4F7 Adaptive Filters (and Spectrum Estimation)

Recursive Least Squares (RLS) Algorithm

Sumeetpal Singh
Engineering Department
Email : sss40@eng.cam.ac.uk

1 Aim

- The Wiener filter solved min $J(\mathbf{h})$ where $J(\mathbf{h}) = E\left\{e^2(n)\right\}$ and

$$e(n) = d(n) - \mathbf{h}^{\mathsf{T}}\mathbf{u}(n)$$

- Then we derived the SD, LMS and NLMS to solve this same problem
- Today we will derive the Recursive Least Squares (RLS) to minimise the following cost function at time n,

$$J(\mathbf{h},n) = \sum_{k=0}^{n} \lambda^{n-k} e^2(k)$$

- The minimiser $\mathbf{h}_{\mathrm{opt}}(n)$ will be our filter
- We will then derive a recursion for $\mathbf{h}_{\mathrm{opt}}(n)$, i.e., relating $\mathbf{h}_{\mathrm{opt}}(n+1)$ to $\mathbf{h}_{\mathrm{opt}}(n)$
- The main point here is that the cost function is time varying and there is no expectation in the cost function. A random cost function and $\mathbf{h}_{\mathrm{opt}}(n)$ is as well random

2 Outline

- Derive RLS
- Initialising the RLS
- Simulation examples

3 The RLS algorithm

- Want to minimise the cost function

$$J(\mathbf{h},n) = \sum_{k=0}^{n} \lambda^{n-k} e^2(k)$$

where $e(k) = d(k) - \mathbf{h}^{\mathsf{T}}\mathbf{u}(k)$ and, $0 < \lambda \leq 1$. λ is a called the **forgetting factor**

- If $\lambda = 1$, one notices that

$$\frac{1}{n+1} J(\mathbf{h},n)\Big|_{\lambda=1} = \frac{1}{n+1}\sum_{k=0}^{n} e^2(k)$$

- This is a sample average and should converge to $E\left\{e^2(k)\right\}$ or $J(\mathbf{h})$ in the SD and LMS lectures.
- So, we can now argue that the RLS solution and the Wiener filter coincide when $\lambda = 1$. Note that the RLS is solving for the minimiser of $J(\mathbf{h},n)$ at time n. Dividing this quantity by $n+1$ does not change the minimizer. Since $(n+1)^{-1}J(\mathbf{h},n)$ tends to the Wiener filter cost function in the limit, the RLS solution should agree with the Wiener filter in the limit
- For $\lambda < 1$, $J(\mathbf{h},n)$ regards the past errors as less important since they are weighted by λ^{n-k}
  – the smaller λ is, the quicker the RLS will respond if the Wiener filter is time varying (see simulations)

- Solve for the RLS solution by setting the derivative to zero:

$$J(\mathbf{h},n) = \sum_{k=0}^{n} \lambda^{n-k} \left(d(k) - \mathbf{h}^{\mathtt{T}}\mathbf{u}(k) \right)^2$$

$$\nabla J(\mathbf{h},n) = -2 \sum_{k=0}^{n} \lambda^{n-k} \left(d(k) - \mathbf{h}^{\mathtt{T}}\mathbf{u}(k) \right) \mathbf{u}(k)$$

Thus

$$\mathbf{h}_{\mathrm{opt}}(n) = \left[\sum_{k=0}^{n} \lambda^{n-k} \mathbf{u}(k) \mathbf{u}^{\mathrm{T}}(k) \right]^{-1} \times \sum_{k=0}^{n} \lambda^{n-k} \mathbf{u}(k) d(k)$$

- Note that the RLS agrees with Wiener when $\lambda = 1$ since

$$\mathbf{h}_{\mathrm{opt}}(n) = \left[\frac{1}{n+1} \sum_{k=0}^{n} \mathbf{u}(k) \mathbf{u}^{\mathrm{T}}(k) \right]^{-1} \times \frac{1}{n+1} \sum_{k=0}^{n} \mathbf{u}(k) d(k)$$

and under stationarity assumptions,

$$\left[\frac{1}{n+1}\sum_{k=0}^{n}\mathbf{u}\left(k\right)\mathbf{u}^{\mathrm{T}}\left(k\right)\right]^{-1} \rightarrow \mathbf{R}^{-1},$$

$$\frac{1}{n+1}\sum_{k=0}^{n}\mathbf{u}\left(k\right)d\left(k\right) \rightarrow \mathbf{p},$$

and so $\lim_{n\rightarrow\infty}\mathbf{h}_{\mathrm{opt}}\left(n\right)$ is the Wiener filter

4 RLS update rule

- Firstly note that if

$$\mathbf{R}(n) = \sum_{k=0}^{n} \lambda^{n-k} \mathbf{u}(k)\, \mathbf{u}^{\mathrm{T}}(k)$$

$$\mathbf{p}(n) = \sum_{k=0}^{n} \lambda^{n-k} \mathbf{u}(k)\, d(k)$$

then

$$\mathbf{R}(n) = \lambda \mathbf{R}(n-1) + \mathbf{u}(n)\, \mathbf{u}^{\mathrm{T}}(n),$$
$$\mathbf{p}(n) = \lambda \mathbf{p}(n-1) + \mathbf{u}(n)\, d(n)$$

- At time n we are seeking the solution to $\mathbf{R}(n)\, \mathbf{h}(n) = \mathbf{p}(n)$, which has a computational complexity of $O\left(M^3\right)$ because of the matrix inversion
- Applying a well known result in matrix algebra, called the **matrix**

inversion lemma yields

$$\mathbf{R}^{-1}(n) = \lambda^{-1}\mathbf{R}^{-1}(n-1) - \frac{\lambda^{-2}\mathbf{R}^{-1}(n-1)\,\mathbf{u}(n)\,\mathbf{u}^{\mathrm{T}}(n)\,\mathbf{R}^{-1}(n-1)}{1+\lambda^{-1}\mathbf{u}^{\mathrm{T}}(n)\,\mathbf{R}^{-1}(n-1)\,\mathbf{u}(n)}$$

- Let $\mathbf{A}$ and $\mathbf{B}$ be two symmetric positive definite matrices of dimension $M \times M$ and such that

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}^{-1}\mathbf{C}^{\mathrm{T}}$$

where $\mathbf{D}$ is a symmetric positive definite matrix of dimension $L \times L$ and $\mathbf{C}$ is a matrix of dimension $M \times L$. The inverse matrix is given by (check it)

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

For our problem, one sets

$$\mathbf{A} = \mathbf{R}(n),\ \mathbf{B}^{-1} = \lambda\mathbf{R}(n-1),\ \mathbf{C} = \mathbf{u}(n),\ \mathbf{D} = 1$$

- This helps because since there are only matrix multiplications involved, the computational complexity is $O\left(M^2\right)$

- To get the RLS recursion, let $\mathbf{S}(n) = \mathbf{R}^{-1}(n)$. We obtain,

$$\alpha(n) = d(n) - \mathbf{u}^{\mathrm{T}}(n)\,\mathbf{h}(n-1) \quad \text{(predicted error)}$$

$$\mathbf{g}(n) = \left(\lambda + \mathbf{u}^{\mathrm{T}}(n)\,\mathbf{S}(n-1)\,\mathbf{u}(n)\right)^{-1}\mathbf{S}(n-1)\,\mathbf{u}(n)$$
$$\text{(the gain)}$$

$$\mathbf{S}(n) = \lambda^{-1}\left(\mathbf{I} - \mathbf{g}(n)\,\mathbf{u}^{\mathrm{T}}(n)\right)\mathbf{S}(n-1)$$
$$\text{(inverse covariance)}$$

$$\begin{aligned}\mathbf{h}(n) &= \mathbf{h}(n-1) + \mathbf{g}(n)\,\alpha(n)\\ &= \mathbf{h}(n-1) + \mathbf{S}(n)\,\mathbf{u}(n)\,\alpha(n) \quad \text{(update)}\end{aligned}$$

- Last line using $\mathbf{g}(n) = \mathbf{S}(n)\,\mathbf{u}(n)$. Computational complexity is $O\left(M^2\right)$ while LMS was $O(M)$
- In the LMS, we had μ instead of $\mathbf{S}(n)$

5 Initializing the RLS

- To initialize the RLS algorithm at time $n = 0$, we need $\mathbf{h}(-1)$ and $\mathbf{S}(-1) = \mathbf{R}^{-1}(-1)$
- We could of course wait long enough until $\mathbf{R}(n)$ is invertible and then initialize the algorithm with $\mathbf{S}(-1) = \mathbf{R}(n)^{-1}$ and $\mathbf{h}(-1) = \mathbf{R}(n)^{-1}\mathbf{p}(n)$
- This is called **exact** initialization
- Another way that doesn't have to wait for samples is as follows. Choose a small positive constant δ and set $\mathbf{S}(-1) = \delta^{-1}\mathbf{I}$, $\mathbf{h}(-1) = 0$
- For large n, $\lambda^n \delta$ is small and

$$\begin{aligned}\mathbf{R}(n) &= \lambda^{n+1}\delta\mathbf{I} + \sum_{k=0}^{n} \lambda^{n-k}\mathbf{u}(k)\,\mathbf{u}^{\mathrm{T}}(k) \\ &\approx \sum_{k=0}^{n} \lambda^{n-k}\mathbf{u}(k)\,\mathbf{u}^{\mathrm{T}}(k)\end{aligned}$$

- $\mathbf{h}(n) \neq \mathbf{h}_{\mathrm{opt}}(n)$ but is equal asymptotically

- Compare performance of RLS and LMS by running code on webpage. Using $\lambda < 1$ has better tracking performance in a non-stationary environment

	LMS	**RLS**
Free parameters	M, μ	M, λ, δ
Comp. complexity	$O(M)$	$O\left(M^2\right)$
Stationary environment	$\mathbf{h}(n) \nrightarrow \mathbf{h}_{\mathrm{true}}$	$\mathbf{h}(n) \rightarrow \mathbf{h}_{\mathrm{true}}$ for $\lambda = 1$
$E\left\{\mathbf{u}(k)\,\mathbf{u}^{\mathrm{T}}(k)\right\}$ sensitivity	High	Low

(where $\mathbf{h}_{\mathrm{true}}$ is Wiener filter)