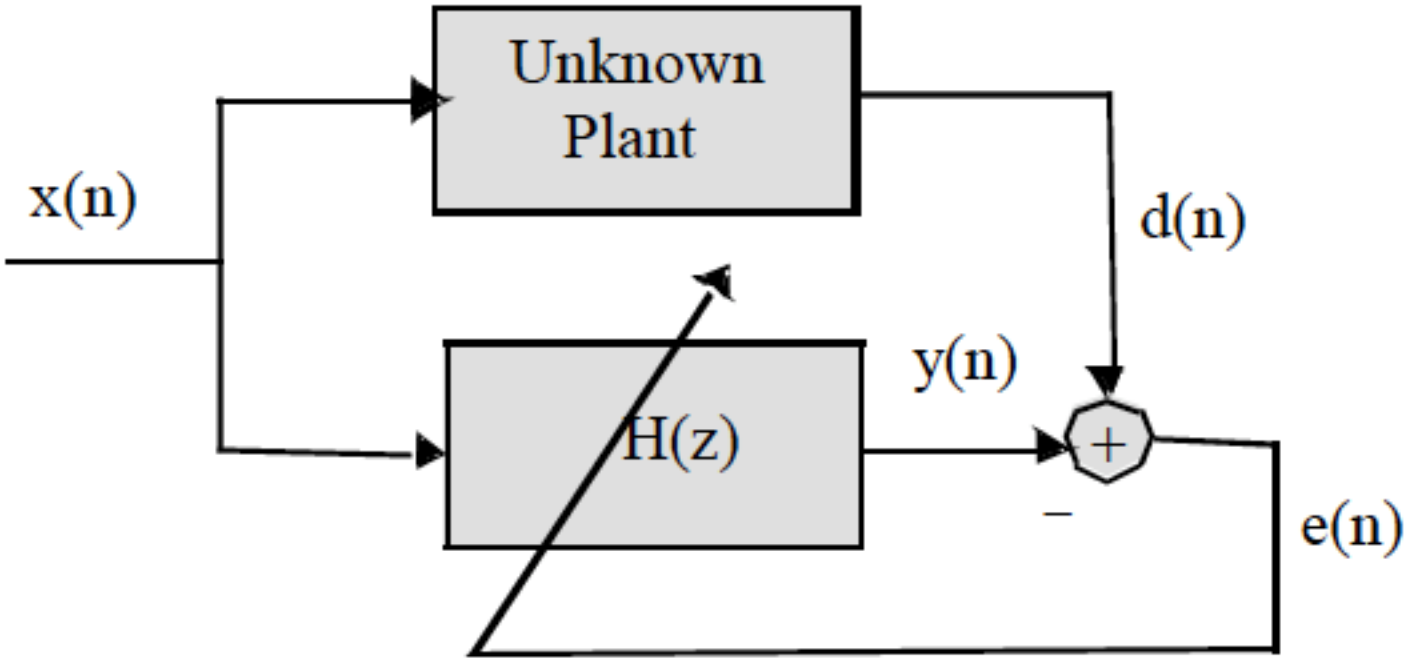
- Linear model for time series structure.



$$M(z)D(z) = k \quad \rightarrow \quad M(z) = \frac{k}{1 - z^{-1}H(z)}$$

Application of Adaptive Filters

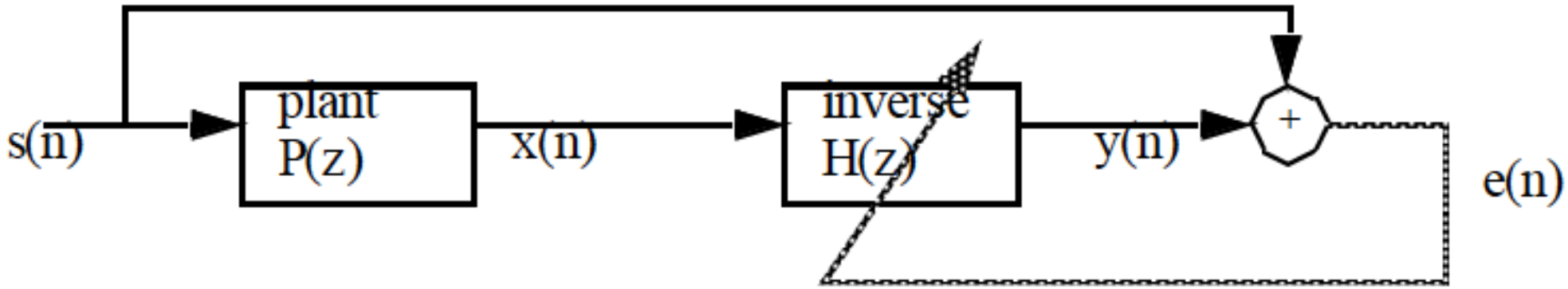
- System identification



- Minimizing the error makes the model output a substitute of the input output relation of the plant.

Application of Adaptive Filters

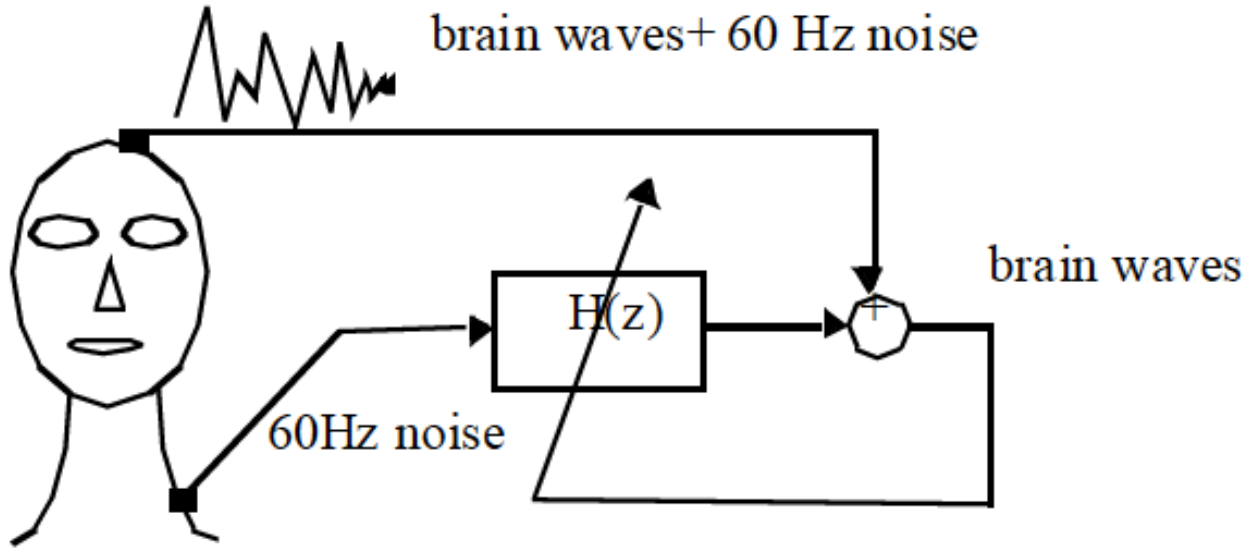
- Inverse modeling of a system



$$Y(z) \approx S(z) = H(z)P(z)S(z) \rightarrow \qquad H(z) = \frac{1}{P(z)}$$

Application of Adaptive Filters

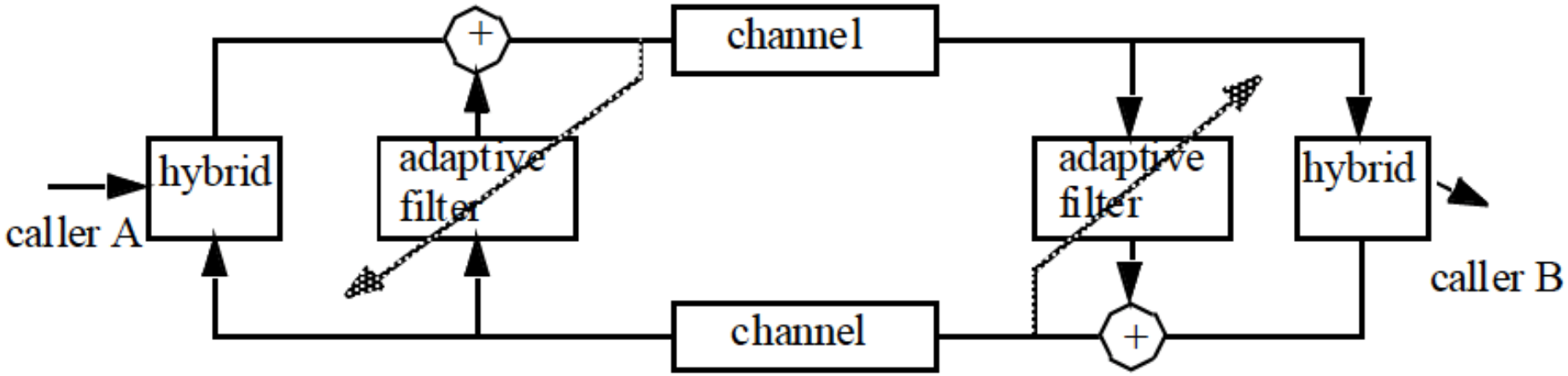
- Interference Cancelling



- Notice that the signal of interest appears as the error in this block diagram

Application of Adaptive Filters

- Echo Cancellation



- The hybrid circuit if leaky, can mix the voice of caller A (echo because of the channel transmission) into the headphone of caller A which is very annoying.

Performance Analysis of Adaptive Filters

- Wiener / Regression / Least Squares
- This is an analytic solution, so the problem we face is simply one of selecting Hyper parameters, and testing the validity of the solution.
- The hyper parameters are just the size of the data set to properly train the parameters, the filter order (for Wiener only) and to make sure that R^{-1} exists.
- For the Wiener the last lag of the autocorrelation function should be estimated with at least 1000 data samples, so depending upon the order P, we will have 1000+2P. Any value above this will be OK.
- The filter order for multivariate static data is predetermined by P. For time series, it depends upon the number of degrees of freedom of the system that generated the data.

P. Grassberger and I. Procaccia, *Measuring Strangeness of Attactors*, Physica D **9**, 198 (1983); Phys. Rev. Lett. **50**, 346 (1983).
*Kennel, M.; Brown, R.; Abarbanel, H. (1992). "Determining embedding dimension for phase-space reconstruction using a geometrical construction". Physical Review A. **45** (6): 3403–3411.* Bibcode:1992PhRvA..45.3403K. PMID 9907388. doi:10.1103/PhysRevA.45.3403.

Performance Analysis of Adaptive Filters

Wiener, Regression, Least Squares

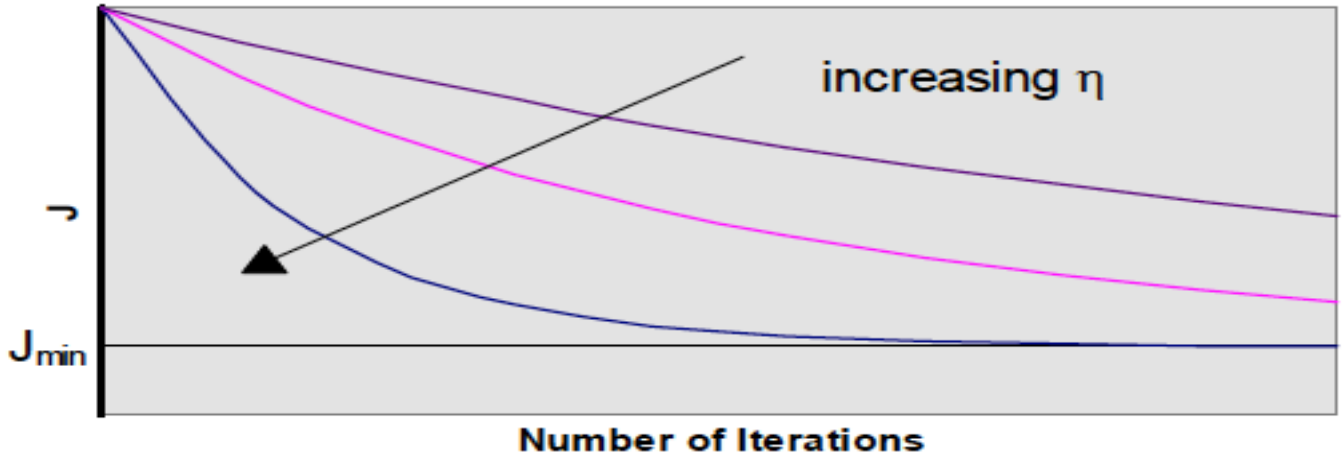
- Wiener / Regression / Least Squares
- Regarding the regularization to make R^{-1} full rank, there is a large theory on how to accomplish this, starting with Tikhonov regularization theory.
- The simplest approach is to add a full matrix to R before taking the inverse as explained before. This solution will change the location of the ideal w*, but allows a single solution, as we require.
- The parameter λ should be selected at 10% of the smallest non zero eigenvalue of R not to affect appreciably w*. However this is sometimes expensive to do, so we should use cross validation in a validation set to find the proper value.

Tikhonov, A. N. (1963). Translated in *"Solution of incorrectly formulated problems and the regularization method". Soviet Mathematics. **4**: 1035–1038.*

Golub, G.; Heath, M.; Wahba, G. (1979). "Generalized cross-validation as a method for choosing a good ridge parameter" (PDF). Technometrics. 21: 215–223.

Performance Analysis of Adaptive Filters
Steepest Descent and LMS

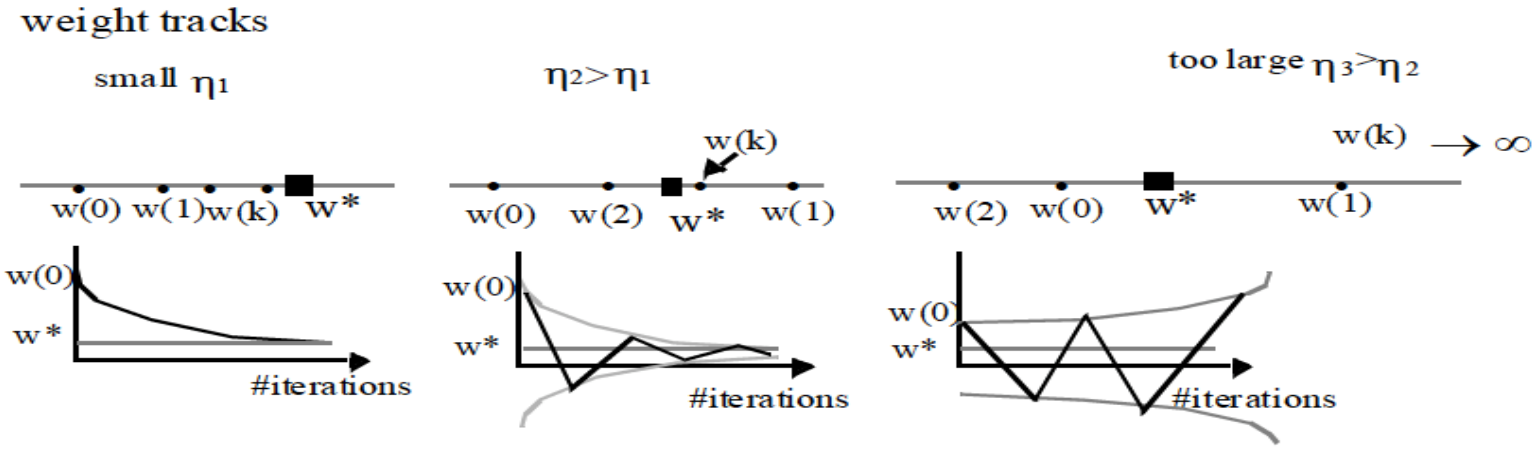
- Steepest descent is the iterative algorithm that searches the performance surface from some initial condition. It is dependent on the stepsize.
- The best way to monitor the adaptation process is to plot the change of the cost J over time, which we call the *learning curve*.

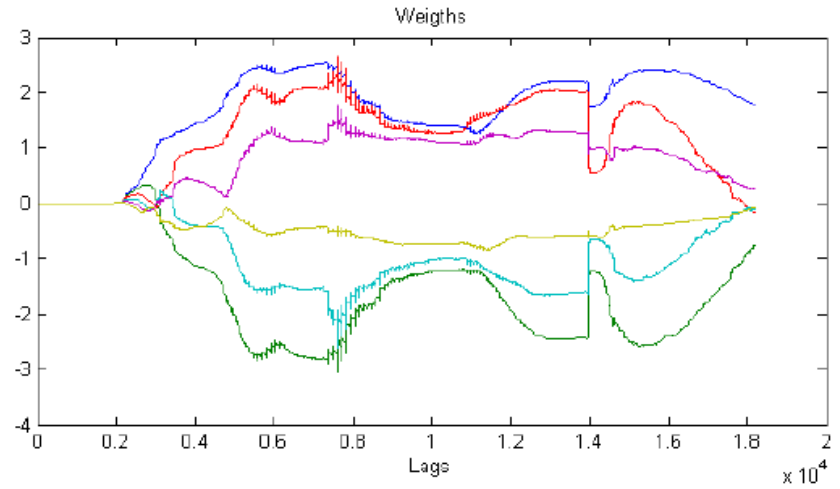


- We should avoid divergence at all costs, because the algorithm forgets the value of weights when it diverges, so we lose ALL the previous data information!!!!!

Performance Analysis of Adaptive Filters
Steepest Descent and LMS

- The plot of the weight tracks is also a very important indicator of the adaptation performance, and it is more detailed because it also tells us if the mapper is well dimensioned (if more than one weight converges to the same small value, it means we have too large a model order).





- Moreover, it also tells us if the weights have stabilized so also can control the selection of the stepsize and when to stop training.

Performance Analysis of Adaptive Filters
Steepest Descent and LMS

- There is a fundamental compromise in steepest descent that is controlled by the stepsize:
- Let us define misadjustment as $M = \frac{J_{final} - J_{\min}}{J_{\min}}$ The misadjustment is $M = \eta trace\,(R)$ and it measures the penalty in the smallest error power (J_{min}).
- The adaptation process can be modeled as an exponential decay of J to its minimum value (exp (n/τ) so time constant of adaptation $\tau = 1/(2\eta\lambda)$
- If we want steepest descent to converge fast, we increase stepsize (up to a point), but the misadjustement increases proportionally.
- If we want to converge faster and achieve J_{min}, we need to go beyond the gradient descent procedure (for instance using the recursive least square RLS algorithm).

Performance Analysis of Adaptive Filters
Steepest Descent and LMS

- The big issue with gradient descent is how to guarantee convergence in the selection of the stepsize or learning rate.
- We can show that the limit is associated with the structure of the input autocorrelation matrix, specifically its eigenvalues. In fact, for the single weight case, the performance surface can be written as (centered at w*)

  $J = J_{min} + 0.5\lambda(w - w^*)^2$, so the gradient becomes $\lambda(w - w^*)$.
- So the gradient descent equation can be written

$$w(k+1) = w(k) - \eta\lambda(w(k) - w^*) = (1 - \eta\lambda)w(k) + \eta\lambda w^*$$

- We can ow substitute w(k) by w(k-1)… until w(0) to get

$$w(k+1) = (1 - \eta\lambda)^{k+1}w(0) + \eta\lambda w^*$$

- So we conclude that for convergence to w*, $|1-\eta\lambda|<1$ or $\eta < 2/\lambda$

Performance Analysis of Adaptive Filters

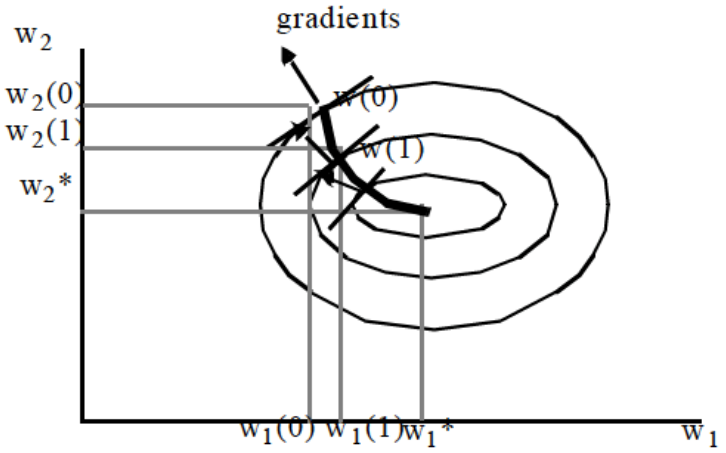
Steepest Descent and LMS

- This generalizes for the multidimensional weight case, so the parabola generalizes to an hyper paraboloid.

$$J = J_{min} + 0.5(w - w^*)^T R(w - w^*)$$

- We can still write $\mathbf{w}(k+1) = (\mathbf{I} - \eta\mathbf{R})\mathbf{w}(k) + \eta\mathbf{R}\mathbf{w}^*$
- It turns out that the adaptation is cross-coupled, i.e. convergence of **w**(n) to **w*** depends upon all the P+1 directions.

- In each direction i, the stepsize must be less than the inverse of the corresponding eigenvalue.
- Normally one just uses a single stepsize, so we have to put ourselves in the worst case condition and select $\eta < 2/\lambda_{max}$



Performance Analysis of Adaptive Filters
Steepest Descent and LMS

- Because of this coupling, the time until convergence is controlled by the smallest eigenvalue, i.e. $\tau = 1/\eta\lambda_{min}$
- Therefore the *eigenvalue spread of the input data autocorrelation matrix practically controls steepest descent learning*!
- The learning curve for the multi weight case will then show different time constants: first a fast convergence along the direction of the largest eigenvalue, but then it slows down and later it is controlled by the smallest eigenvalue.
- In practice we should use a normalized stepsize in LMS, i.e. $\eta = \eta_0/trace(R)$, with η_0 from 0.01 to 0.5.