Types of Filters

We saw three types of filters, viz. MA, AR, and ARMA. These do not include the entire class of filters but cover an important class of filters and are very useful.

For linear systems where output y(n) is related to the input x(n) by the difference equation:

$$y(n)+\sum_{k=1}^{p} g_k y(n-k)=\sum_{k=0}^{q} \mathrm{f_k} x(n-k)$$

We considered three cases:

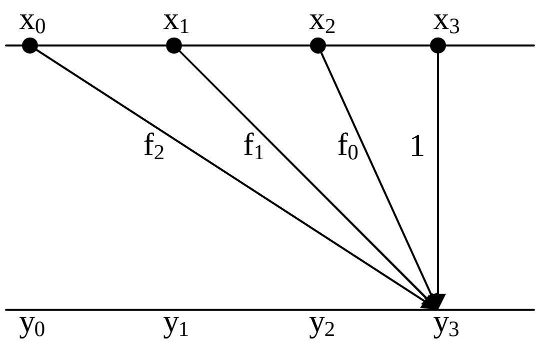
1. **Moving Average Filters (MA)**

In this case, the linear filter,

is an all-zero filter and the difference equation for their input-output relationship is

$$y(k)=x_k+f_0x_{k-1}+f_1x_{k-2}+\ldots$$

Note: The strength of these filters may not be 1

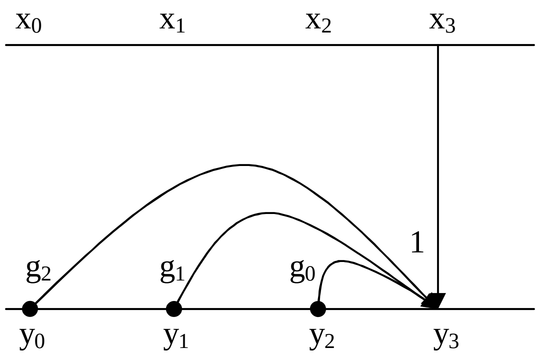


2. **Autoregressive Filters (AR)**

In this case, the linear filter,

is an all-zero filter and the difference equation for their input-output relationship is

$$y(k)=x_k+g_0y_{k-1}+g_1y_{k-2}+\ldots$$



AR Filters are linear shift invariant systems

Refer to previous AR filter figure.

Using the figure we can calculate the expressions for y_0, y_1…

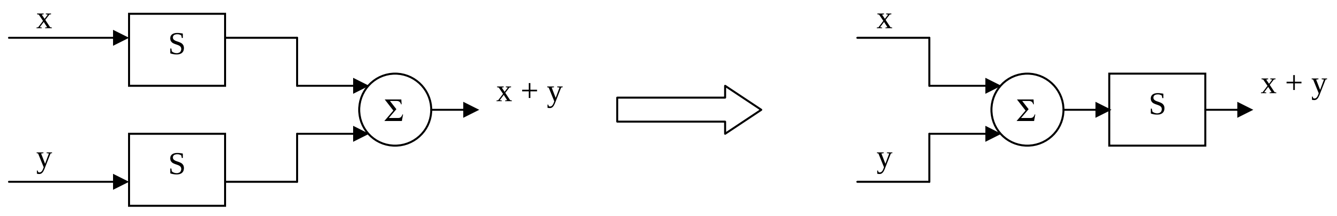
$$y_0 = x_0$$
$$y_1 = x_1 + g_0 x_0$$
$$y_2 = x_2 + g_0 x_1 + (g_0{}^2 + g_1) x_0$$
$$y_3 = x_3 + g_0 x_2 + (g_0{}^2 + g_1) x_1 + (g_0{}^3 + 2 g_0 g_1 + g_2) x_0$$

This can be seen as

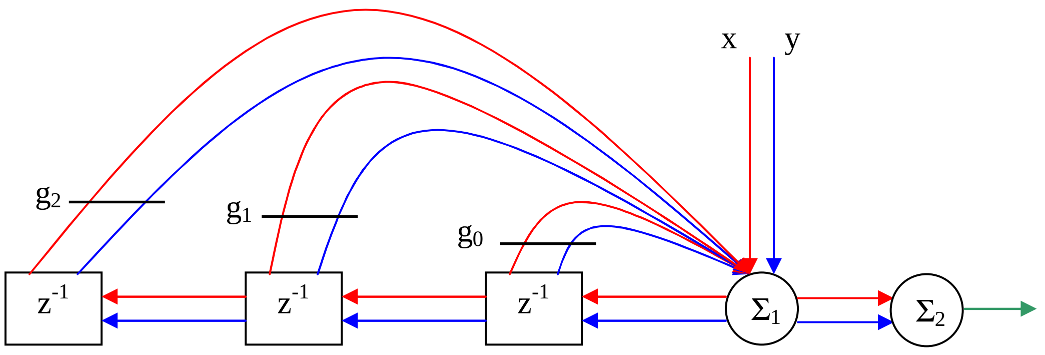
$$Y = X * F$$
$$F = \begin{bmatrix} 1 & g_0 & (g_0{}^2 + g_1) & (g_0{}^3 + 2 g_0 g_1 + g_2)\ldots \end{bmatrix}$$

By looking at the difference equation of AR filters we observe that y_k is described in terms of k and does not refer to any particular value of y or x. Hence we can say that AR filters are shift invariant

To show that AR is linear, we have to show that

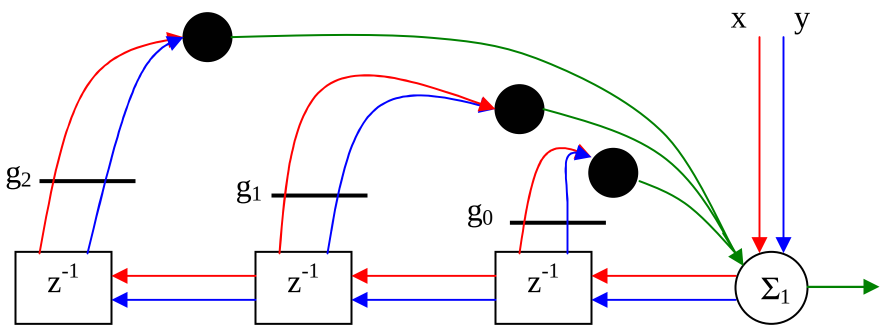


Let us superimpose the blue and red AR systems as in the figure below and finally add up the two results in Σ_2. We can look at the AR system as containing the summation element Σ_1, the delay elements z^{-1} and the multiplying elements g_k.

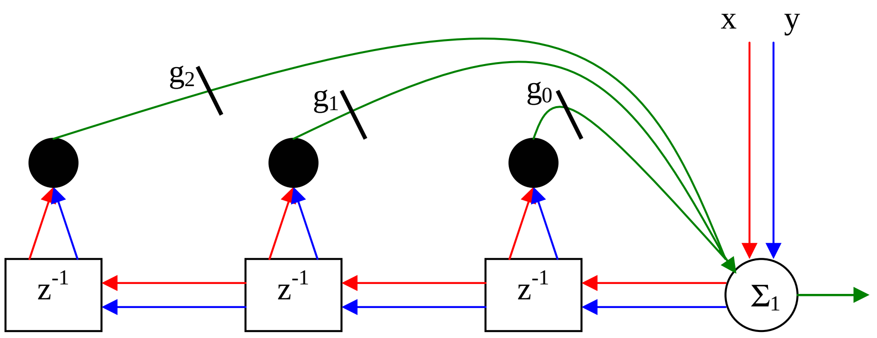


Let us denote, for convenience, Σ_2 as a black filled circle.

Add (Σ_2) before the addition (Σ_1) or after, it doesn't make a difference…

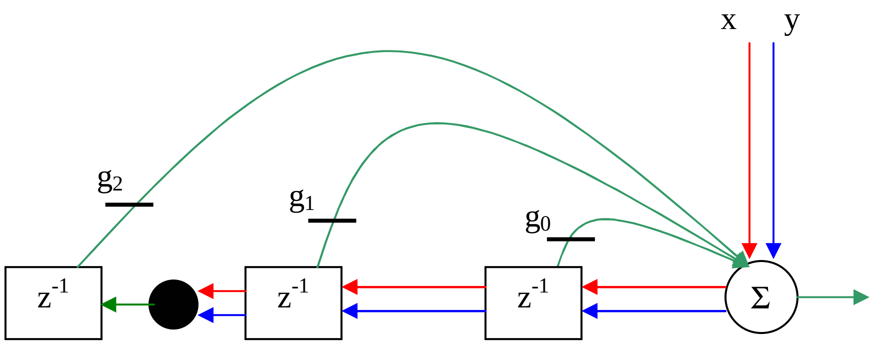


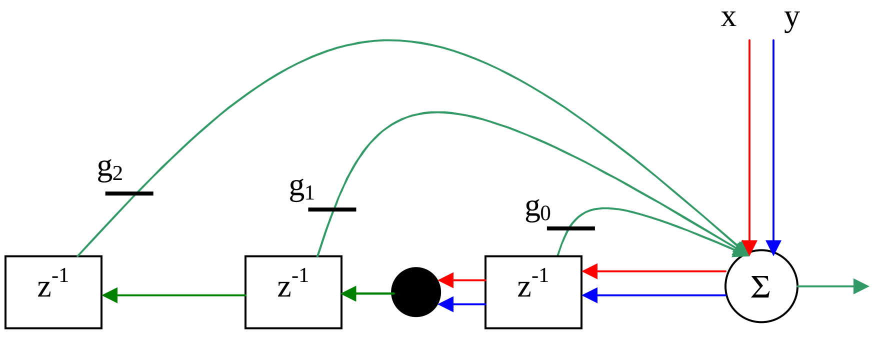
Add (Σ_2) before the multiplication (g_k) or after, it doesn't make a difference…

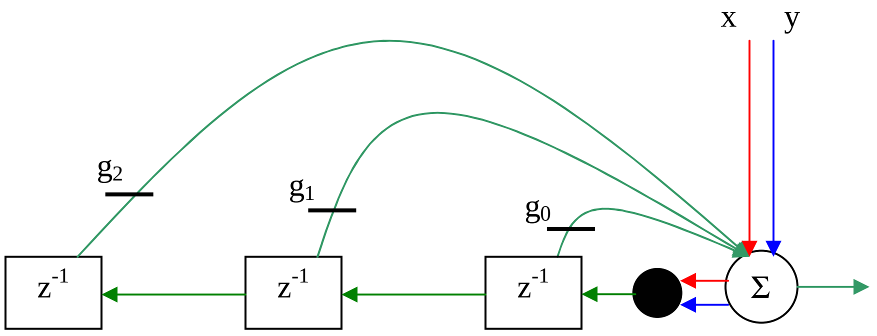


Now, for each of the three Σ_2 we have the following:
Next three figures for the Σ_2 before g_2
Among the next three, the 2nd and 3rd for the Σ_2 before g_1
Among the next three, the 3rd for the Σ_2 before g_0
At each step we combine the Σ_2's between the delay elements

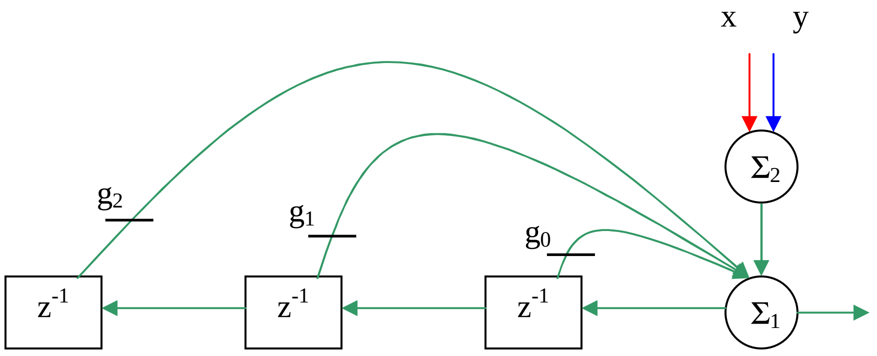
Add (Σ_2) before the delay (z^{-1}) or after, it doesn't make a difference…







Add (Σ_1) before the addition (Σ_2) or after, it doesn't make a difference…



Another way of showing AR filters as linear is by induction.
$y_0 = x_0$ forms the basis of the induction hypothesis
Let us assume y_{k-1}, y_{k-2},… to be linear
y_k is defined in terms of x_k and y_{k-1}, y_{k-2},… in the AR filter difference equation.
Since x_k is just added and the y terms are multiplied by constants, we can say that y_k is also linear.
Hence AR filters are linear systems.

3. Autoregressive, Moving Average Filters (ARMA)

In this case, the linear filter,

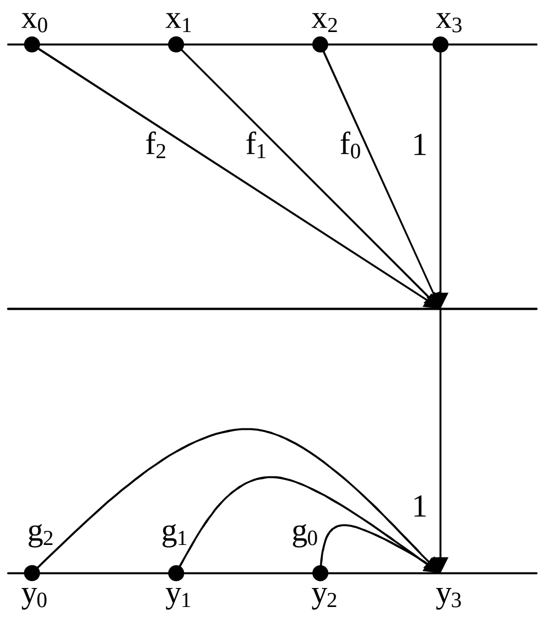
$$\frac{H(z)}{G(z)} = H(z).\frac{1}{G(z)}$$

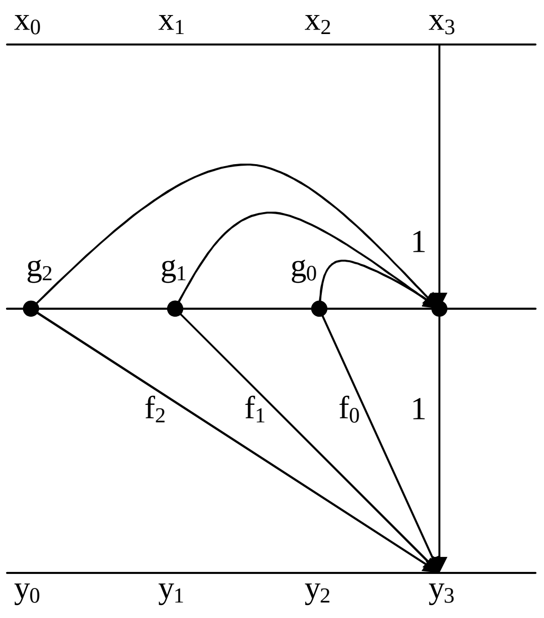
is an pole-zero filter which has both finite poles and zeroes.

ARMA filters are basically cascaded MA and AR filters.

There are 2 types[i] of ARMA filters:

Type 1 (MA $\rightarrow$ AR) **Type 2 (AR $\rightarrow$ MA)**





Type 2 filters have an advantage over Type 1 in that only one set of latches is required for their implementation.

When constructing an ARMA filter, the AR filter may be unstable. As long as the poles of the AR filter match the zeros of the MA filter, the resulting ARMA filter is stable. However, sometimes we may not be able to match the poles and the zeros perfectly. Some of the reasons are:

1. On computers, due to precision / truncation errors
2. Incapability of specifying the physical media (plant errors)

Given two vectors y and x, if we wanted to fit them together we would scale one of them by a scalar a. Using mathematical symbols we would write it as:

$$f(a) = \|y - ax\|_2^2$$
$$f(a) = (y - ax)^T (y - ax)$$
$$f(a) = y^T y - 2ay^T x + a^2 x^T x$$

This is a minimization problem where a has to be varied to minimize *f*

$$\frac{df(a)}{da} = 2ax^T x - 2y^T x = 0$$
$$\therefore a = \frac{y^T x}{x^T x}$$

f(a) is a parabola which can have a minimum only if the coefficient of the second degree term is greater than zero viz. $x^T x$. This is true for an upward facing parabola as this term dominates every other terms.

For a parabola ax^2+bx+c the minima is at $-b/2a$. This is obvious because the intersections with the x-axis are at $-b/a$ and b/a and the parabola is symmetric[ii].

Deconvolution Revisited

As we have seen before an LTI system can be represented as

$$Y = AX$$

Further, deconvolution is the process of finding the kernel/input given the output and the input/kernel. Hence we can view deconvolution as matrix inversion where in we need to find *X* given *Y* and *A* by finding A^{-1} and pre-multiplying the above equation.

A generalization of the above problem can be stated as

In most practical situations, *X* does not span the space of *Y* and hence there is no exact solution to the above equation. Thus we formulate the minimization problem as:

Given *Y* and *A*, we have to get *X* such that

$$f(X) = \|Y - AX\|_2 \text{ is minimum.}$$

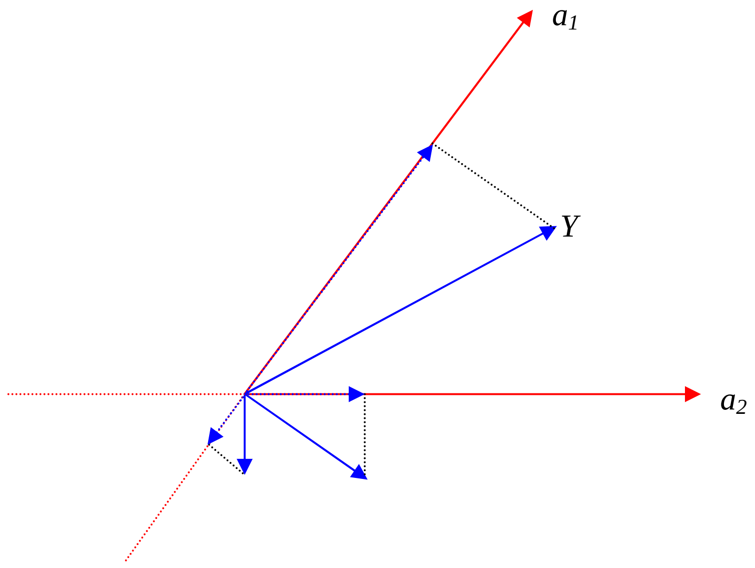
Where $\|U\|_2^2 = \sum U_i^2 = U^T U = \langle U, U \rangle$ (L_2 Norm)

This problem is very similar to the problem we encountered earlier. *Y*, *A* and *X* in the above equation correspond to *y*, *x*, and *a* in the earlier problem. There *a* was a scalar here *X* is a vector. We can see this as multiplying each multiplying each column of *A* (a vector) with each row of *X* (a scalar).

Thus it may seem that we have reduced this problem into multiple instances of the previous problem. However, this is not so because the columns of *A* do not form an orthogonal basis.

The problem can be seen when we consider the following figure in which *Y* is a vector in the plane of a_1 and a_2 (*A* spans *Y*). Let us try and adopt the method of the previous problem here. We project *Y* onto a_1 and project whatever is remaining onto a_2. We see that we are still left with some amount of *Y* and we have to repeat the same procedure again (and again…). Although this procedure converges, it takes a lot of time.

The figure below shows the first few steps in the projection and re-projection of *Y* along a_1 and a_2



Least Square Matrix Inversion

$$f(X) = \|Y - AX\|_2^2$$

$$f(X) = Y - \begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}$$

$$f(X) = (Y - AX)^T (Y - AX)$$

$$f(X) = (Y^T - X^T A^T)(Y - AX)$$

$$f(X) = X^T (A^T A) X - (Y^T AX) - (Y^T AX)^T + Y^T Y$$

$$f(x) = X^T \underbrace{(A^T A)}_{P} X - \underbrace{(2Y^T A)}_{Q^T} X + \underbrace{Y^T Y}_{R}$$

We write the above equation as

$f(X) = X^T PX + Q^T X + R$

where $P = A^T A$
$Q = -2A^T Y$
$R = Y^T Y$

$f(X)$ is a field representing an n-dimensional paraboloid[iii].

The above equation will have a minima only if $\forall x \; X^T PX > 0$
P is the positive definite, written as $P \succ 0$

The above equation using summations:

$$f(X) = \sum_i \sum_j P_{ij} X_i X_j + \sum_i Q_i$$

We take the partial derivative with respect to x_i in order to minimize $f(X)$

$$\frac{\partial f}{\partial x_i} = \sum_{j \neq i} (P_{ij} + P_{ji}) X_j + 2P_{ii} X_i + Q_i$$

$$\frac{\partial f}{\partial x_i} = \sum_{j} (P_{ij} + P_{ji}) X_j + Q_i$$

$$\nabla f = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix} = \begin{bmatrix} \sum_j (P_{1j} + P_{j1}) X_j + Q_1 \\ \sum_j (P_{1j} + P_{j1}) X_j + Q_1 \\ \vdots \\ \sum_j (P_{nj} + P_{jn}) X_j + Q_n \end{bmatrix}$$

$$\nabla f = \begin{bmatrix} \sum_j P_{1j} X_j \\ \sum_j P_{2j} X_j \\ \vdots \\ \sum_j P_{nj} X_j \end{bmatrix} + \begin{bmatrix} \sum_j P_{j1} X_j \\ \sum_j P_{j1} X_j \\ \vdots \\ \sum_j P_{jn} X_j \end{bmatrix} + \begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_n \end{bmatrix}$$

$$\nabla f = \left(\begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix} + \begin{bmatrix} P_{11} & P_{21} & \cdots & P_{n1} \\ P_{1} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix} \right) . \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} + \begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_n \end{bmatrix}$$

$$\nabla f = (P + P^T) X + Q$$

The minima of *f(X)* can be found by solving $\nabla f = \vec{0}$.

$$(P + P^T) X + Q = \vec{0}$$
$$\therefore (P + P^T) X = -Q$$
$$X = -(P + P^T)^{-1} Q$$

This is very similar to the solution *–b/2a* if we put Q as *b* and *P* as *a* (Assuming P is symmetric which it is in most practical cases).

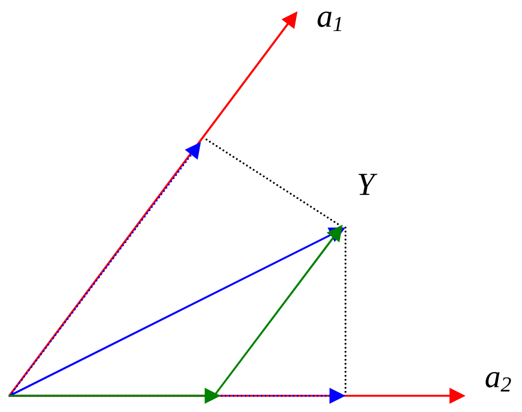
Substituting for *P* and *Q*

$$X = -2(A^T A + A^T A)^{-1} A^T Y$$
$$X = (A^T A)^{-1} A^T Y$$

$X = A^{\dagger} Y$

$A^{\dagger} = (A^T A)^{-1}$ is called the pseudo inverse[iv].

In the above equation A^T is the projection matrix. It projects Y onto the basis vectors of A. Finally $(A^TA)^{-1}$ converts the projections into linear combinands[v].



In the above figure, the red vectors are the basis, the blue vectors are the projections of Y and the green vectors are the linear combinands.

Let us review what we have done. Given the vectors *Y* and transformation we had to find *X* such that *Y* - *AX* was minimum. Usually Y has far more components than X. We have to tweak only a few parameters of the input vector X. For example,

$$\underbrace{\begin{bmatrix}\vdots\\\vdots\\\vdots\\\vdots\end{bmatrix}_{(4x1)}}_{Y} = \underbrace{\begin{bmatrix}\vdots&\vdots\\\vdots&\vdots\\\vdots&\vdots\\\vdots&\vdots\end{bmatrix}_{(4x2)}}_{A} \underbrace{\begin{bmatrix}\vdots\\\vdots\end{bmatrix}_{(2x1)}}_{X}$$

Here *X* is a 2-dimensional vector and *Y* is a 4-dimensional vector. *A* converts *X* from its coordinate system into the coordinate system of *Y*, thus making *Y* and *AX* comparable. AX is the linear combination of the bases of *A* weighted by the components of *X*. *AX* still lies in its plane of *X* while *Y* is outside that plane. Hence we cannot directly equate them. Instead we drop perpendiculars to the space of *A* and equate them. Dropping perpendiculars is done by pre-multiplying them with A^T

Hence, $A^T AX = A^T Y$

And $X = (A^TA)^{-1} A^T Y$

Note that this looks very similar to $a = \dfrac{y^T x}{x^T x}$ which we had found earlier.

Udayan's comments and corrections –

[i] Because of commutativity, type I and type II systems are equivalent. Thus, these are not two types of ARMA filters. Rather they are two ways of implementing them.

[ii] The parabola *we* are talking about is completely positive, and has no roots. But the $-b/2a$ formula holds for complex-root parabolas just like it does for real root parabolas.

[iii] "Quadratic surface"

[iv] $A^{\dagger} = (A^T A)^{-1} A^T$

[v] "Combinants"!!