Lectures 3

Recursive (On-line) Identification Methods

- Recursive Least Squares (RLS) Methods
- Forgetting Factor and Tracking Time-Varying Parameters
- Identification condition problem
- Computational Aspects

Course Outline

- Introduction and overview on system identification
- Non-recursive (off-line) identification methods
- *Recursive (on-line) identification methods (I)*
- Recursive (on-line) identification methods (II)
- Practical aspects and applications of system identification

Why?

Why is recursive identification of interest?

- On-line Estimation.
- Adaptive Systems.
- Time Varying Parameters.
- Fault Detection and Diagnosis.
- Simulation.

How?

How do we estimate time-varying parameters?

- Update the model regularly (once every sampling instant)
- Make use of previous calculations in an efficient manner.
- The basic procedure is to modify the corresponding off-line method, *e.g.*, the block/batch least squares method, the prediction error method.

Desirable Properties

We desire our recursive algorithms to have the following properties:

- Fast convergence.
- Consistent estimates (time-invariant models).
- Good tracking (for time-varying parameters, e.g. in the event of fault occurrence or operating condition changes).
- Computationally simple (perform all calculations during one sampling interval).

Trade-offs

No algorithm is perfect. The design is always based on trade-offs, such as:

- Convergence versus tracking.
- Computational complexity versus accuracy.

Recursive Least Squares Method (RLS)

$$\hat{\boldsymbol{\theta}}(t) = \arg\min_{\boldsymbol{\theta}} V_t(\boldsymbol{\theta}), \quad V_t(\boldsymbol{\theta}) = \sum_{k=1}^{t} \varepsilon^2(k)$$

where $\varepsilon(k) = y(k) - \boldsymbol{\varphi}^T(k)\boldsymbol{\theta}$. The solution reads:

$$\hat{\boldsymbol{\theta}}(t) = \boldsymbol{R}_t^{-1}\boldsymbol{r}_t$$

where

$$\boldsymbol{R}_t = \sum_{k=1}^{t} \boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k), \qquad \boldsymbol{r}_t = \sum_{k=1}^{t} \boldsymbol{\varphi}(k)y(k)$$

- The criterion function $V_t(\boldsymbol{\theta})$ changes every time step, hence the estimate $\hat{\boldsymbol{\theta}}(t)$ changes every time step.
- How can we find a recursive implementation of $\hat{\boldsymbol{\theta}}(t)$?

RLS

Algorithm:

At time $t = 0$: Choose initial values of $\hat{\boldsymbol{\theta}}(0)$ and $\boldsymbol{P}(0)$

At each sampling instant, update $\boldsymbol{\varphi}(t)$ and compute

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + \boldsymbol{K}(t)\varepsilon(t)$$

$$\varepsilon(t) = y(t) - \boldsymbol{\varphi}^T(t)\hat{\boldsymbol{\theta}}(t-1)$$

$$\boldsymbol{K}(t) = \boldsymbol{P}(t)\boldsymbol{\varphi}(t)$$

$$\boldsymbol{P}(t) = \left[\boldsymbol{P}(t-1) - \frac{\boldsymbol{P}(t-1)\boldsymbol{\varphi}(t)\boldsymbol{\varphi}^T(t)\boldsymbol{P}(t-1)}{1 + \boldsymbol{\varphi}^T(t)\boldsymbol{P}(t-1)\boldsymbol{\varphi}(t)}\right]$$

Question: How to obtain/derive this recursive version of LS from the block/batch LS?

Tracking

How do we handle time-varying parameters? — two ways:

- Postulate a time-varying model for the parameters. Typically we let the parameters vary according to a random walk and use the Kalman filter as an estimator.
- Modify the cost function so that we gradually forget old data. Hence, the model is fitted to the most recent data (the parameters are adapted to describe the newest data).

- Modified cost function:

$$\hat{\boldsymbol{\theta}}(t) = \arg\min_{\boldsymbol{\theta}} V_t(\boldsymbol{\theta}), \quad V_t(\boldsymbol{\theta}) = \sum_{k=1}^{t} \beta(t,k)\varepsilon^2(k)$$

- Suppose that the weighting function $\beta(t,k)$ satisfies

$$\beta(t,k) = \lambda(t)\beta(t-1,k), \quad 0 \le k < t$$
$$\beta(t,t) = 1$$

A common choice is to let $\lambda(t) = \lambda$, where λ is referred to as a so-called *forgetting factor*. In this case we get:

$$\beta(t,k) = \lambda^{t-k}, \quad 0 < \lambda \le 1$$

- $\lambda = 1$ corresponds to the standard RLS.

Weighted RLS

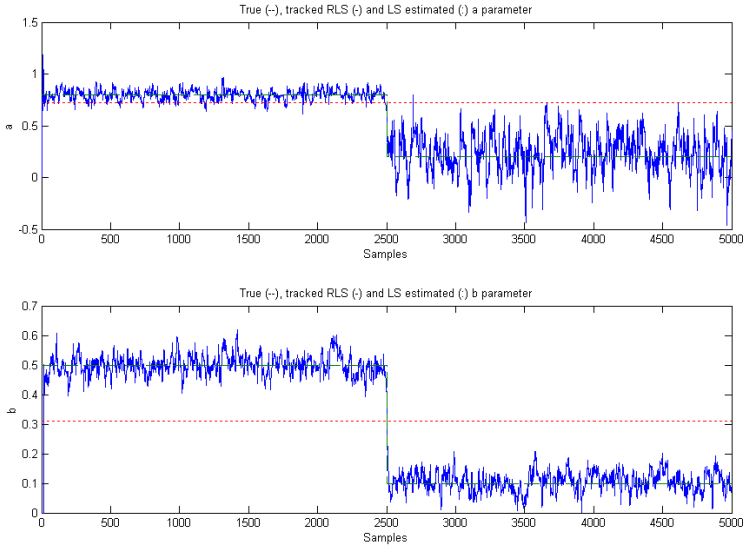
Algorithm:

At time $t = 0$: Choose initial values of $\hat{\boldsymbol{\theta}}(0)$ and $\boldsymbol{P}(0)$

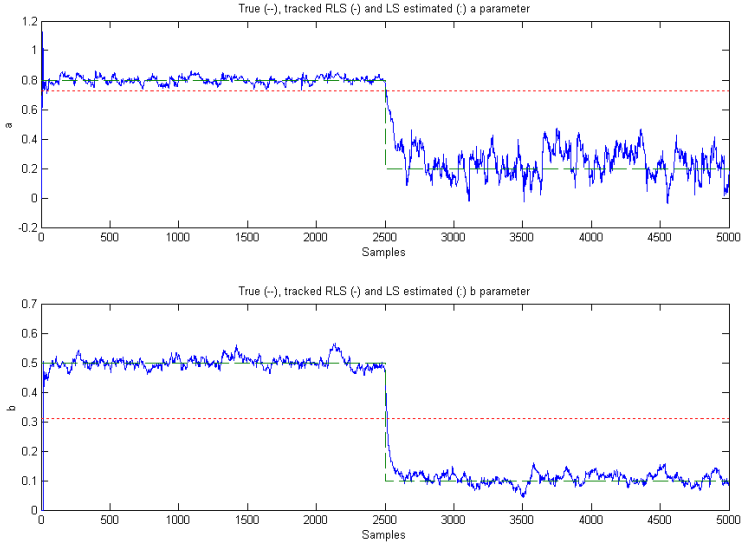
At each sampling instant, update $\boldsymbol{\varphi}(t)$ and compute

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + \boldsymbol{K}(t)\varepsilon(t)$$
$$\varepsilon(t) = y(t) - \boldsymbol{\varphi}^T(t)\hat{\boldsymbol{\theta}}(t-1)$$
$$\boldsymbol{K}(t) = \boldsymbol{P}(t)\boldsymbol{\varphi}(t)$$
$$\boldsymbol{P}(t) = \frac{1}{\lambda(t)}\left[\boldsymbol{P}(t-1) - \frac{\boldsymbol{P}(t-1)\boldsymbol{\varphi}(t)\boldsymbol{\varphi}^T(t)\boldsymbol{P}(t-1)}{\lambda(t) + \boldsymbol{\varphi}^T(t)\boldsymbol{P}(t-1)\boldsymbol{\varphi}(t)}\right]$$

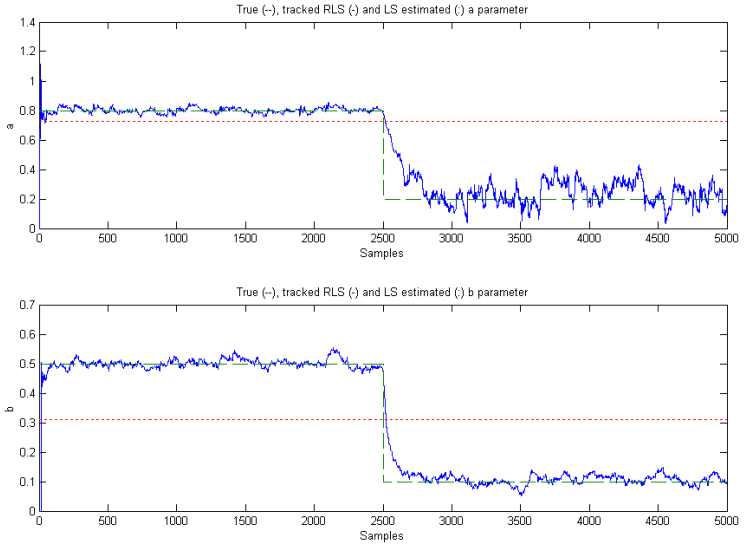
$\lambda = 0.90$



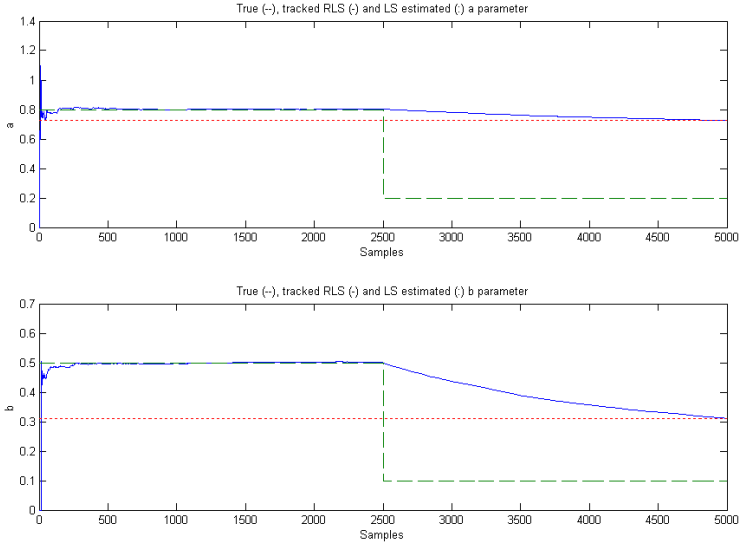
$\lambda = 0.97$



$\lambda = 0.98$



$\lambda = 1.0$



```
%%%
%%% Exercise 2 for Lecture 3.
%%%

clear all, close all

N = 5000;   % Sample number
Ts = 1;     % Sampling time
T  = [1:N]';

randn('state',1);
e = randn(N,1); % Equation error
sigma_e = 0.1;
e = sigma_e*(e - mean(e))/std(e);

u = idinput(N,'prbs',[0 1],[0 1]); % Identification deterministic input

tc = round(N/2);
u1 = u(1:tc,1);
u2 = u(tc+1:N,1);
e1 = e(1:tc,1);
e2 = e(tc+1:N,1);

a1 =  0.8;
b1 =  0.5;
a2 =  0.2;
b2 =  0.1;

th1_star = [a1 b1];
th2_star = [a2 b2];

d = length(th1_star);

    G1z = tf([0 b1],[1 a1],Ts); % Deterministic ARX transfer function
    H1z = tf([0 1],[1 a1],Ts); % Stochastic ARX transfer function

    G2z = tf([0 b2],[1 a2],Ts); % Deterministic ARX transfer function
    H2z = tf([0 1],[1 a2],Ts); % Stochastic ARX transfer function


y1 = lsim([G1z H1z],[u1 e1]); % ARX 1st model simulation
y2 = lsim([G2z H2z],[u2 e2]); % ARX 2nd model simulation

y = [y1;y2];

P0 = 100*eye(d);
th0 = zeros(1,d);
lam = 1.5; % 0.98

thm   = rarx([y u],[1 1 1],'ff',lam,th0,P0);
m_arx = arx([y u],[1 1 1]); [A,B,dA,dB] = arxdata(m_arx);

am = [a1*ones(size(u1)); a2*ones(size(u2))];
bm = [b1*ones(size(u1)); b2*ones(size(u2))];


figure, subplot(211), plot(T,thm(:,1),'-',T,am,'--'), xlabel('Samples'), ylabel('a'),
       title('True (--) and tracked RLS (-) a parameter')
       subplot(212), plot(T,thm(:,2),'-',T,bm,'--'), xlabel('Samples'), ylabel('b')
       title('True (--) and tracked RLS (-) b parameters')


figure, subplot(211), plot(T,thm(:,1),'-',T,am,'--',T,A(2)*ones(size(T)),':')
       xlabel('Samples'), ylabel('a'), title('True (--), tracked RLS (-) and LS estimated (:) a parameter')
       subplot(212), plot(T,thm(:,2),'-',T,bm,'--',T,B(2)*ones(size(T)),':')
       xlabel('Samples'), ylabel('b'), title('True (--), tracked RLS (-) and LS estimated (:) b parameter')
```


Initial Conditions

- $\hat{\boldsymbol{\theta}}(0)$ is the initial parameter estimate.
- View $\boldsymbol{P}(0)$ as an estimate of the covariance matrix of the initial parameter estimate.
  - $\boldsymbol{P}(0)$ (and $\boldsymbol{P}(t)$) are covariance matrices, and must be symmetric and positive definite.
  - Choose $P(0) = \rho \boldsymbol{I}$.
  - ρ large $\Rightarrow$ large initial response. Good if initial estimate $\hat{\boldsymbol{\theta}}(0)$ is uncertain.

Forgetting Factor

Let $\lambda(t) = \lambda$. The forgetting factor λ will then determine the tracking capability.

- We must have $\lambda = 1$ to get convergence.
- λ small $\Rightarrow$ Old data is forgotten faster, hence better tracking.
- λ small $\Rightarrow$ the algorithm is more sensitive to noise (bad convergence).
- The memory constant is defined as $T_0 = \frac{1}{1-\lambda}$. If $\lambda = 0.95, T_0 = 20$

The choice of λ is consequently a trade-off between tracking capability and noise sensitivity. A typical choice is $\lambda \in (0.95, 0.99)$. It is common to let $\lambda(t)$ tend exponentially to 1, *e.g.*,

$$\lambda(t) = 1 - \lambda_0^t(1 - \lambda(0))$$

Simulation Example of RLS

In this example, data is generated from the model:

$$y(t) + ay(t-1) = bu(t-1) + e(t) + ce(t-1)$$

where $a = -0.8$, $b = 0.5$, $e(t)$ is zero mean white noise with standard deviation $\sigma = 0.5$.

The model to fit has the following structure:

$$y(t) + \hat{a}y(t-1) = \hat{b}u(t-1) + e(t)$$

The following parametrization is used

$$\hat{\theta} = \begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix}; \varphi(t) = \begin{bmatrix} -y(t-1) \\ u(t-1) \end{bmatrix}$$

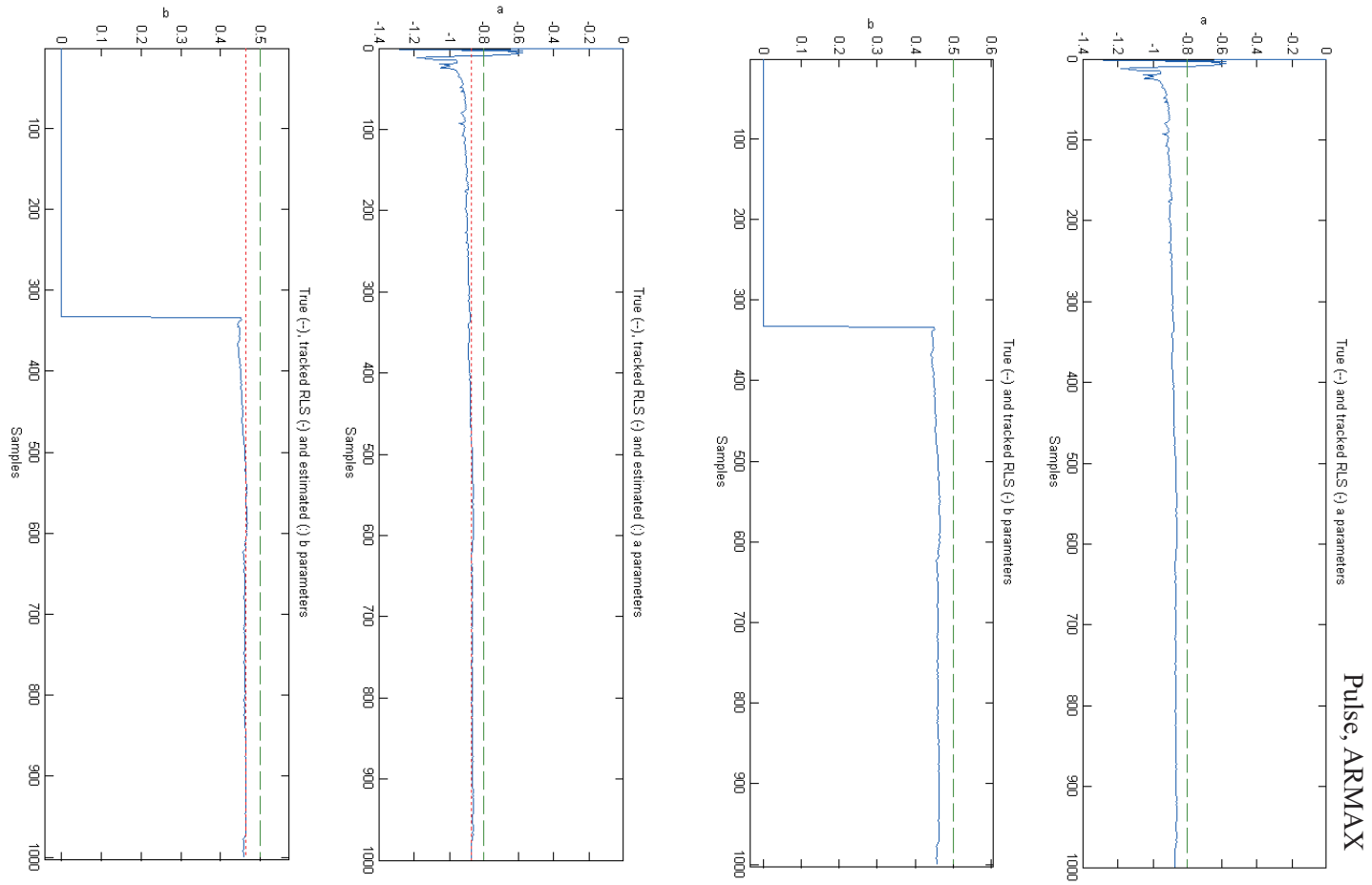
The recursive estimation was started from the initial conditions:

$$\hat{\boldsymbol{\theta}}(0) = \mathbf{0}; P(0) = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} = 100\boldsymbol{I}$$

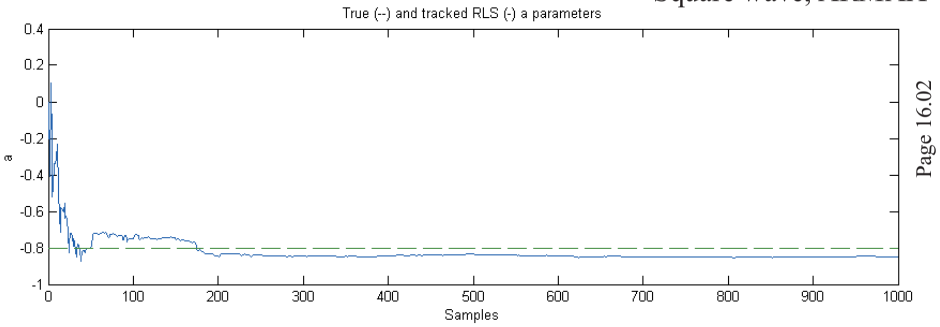
With this example, we want to illustrate:

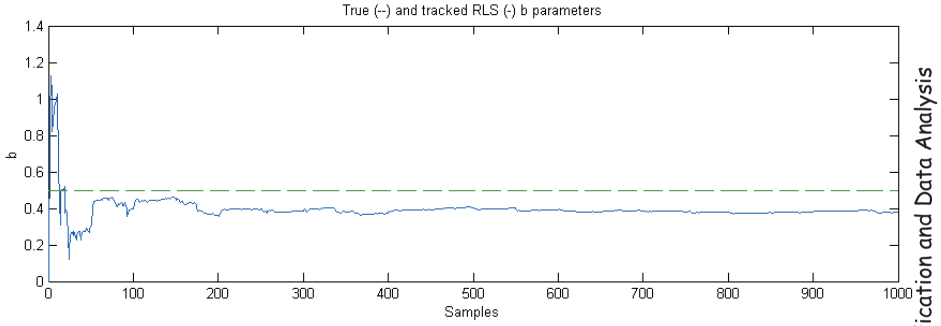
- Effects of the character of the input (two types of input signals are used: (1) a *unit pulse* at $t = 50$; (2) a *square wave* with values $\{0, 1\}$ with a period of 100 samples)
- Effects of the correctness of the model structure

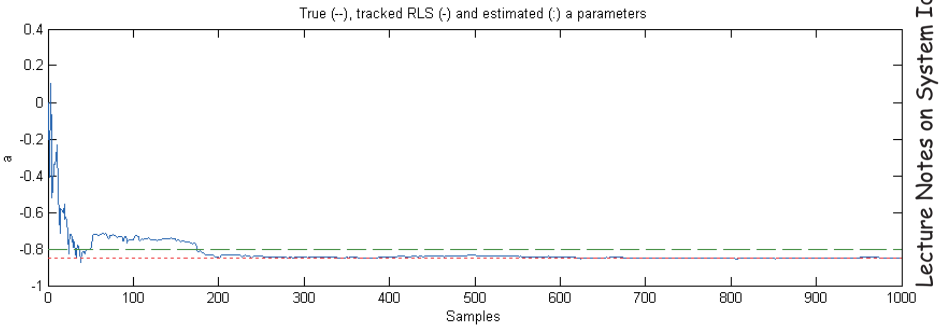
Please see simulation results from Lab. Experiences.

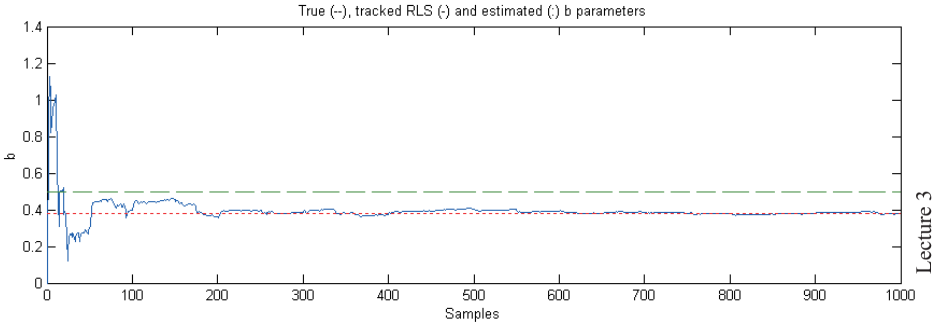


Square wave, ARMAX

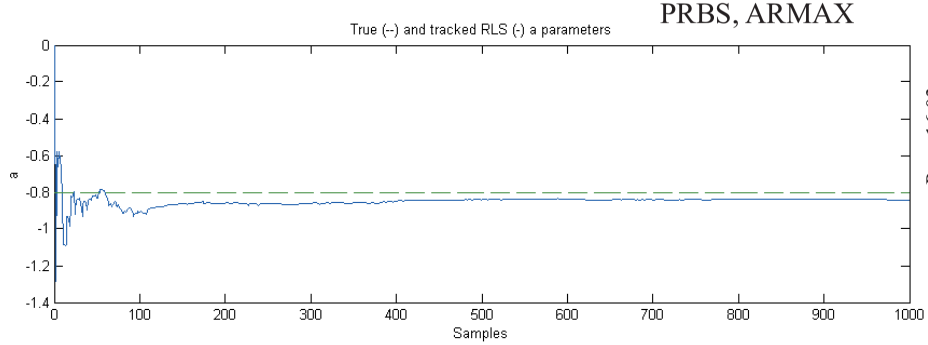


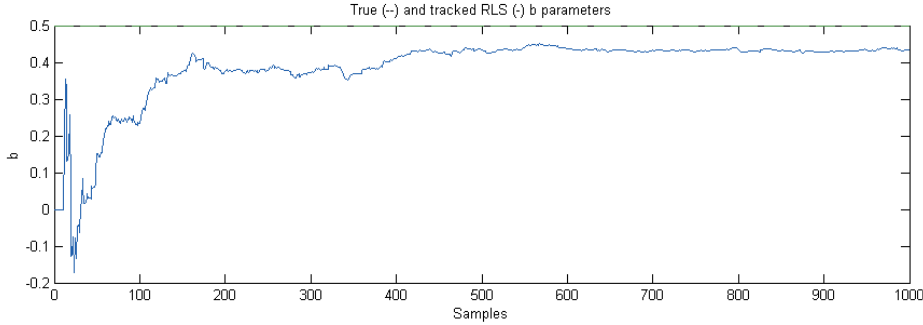


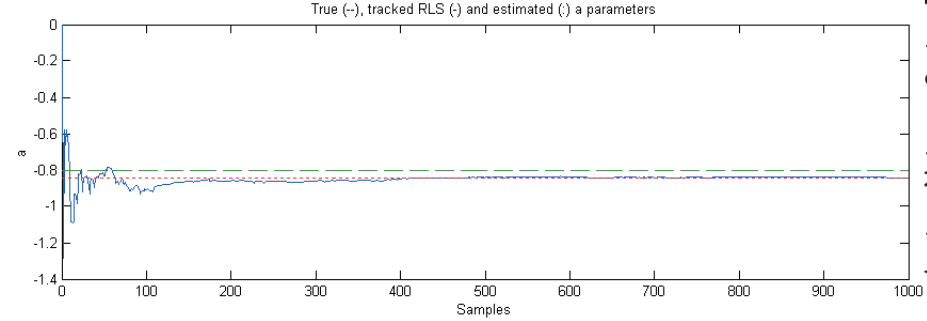


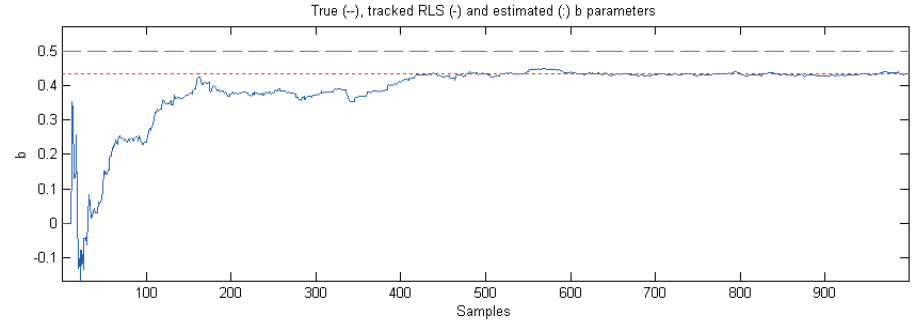


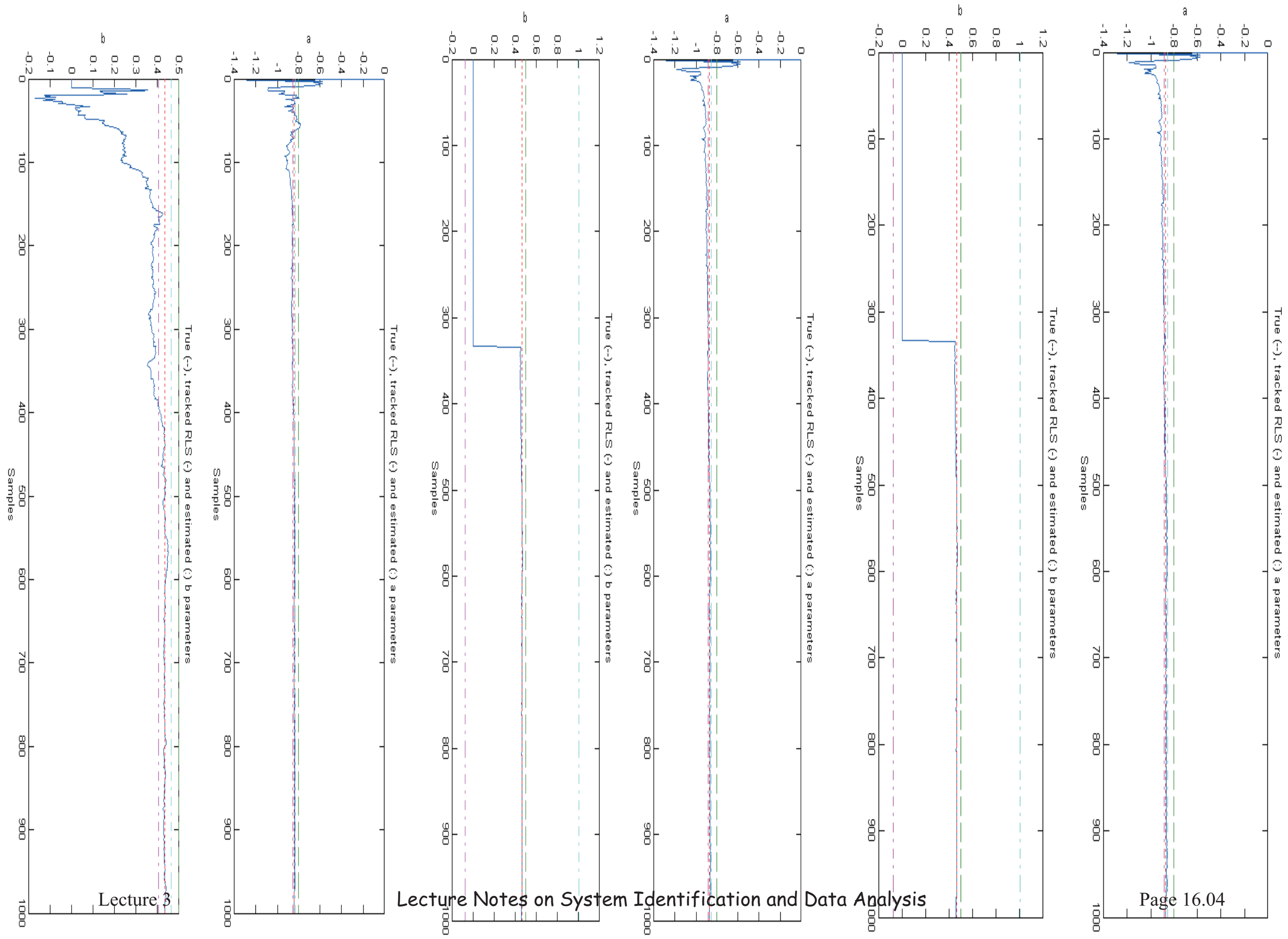
PRBS, ARMAX



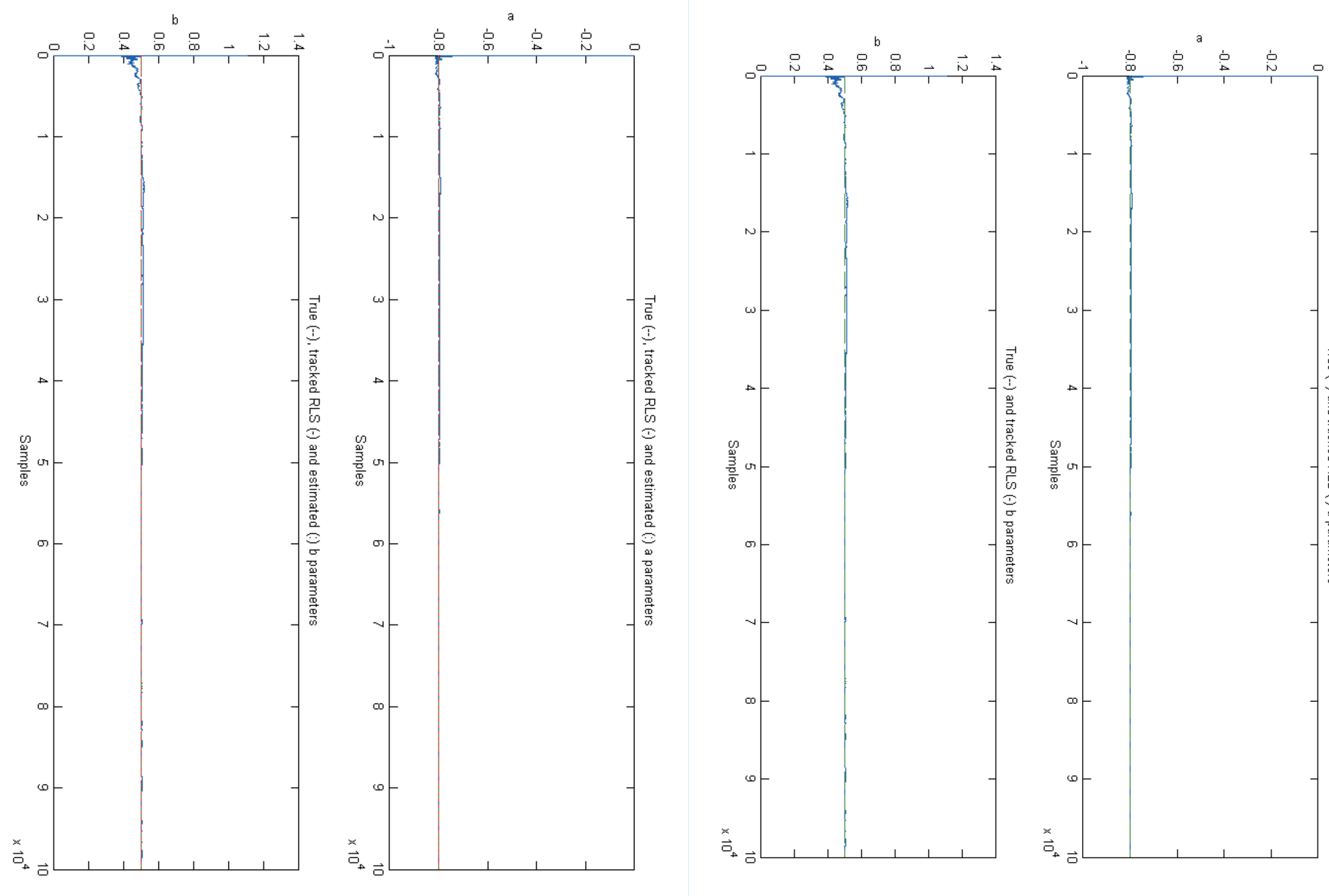


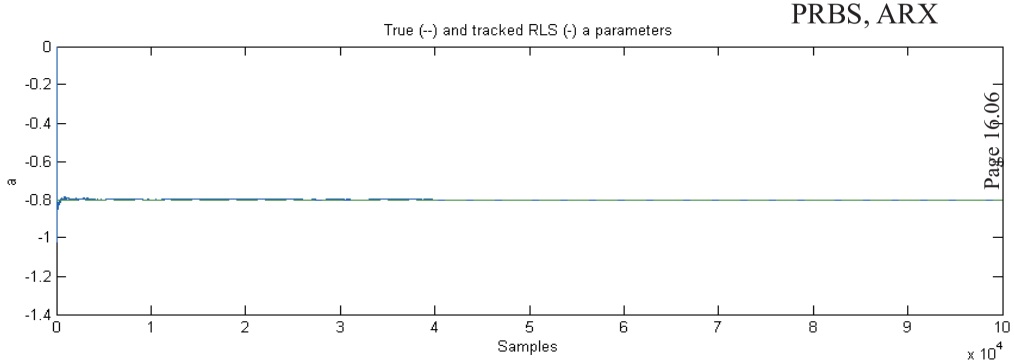


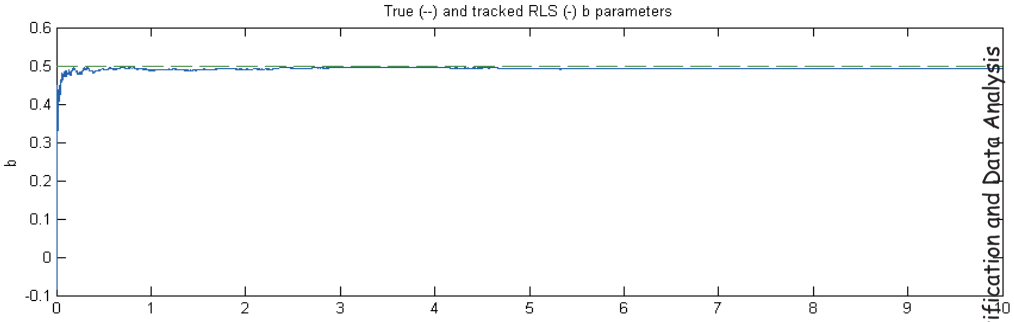


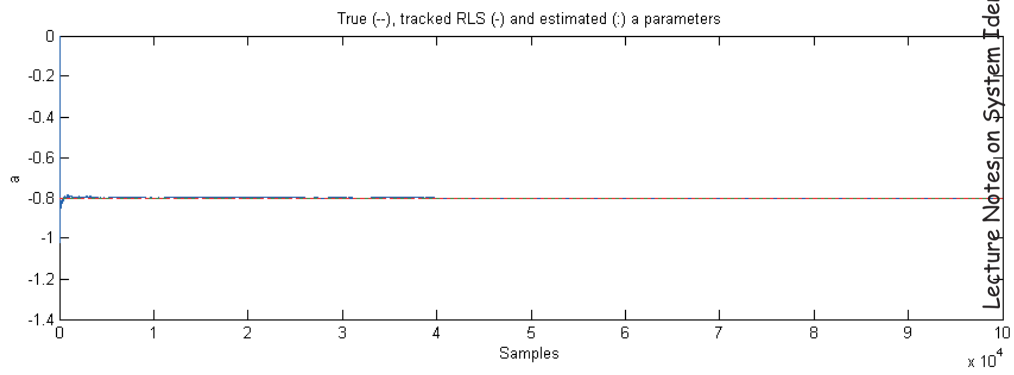


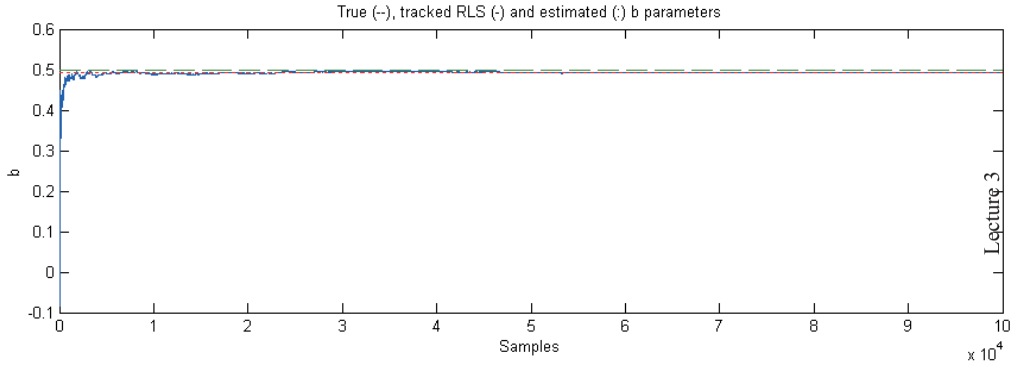
Square Wave, ARX












```
%%%
%%% Exercise Lecture 3.
%%%

clear all, close all

N = 1000;   % Sample number
Ts = 1;     % Sampling time
T  = [1:N]';

randn('state',1);
e = randn(N,1); % Equation error
sigma_e = 0.5;
e = sigma_e*(e - mean(e))/std(e);

pulse_signal = 0;
square_wave = 0;

if(pulse_signal),
   tp = round(N/3);
   u = zeros(N,1); u(tp) = 1;
elseif(square_wave),
   period = 100;
   u = square(2*pi*T/period);
else,
   u = idinput(N,'prbs',[0 1],[0 1]);
end;

a = -0.8;
b =  0.5;
c =  0.4;

th_star = [a b];
d = length(th_star);
armax_model = 1;
arx_model = 0;

if(armax_model),
   Gz = tf([0 b],[1 a],Ts); % Deterministic ARMAX transfer function
   Hz = tf([1 c],[1 a],Ts); % Stochastic ARMAX transfer function
elseif(arx_model),
   Gz = tf([0 b],[1 a],Ts); % Deterministic ARX transfer function
   Hz = tf([0 1],[1 a],Ts); % Stochastic ARX transfer function
end;

y = lsim([Gz Hz],[u e]); % ARX or ARMAX model simulation


P0 = 100*eye(d);
th0 = zeros(1,d);
lam = 1; % 0.98

thm   = rarx([y u],[1 1 1],'ff',lam,th0,P0);
m_arx = arx([y u],[1 1 1]); [A,B,dA,dB] = arxdata(m_arx);

figure, subplot(211), plot(T,thm(:,1),'-',T,th_star(1)*ones(size(T)),'--'), xlabel('Samples'), ylabel('a'),
      title('True (--) and tracked RLS (-) a parameters')
      subplot(212), plot(T,thm(:,2),'-',T,th_star(2)*ones(size(T)),'--'), xlabel('Samples'), ylabel('b')
      title('True (--) and tracked RLS (-) b parameters')


figure, subplot(211), plot(T,thm(:,1),'-',T,th_star(1)*ones(size(T)),'--',T,A(2)*ones(size(T)),':')
      xlabel('Samples'), ylabel('a'), title('True (--), tracked RLS (-) and estimated (:) a parameters')
      subplot(212), plot(T,thm(:,2),'-',T,th_star(2)*ones(size(T)),'--',T,B(2)*ones(size(T)),':')
      xlabel('Samples'), ylabel('b'), title('True (--), tracked RLS (-) and estimated (:) b parameters')


figure, subplot(211), plot(T,thm(:,1),'-',T,th_star(1)*ones(size(T)),'--',T,A(2)*ones(size(T)),':',...
                T,(A(2)+dA(2))*ones(size(T)),'-.',T,(A(2)-dA(2))*ones(size(T)),'-.')
      xlabel('Samples'), ylabel('a'), title('True (--), tracked RLS (-) and estimated (:) a parameters')
      subplot(212), plot(T,thm(:,2),'-',T,th_star(2)*ones(size(T)),'--',T,B(2)*ones(size(T)),':',...
                T,(B(2)+dB(2))*ones(size(T)),'-.',T,(B(2)-dB(2))*ones(size(T)),'-.')
      xlabel('Samples'), ylabel('b'), title('True (--), tracked RLS (-) and estimated (:) b parameters')
```

Common Problems for Recursive Identification

- Persistent excitation.
- Estimator windup.
- $\boldsymbol{P}(t)$ becomes indefinite.

Persistent Excitation

Just as for the off-line case, it is important that the input signal changes sufficiently in order to excite the system so that the experimental data contains enough information about the dynamics of the system. This leads to the concept of *persistent excitation* relating to the input signal. Such persistent excitation concept/condition applies during the whole identification period.

Most stable linear systems may be represented by the so called finite impulse response (FIR) model:

$$y(t) = b_1 u(t-1) + b_2 u(t-2) + ... + b_n u(t-n) = \varphi^T(t)\theta$$

where

$$\theta = [b_1 \ \ldots \ b_n]^T$$

$$\varphi = [u(t-1) \ \ldots \ u(t-n)]^T$$

The estimate is given by $\hat{\theta} = [\Phi^T\Phi]^{-1}\Phi^T Y$, where

$$\Phi^T\Phi = \begin{bmatrix} \Sigma_{k=n+1}^{t} u(k-1)^2 & \Sigma_{k=n+1}^{t} u(k-1)u(k-2) & \cdots & \Sigma_{k=n+1}^{t} u(k-1)u(k-n) \\ \Sigma_{k=n+1}^{t} u(k-2)u(k-1) & \Sigma_{k=n+1}^{t} u(k-2)^2 & \cdots & \Sigma_{k=n+1}^{t} u(k-2)u(k-n) \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{k=n+1}^{t} u(k-n)u(k-1) & \Sigma_{k=n+1}^{t} u(k-n)u(k-2) & \cdots & \Sigma_{k=n+1}^{t} u(k-n)^2 \end{bmatrix}$$

This matrix has to be non-singular for the estimate to be unique. This is called an excitation condition. For a long data set, $t \to \infty$, and all sums may be taken from 1 to t.

Define:

$$C_n = \lim_{t\to\infty} \frac{1}{t}\Phi^T\Phi = \begin{bmatrix} c(0) & c(1) & \cdots & c(n-1) \\ c(1) & c(0) & \cdots & c(n-2) \\ \vdots & \vdots & \ddots & \vdots \\ c(n-1) & c(n-2) & \cdots & c(0) \end{bmatrix}$$

where $c(k)$ are the empirical covariances of the input. That is:

$$c(k) = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} u(i)u(i-k)$$

Definition: A signal u is called *persistently exciting* (PE) of order n if the matrix C_n is positive definite.

Examples:

- **White noise**: $u(t)$ is white noise, with zero mean and variance σ^2. Then $c(n) = \sigma^2 \delta_n$, and $C_n = \sigma^2 I_n$, which is always positive definite. Thus C_n is nonsingular for all n, and white noise signal is PE of all orders.
- **Step signal**: $u(t)$ is a step of magnitude σ, then $c(k) = \sigma^2$, and C_n is nonsingular only if $n = 1$. Then a step is PE of order 1.
- **Impulse signal**: $u(t) = 1$ for $t = 0$, and 0 otherwise. This gives $c(n) = 0$ for all n and $C_n = 0$. Therefore, this signal is not PE of any order.

Important Note: It is necessary for consistent estimation of an n-th order system that the input signal be at least persistently exciting of order $2n$.

Estimator Windup

Often, some periods of an identification experiment exhibit poor excitation. This causes problems for the identification algorithms. Consider the extreme situation when $\boldsymbol{\varphi}(t) = 0$ in the RLS algorithm. Then

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1)$$

$$\boldsymbol{P}(t) = \frac{1}{\lambda}\boldsymbol{P}(t-1)$$

Notice:

- $\hat{\boldsymbol{\theta}}$ is constant as t increases.
- $\boldsymbol{P}$ increases exponentially with time for $\lambda < 1$.

When the system is excited again ($\boldsymbol{\varphi}(t) \neq 0$), then the estimator gain $\boldsymbol{K}(t) = \boldsymbol{P}(t)\boldsymbol{\varphi}(t)$ will be very large, and there will be an abrupt change in the estimate $\hat{\boldsymbol{\theta}}$, despite the fact that the system has not changed. This is referred to as *estimator windup*.

Solution:

- Do not update $\boldsymbol{P}(t)$ if we have poor excitation. There exist several algorithms for doing this automatically.

$\boldsymbol{P}(t)$ Indefinite

$\boldsymbol{P}(t)$ is a covariance matrix $\Rightarrow$ must be symmetric and positive definite.

Rounding errors may accumulate to make $\boldsymbol{P}(t)$ indefinite (which will make the estimate diverge). The solution is to note that every positive definite matrix can be written as

$$\boldsymbol{P}(t) = \boldsymbol{S}(t)\boldsymbol{S}^T(t)$$

One then rewrites the algorithm to recursively update $\boldsymbol{S}(t)$ instead (*Potters Square Root Algorithm*).

We can also use the UD factorization and SVD (singular value decomposition) to solve the problem, i.e.

$$\boldsymbol{P}(t) = UDU^T \text{ or } \boldsymbol{P}(t) = U\sum V^T$$

Conclusions

- In practical scenarios, one often need to use recursive identification (time-varying systems, online identification, fault diagnosis).
- Both the LS and the IVM can easily be recast in recursive forms. The PEM can only be approximated.
- The properties of the on-line methods are comparable with the off-line case.
- Tracking capability can be incorporated by using forgetting factor techniques, or by model the parameter variations.
- There is always a tradeoff between convergence speed and tracking properties, as well as computational complexity and accuracy.
- In practice, one can make simplifications and modifications to make the recursion cheaper and more numerically robust.

Reading and Exercises

- Reading: Chapter 11
- Exercises: *See Lab. Experiences...*