DSP

Chapter-7 : Recursive Least Squares Algorithms

Marc Moonen
Dept. E.E./ESAT-STADIUS, KU Leuven
marc.moonen@esat.kuleuven.be
www.esat.kuleuven.be/stadius/

Part-III : Optimal & Adaptive Filters

Chapter-6 **Wieners Filters & the LMS Algorithm**

- Introduction / General Set-Up
- Applications
- Optimal Filtering: Wiener Filters
- Adaptive Filtering: LMS Algorithm

Chapter-7 **Recursive Least Squares Algorithms**

- Least Squares Estimation
- Recursive Least Squares (RLS)
- Square Root Algorithms
- Fast RLS Algorithms

1. Least Squares (LS) Estimation

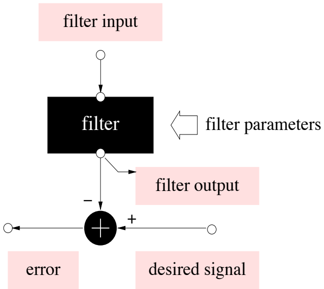
Prototype optimal/adaptive filter revisited

filter structure ?

→ FIR filters
(=pragmatic choice)

cost function ?

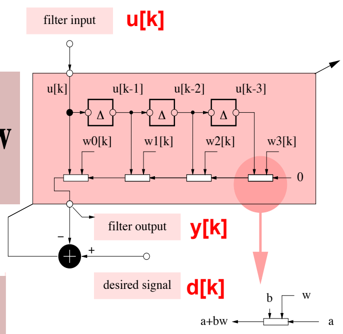
→ quadratic cost function
(=pragmatic choice)



1. Least Squares (LS) Estimation

FIR filters (=tapped-delay line filter/'transversal' filter)

$$y_k = \sum_{l=0}^{L} w_l \cdot u_{k-l} = \mathbf{w}^T \cdot \mathbf{u}_k = \mathbf{u}_k^T \cdot \mathbf{w}$$



where

$$\mathbf{w}^T = \begin{bmatrix} w_0 & w_0 & \ldots & w_L \end{bmatrix}$$

$$\mathbf{u}_k^T = \begin{bmatrix} u_k & u_{k-1} & \ldots & u_{k-L} \end{bmatrix}$$

PS: Shorthand notation u_k=u[k], y_k=y[k], d_k=d[k], e_k=e[k],
Filter coefficients ('weights') are w_l (replacing b_l of previous chapters)
For adaptive filters w_l also have a time index w_l[k]

1. Least Squares (LS) Estimation

Quadratic cost function

MMSE :

$$J_{MSE}(\mathbf{w}) = \mathrm{E}\left\{e_k^2\right\} = \mathrm{E}\left\{(d_k - y_k)^2\right\} = \mathrm{E}\left\{\left(d_k - \mathbf{u}_k^T\mathbf{w}\right)^2\right\}$$

Least-squares(LS) criterion :

if statistical info is not available, may use an alternative 'data-based' criterion...

$$J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} e_l^2 = \sum_{l=1}^{k}(d_l - y_l)^2 = \sum_{l=1}^{k}\left(d_l - \mathbf{u}_l^T\mathbf{w}\right)^2$$

Interpretation? : see below

1. Least Squares (LS) Estimation

filter input sequence : $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \ldots \ \mathbf{u}_k$

corresponding desired response sequence is : $d_1, d_2, d_3, \ldots, d_k$

$$\underbrace{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_k \end{bmatrix}}_{\text{error signal } \mathbf{e}} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}}_{\mathbf{d}} - \underbrace{\begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}}_{U} \cdot \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_L \end{bmatrix}}_{\mathbf{w}}$$

cost function $J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} e_l^2 = \|\mathbf{e}\|_2^2 = \|\mathbf{d} - U\mathbf{w}\|_2^2$

$\rightarrow$ *linear least squares problem* : $\min_{\mathbf{w}} \|\mathbf{d} - U\mathbf{w}\|_2^2$

1. Least Squares (LS) Estimation

$$J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} e_l^2 = \|\mathbf{e}\|_2^2 = \mathbf{e}^T.\mathbf{e} = \|\mathbf{d} - U\mathbf{w}\|_2^2$$

minimum obtained by setting gradient $= 0$:

$$0 = [\frac{\partial J_{LS}(\mathbf{w})}{\partial \mathbf{w}}]_{\mathbf{w}=\mathbf{w}_{LS}} = [\frac{\partial}{\partial \mathbf{w}}(\mathbf{d}^T\mathbf{d} + \mathbf{w}^T U^T U \mathbf{w} - 2\mathbf{w}^T U^T \mathbf{d})]_{\mathbf{w}=\mathbf{w}_{LS}}$$
$$= [2\underbrace{U^T U}_{\mathbb{X}_{uu}}\mathbf{w} - 2\underbrace{U^T \mathbf{d}}_{\mathbb{X}_{du}}]_{\mathbf{w}=\mathbf{w}_{LS}}$$

$$\mathbb{X}_{uu} \cdot \mathbf{w}_{LS} = \mathbb{X}_{du} \quad \rightarrow \quad \mathbf{w}_{LS} = \mathbb{X}_{uu}^{-1}\mathbb{X}_{du}$$

'Normal equations'
(L+1 equations in L+1 unknowns)

This is the 'Least Squares Solution'

1. Least Squares (LS) Estimation

Note : correspondences with Wiener filter theory ?

♣ estimate $\bar{\mathbb{X}}_{uu}$ and $\bar{\mathbb{X}}_{du}$ by time-averaging (ergodicity!)

$$\text{estimate}\{\bar{\aleph}_{uu}\} = \frac{1}{k}.\sum_{l=1}^{k}\mathbf{u}_l.\ \mathbf{u}_l^T = \frac{1}{k}.\ U^T U = \frac{1}{k}.\aleph_{uu}$$

$$\text{estimate}\{\bar{\aleph}_{du}\} = \frac{1}{k}.\sum_{l=1}^{k}\mathbf{u}_l.\ d_l = \frac{1}{k}.\ U^T \mathbf{d} = \frac{1}{k}.\aleph_{du}$$

leads to same optimal filter :

$$\text{estimate}\{\mathbf{w}_{WF}\} = (\frac{1}{k}\mathbb{X}_{uu})^{-1} \cdot (\frac{1}{k}\mathbb{X}_{du}) = \mathbb{X}_{uu}^{-1} \cdot \mathbb{X}_{du} = \mathbf{w}_{LS}$$

1. Least Squares (LS) Estimation

Note : correspondences with Wiener filter theory ? (continued)

♣ Furthermore (for ergodic processes!) :

$$\bar{\aleph}_{uu} = \lim_{k\to\infty} \frac{1}{k}.\sum_{l=1}^{k} \mathbf{u}_l.\,\mathbf{u}_l^T \;=\; \lim_{k\to\infty} \frac{1}{k}.\aleph_{uu}$$

$$\bar{\aleph}_{du} = \lim_{k\to\infty} \frac{1}{k}.\sum_{l=1}^{k} \mathbf{u}_l.\,d_l \;=\; \lim_{k\to\infty} \frac{1}{k}.\aleph_{du}$$

so that

$$\lim_{k\to\infty} \mathbf{w}_{LS} = \mathbf{w}_{WF}$$

2. Recursive Least Squares (RLS)

For a fixed data segment 1... k least squares problem is

$$\min_{\mathbf{w}_k} \left\| \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}}_{\mathbf{d}_k} - \underbrace{\begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}}_{U_k} . \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_L \end{bmatrix}}_{\mathbf{w}_k} \right\|_2^2$$

Matrices and vectors now with time index added

$$\mathbf{w}[k] = \aleph_{uu}[k]^{-1}.\aleph_{du}[k] = \left[U_k^T U_k\right]^{-1}.U_k^T \mathbf{d}_k$$

Wanted : recursive/adaptive algorithms

Can LS solution @ time k be computed from solution @ time k-1 ?

2.1 Standard RLS

Rank-1 updates

It is observed that $\aleph_{uu}[k] = \aleph_{uu}[k-1] + \mathbf{u}_k.\mathbf{u}_k^T$ (and $\aleph_{du}[k] = \aleph_{du}[k-1] + \mathbf{u}_k.d_k$)

The *matrix inversion lemma* states that (check 'matrix inversion lemma' in Wikipedia)

$$\aleph_{uu}[k]^{-1} = \aleph_{uu}[k-1]^{-1} - \left(\frac{1}{1+\mathbf{u}_k^T \aleph_{uu}[k-1]^{-1}\mathbf{u}_k}\right).\mathbf{k}_k\mathbf{k}_k^T \quad \text{with} \quad \mathbf{k}_k = \aleph_{uu}[k-1]^{-1}\mathbf{u}_k$$

With this it is proved that:

$$\mathbf{w}_{LS}[k] = \mathbf{w}_{LS}[k-1] + \underbrace{\overbrace{\aleph_{uu}[k]^{-1}\mathbf{u}_k}^{\text{'Kalman gain vector'}}}_{=\left(\frac{1}{1+\mathbf{u}_k^T\aleph_{uu}[k-1]^{-1}\mathbf{u}_k}\right).\mathbf{k}_k} .\overbrace{(d_k - \mathbf{u}_k^T\mathbf{w}_{LS}[k-1])}^{\text{'a priori residual'}}$$

= **standard recursive least squares (RLS) algorithm**

Remark : $O(L^2)$ instead of $O(L^3)$ *operations per time update*

Remark : *square-root algorithms with better numerical properties*
see below

2.2 Exponentially Weighted RLS

Exponentially weighted RLS : Goal is to give a smaller weight to 'older' data, i.e.

$$J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} \lambda^{2(k-l)} e_l^2$$

$0 < \lambda < 1$ is *weighting factor* or *forget factor*

$\frac{1}{1-\lambda}$ is a 'measure of the memory of the algorithm'

Which leads to…

$$\min_{\mathbf{w}_k} \left\| \underbrace{\begin{bmatrix} \lambda^{k-1}d_1 \\ \lambda^{k-2}d_2 \\ \vdots \\ \lambda^0 d_k \end{bmatrix}}_{\mathbf{d}_k} - \underbrace{\begin{bmatrix} \lambda^{k-1}\mathbf{u}_1^T \\ \lambda^{k-2}\mathbf{u}_2^T \\ \vdots \\ \lambda^0\mathbf{u}_k^T \end{bmatrix}}_{U_k} . \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_L \end{bmatrix}}_{\mathbf{w}_k} \right\|_2^2$$

$$\mathbf{w}_k = \aleph_{uu}[k]^{-1}.\aleph_{du}[k] = \left[U_k^T U_k\right]^{-1}.U_k^T\mathbf{d}_k$$

2.2 Exponentially Weighted RLS

It is observed that $\aleph_{uu}[k]=\lambda^2.\aleph_{uu}[k-1]+\mathbf{u}_k.\mathbf{u}_k^T$ (and $\aleph_{du}[k]=\lambda^2.\aleph_{du}[k-1]+\mathbf{u}_k.d_k$)

hence

$$\aleph_{uu}[k]^{-1}=\frac{1}{\lambda^2}\aleph_{uu}[k-1]^{-1}-\left(\frac{1}{1+\frac{1}{\lambda^2}\mathbf{u}_k^T\aleph_{uu}[k-1]^{-1}\mathbf{u}_k}\right).\mathbf{k}_k\mathbf{k}_k^T \quad \text{with} \quad \mathbf{k}_k=\frac{1}{\lambda^2}\aleph_{uu}[k-1]^{-1}\mathbf{u}_k$$

$$\mathbf{w}_{LS}[k]=\mathbf{w}_{LS}[k-1]+\aleph_{uu}[k]^{-1}\mathbf{u}_k.(d_k-\mathbf{u}_k^T\mathbf{w}_{LS}[k-1])$$

i.e. exponential weighting hardly changes RLS formulas.. (easy!)

3. Square-root Algorithms

- Standard RLS exhibits unstable roundoff error accumulation, hence not the algorithm of choice in practice
- Alternative algorithms ('square-root algorithms'), which have been proved to be stable numerically, are based on orthogonal matrix decompositions, namely QR decomposition (+ QR updating, inverse QR updating, see below)

3.1 QRD-based RLS Algorithms

QR Decomposition for LS estimation

least squares problem

$$\min_{\mathbf{w}} \|\mathbf{d} - U\mathbf{w}\|_2^2$$

'square-root algorithms' based on *QR decomposition (QRD)* :

$$\underbrace{U}_{k\mathrm{x}(L+1)} = \underbrace{Q}_{k\mathrm{x}k} \cdot \underbrace{\begin{bmatrix} R \\ 0 \end{bmatrix}}_{k\mathrm{x}(L+1)} = \underbrace{\tilde{Q}}_{Q(:,1:L+1)} \cdot \underbrace{R}_{(L+1)\mathrm{x}(L+1)}$$

square * rectangular   rectangular * square

$Q^T \cdot Q = I$, Q is orthogonal R is upper triangular

Everything you need to know about QR decomposition

Example :

$$\overbrace{\begin{bmatrix} 1 & 6 & 10 \\ 2 & 7 & -11 \\ 3 & 8 & 12 \\ 4 & 9 & -13 \end{bmatrix}}^{U} = \overbrace{\begin{bmatrix} 0.182 & 0.816 & 0.174 \\ 0.365 & 0.408 & -0.619 \\ 0.547 & 0 & 0.716 \\ 0.730 & -0.408 & -0.270 \end{bmatrix}}^{\tilde{Q}} \cdot \overbrace{\begin{bmatrix} 5.477 & 14.605 & -5.112 \\ 0 & 4.082 & 8.981 \\ 0 & 0 & 20.668 \end{bmatrix}}^{R}$$

Remark : QRD $\approx$ Gram-Schmidt

Remark : $U^T \cdot U = R^T \cdot R$

R is *Cholesky factor* or *square-root* of $U^T \cdot U$

$\rightarrow$ *'square-root' algorithms* !

3.1 QRD-based RLS Algorithms

QRD for LS estimation

if

$$\underbrace{U}_{k\times(L+1)} = \underbrace{Q}_{k\times k} \cdot \underbrace{\begin{bmatrix} R \\ 0 \end{bmatrix}}_{k\times(L+1)} = \underbrace{\tilde{Q}}_{Q(:,1:L+1)} \cdot \underbrace{R}_{(L+1)\times(L+1)}$$

then

$$\min_{\mathbf{w}} \left\| \mathbf{d} - U\mathbf{w} \right\|_2^2 \overset{(**)}{=} \min_{\mathbf{w}} \left\| Q^T(\mathbf{d} - U\mathbf{w}) \right\|_2^2 = \min_{\mathbf{w}} \left\| \begin{bmatrix} \mathbf{z} \\ * \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} \mathbf{w} \right\|_2^2$$

with this

triangular backsubstitution

$$R \cdot \mathbf{w}_{LS} = z \Rightarrow \mathbf{w}_{LS} = R^{-1} \cdot z = [\tilde{Q}^T U]^{-1} \cdot \tilde{Q}^T \mathbf{d}$$

This is a numerically better way of computing the LS solution, better than $\mathbf{w}_{LS} = \left[U^T U\right]^{-1} \cdot U^T \mathbf{d}$

(**) orthogonal transformation preserves norm

3.1 QRD-based RLS Algorithms

QR-updating for RLS estimation

Assume we have computed the QRD at time *k-1*

$$\begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \end{bmatrix} = \tilde{Q}[k-1]^T \cdot \begin{bmatrix} U_{k-1} & \mathbf{d}_{k-1} \end{bmatrix}$$

The corresponding LS solution is $\mathbf{w}_{LS}[k-1] = R[k-1]^{-1} \cdot z[k-1]$

Our aim is to update the QRD into

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \end{bmatrix} = \tilde{Q}[k]^T \cdot \begin{bmatrix} U_k & \mathbf{d}_k \end{bmatrix}$$

and then compute $\mathbf{w}_{LS}[k] = R[k]^{-1} \cdot z[k]$

3.1 QRD-based RLS Algorithms

QR-updating for RLS estimation

It is proved that the relevant QRD-updating problem is

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots0 & * \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

PS: This is based on a QR-factorization as follows:

$$\underbrace{\begin{bmatrix} R[k-1] \\ \mathbf{u}_k^T \end{bmatrix}}_{(L+2)\times(L+1)} = \underbrace{Q[k]}_{(L+2)\times(L+2)} \cdot \underbrace{\begin{bmatrix} R[k] \\ 0 \end{bmatrix}}_{(L+2)\times(L+1)}$$

3.1 QRD-based RLS Algorithms

QR-updating for RLS estimation

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots0 & * \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

$\mathbf{w}_{LS}[k] = R[k]^{-1} \cdot z[k]$ ='triangular backsubstitution'

= *square-root (information matrix) RLS*

Remark . with exponential weighting

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots0 & * \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} \lambda \cdot R[k-1] & \lambda \cdot \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

3.1 QRD-based RLS Algorithms

QRD updating

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots 0 & * \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

basic tool is **Givens rotation**

$$G_{i,j,\theta} \stackrel{\text{def}}{=} \begin{array}{cc} \begin{array}{ccccc} & i & & j & \\ & \downarrow & & \downarrow & \end{array} & \\ \begin{bmatrix} I_{i-1} & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & I_{j-i-1} & 0 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & I_{m-j} \end{bmatrix} & \begin{array}{c} \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \end{array} \end{array}$$

3.1 QRD-based RLS Algorithms

QRD updating

Givens rotation applied to a vector $\tilde{\mathbf{x}} = G_{i,j,\theta} \cdot \mathbf{x}$:

$$\begin{aligned} \tilde{x}_i &= \cos\theta \cdot x_i + \sin\theta \cdot x_j \\ \tilde{x}_j &= -\sin\theta \cdot x_i + \cos\theta \cdot x_j \end{aligned}$$

$$\tilde{x}_l = x_l \quad \text{for} \quad l \neq i, j$$

$$\tilde{x}_j = 0 \text{ iff } \tan\theta = \frac{x_j}{x_i} \text{ !}$$

3.1 QRD-based RLS Algorithms

QRD updating

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots0 & * \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

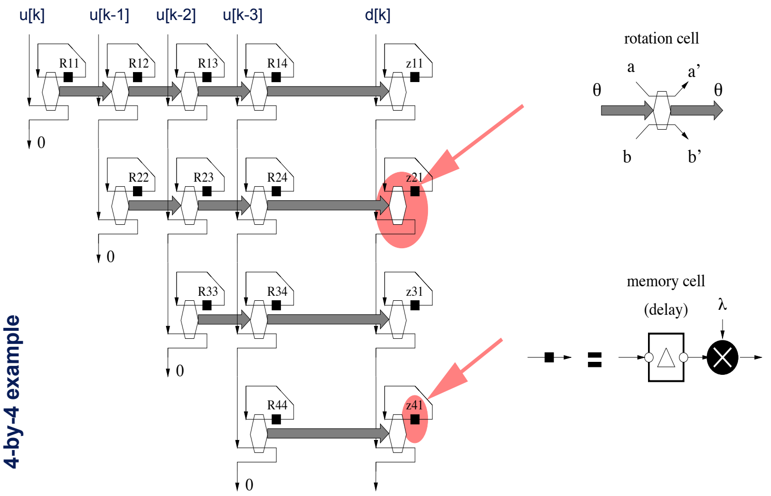
$Q[k]$ is constructed as a product/sequence of Givens transformations

3-by-3 example

$$\left[\begin{array}{ccc|c} x & x & x & x \\ & x & x & x \\ & & x & x \\ \hline x & x & x & x \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} x & x & x & x \\ & x & x & x \\ & & x & x \\ \hline & x & x & x \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} x & x & x & x \\ & x & x & x \\ & & x & x \\ \hline & & x & x \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} x & x & x & x \\ & x & x & x \\ & & x & x \\ \hline & & & \star \end{array}\right]$$

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots0 & * \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

4-by-4 example



3.1 QRD-based RLS Algorithms

Residual extraction

$$\begin{bmatrix} R[k] & \mathbf{z}[k] \\ 0\cdots0 & \varepsilon \end{bmatrix} = Q[k]^T \cdot \begin{bmatrix} R[k-1] & \mathbf{z}[k-1] \\ \mathbf{u}_k^T & d_k \end{bmatrix}$$

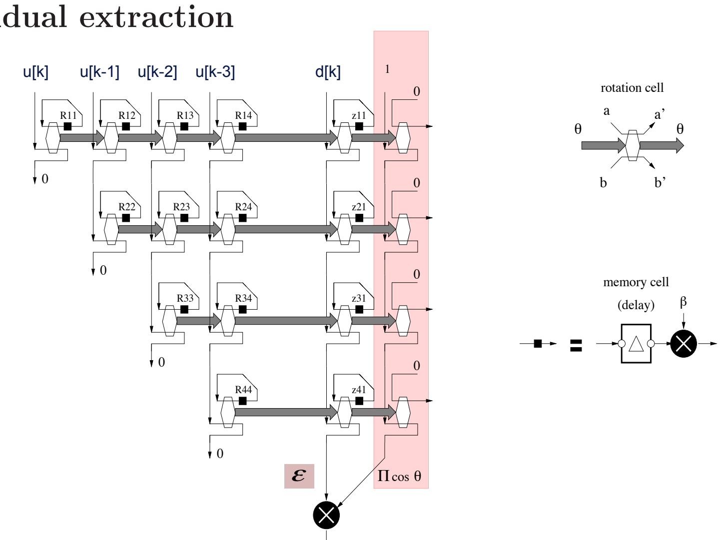
From this it is proved that the 'a posteriori residual' is

$$d_k - \mathbf{u}_k^T \mathbf{w}_{LS}[k] = \varepsilon \cdot \prod_{i=1}^{L+1} \cos(\theta_i)$$

and the 'a priori residual' is

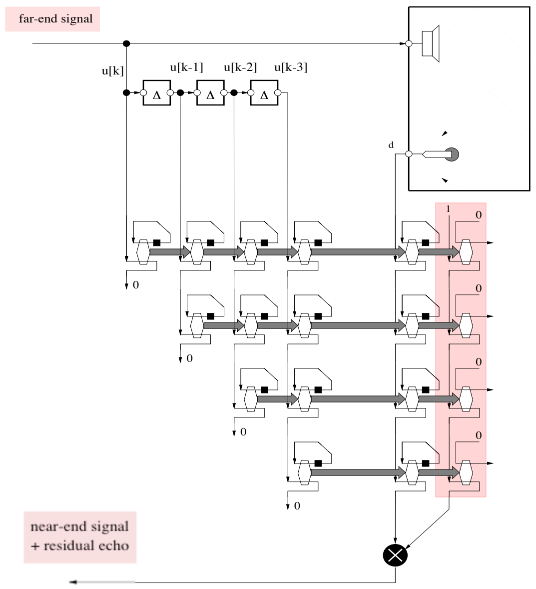
$$d_k - \mathbf{u}_k^T \mathbf{w}_{LS}[k-1] = \frac{\varepsilon}{\prod_{i=1}^{L+1} \cos(\theta_i)}$$

Residual extraction



$$d_k - \mathbf{u}_k^T \mathbf{w}_{LS}[k] = \varepsilon \cdot \prod_{i=1}^{L+1} \cos(\theta_i)$$

Example



Fast Recursive Least Squares Algorithms

RLS and square-root RLS : $O(L^2)$ per time update

When the adaptive filter is an FIR filter, the computational cost may be reduced to $O(L)$ per time update, by exploiting the time-shift structure of the input vectors/signals !

Here :

- **QRD least squares lattice (QRD-LSL)**

Other :

- **Least-squares lattice (LSL)**
- **'Fast QR'**
- **Fast transversal filter (FTF)**

Fast Recursive Least Squares Algorithms

Preliminaries

- vast literature available on *fast least squares algorithms*
- the derivation of fast algorithms is *highly* mathematical (see page 31)
- ~~we show how fast (QRD-based) algorithms can be derived using *signal flow graph (SFG) manipulation*~~
- In doing so we provide additional insight to the algorithmic structure

Fast Recursive Least Squares Algorithms

Example (headache?)

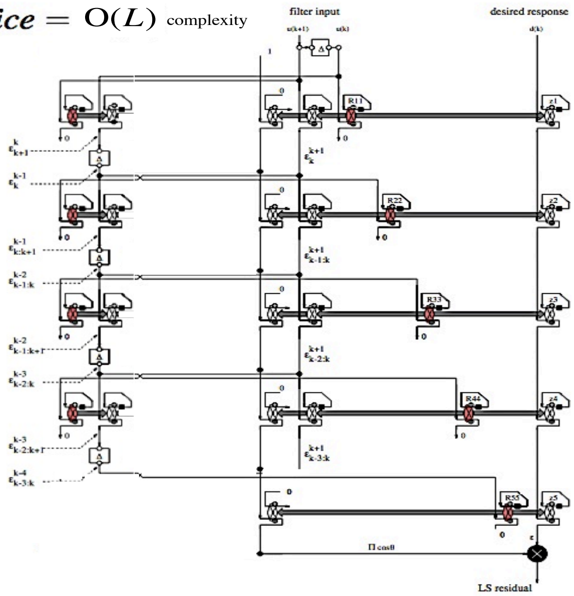
See p.31 for a signal flow graph of this

9.2.1. QRD-based Least Squares Lattice algorithm.

START
INITIALISE {all variables} := 0;
FOR n FROM 1 DO
LET $\alpha_{f,0}(n) := x(n)$; $\alpha_{b,0}(n-1) := x(n-1)$; $\alpha_0(n-1) := y(n-1)$; $\gamma_0(n-1) := 1$;
FOR q FROM 1 TO p DO
LET $\varepsilon_{b,q-1}(n-1) := \sqrt{(\beta\varepsilon_{b,q-1}(n-2))^2 + |\alpha_{b,q-1}(n-1)|^2}$;
IF $\varepsilon_{b,q-1}(n-1) = 0$ THEN LET $c_{f,q} := 1$; $s_{f,q} := 0$
ELSE LET $c_{f,q} := \beta\varepsilon_{b,q-1}(n-2) / \varepsilon_{b,q-1}(n-1)$; $s_{f,q} := \alpha_{b,q-1}(n-1) / \varepsilon_{b,q-1}(n-1)$
END_IF;
LET $\mu_{f,q-1}(n) := c_{f,q}\,\beta\mu_{f,q-1}(n-1) + s^*_{f,q}\,\alpha_{f,q-1}(n)$;
$\alpha_{f,q}(n) := c_{f,q}\,\alpha_{f,q-1}(n) - s_{f,q}\,\beta\mu_{f,q-1}(n-1)$;
$\mu_{q-1}(n-1) := c_{f,q}\,\beta\mu_{q-1}(n-2) + s^*_{f,q}\,\alpha_{q-1}(n-1)$;
$\alpha_q(n-1) := c_{f,q}\,\alpha_{q-1}(n-1) - s_{f,q}\,\beta\mu_{q-1}(n-2)$;
$\gamma_q(n-1) := c_{f,q}\,\gamma_{q-1}(n-1)$;
COMMENT prediction residual $e_{f,p}(n,n) = \gamma_q(n-1)\,\alpha_{f,q}(n)$ COMMENT
$e_p(n-1,n-1) = \gamma_q(n-1)\,\alpha_q(n-1)$ COMMENT q-th order filtered residual COMMENT
LET $\varepsilon_{f,q-1}(n) := \sqrt{(\beta\varepsilon_{f,q-1}(n-1))^2 + |\alpha_{f,q-1}(n)|^2}$;
IF $\varepsilon_{f,q-1}(n) = 0$ THEN LET $c_{b,q} := 1$; $s_{b,q} := 0$
ELSE LET $c_{b,q} := \beta\varepsilon_{f,q-1}(n-1) / \varepsilon_{f,q-1}(n)$; $s_{b,q} := \alpha_{f,q-1}(n) / \varepsilon_{f,q-1}(n)$
END_IF;
LET $\mu_{b,q-1}(n-1) := c_{b,q}\,\beta\mu_{b,q-1}(n-2) + s^*_{b,q}\,\alpha_{b,q-1}(n-1)$;
$\alpha_{b,q}(n) := c_{b,q}\,\alpha_{b,q-1}(n-1) - s_{b,q}\,\beta\mu_{b,q-1}(n-2)$;
COMMENT $\gamma_q(n) := c_{b,q}\,\gamma_{q-1}(n-1)$; backward prediction residual $e_{b,q}(n,n) := \gamma_q(n)\,\alpha_{b,q}(n)$ COMMENT
END_DO
END_DO
FINISH

Example *QRD Lattice* $= \mathrm{O}(L)$ complexity

Derivation omitted…



See p.31 for a signal flow graph of this

Fast Recursive Least Squares Algorithms

Conclusion

- Many 'fast' RLS algorithms available (QRD-lattice, LSL, Fast-QR, FTF,...)
- High performance (*cfr.* RLS) at low cost ($\mathrm{O}(L)$), *i.e.* almost as cheap as LMS)
- Derivation is very mathematical...
- ..but SFG's may help.