Recursive least squares with forgetting.

Recursive least squares with forgetting.

We would like to modify the recursive least squares algorithm so that older data has less effect on the coefficient estimation. We can assign a weighting function to *bias* the error estimate as follows.

$$V_n(\theta) = \sum_{s=1}^{n} \lambda^{n-s} \epsilon^2(s)$$

where λ is chosen to be between 0 and 1. This is comparable with lecture 5 where the *unbiased* error was defined in matrix notation as

$$V_n(\theta) = E^T E$$

Thus if we make λ less than 1 the older values of error will have less effect on the error function and older data will therefore have less effect on the model estimate.

The recursive least squares estimate with forgetting then becomes

$$\begin{aligned}
\hat{\theta}(n) &= \hat{\theta}(n-1) + K(n)\epsilon(n) \\
\epsilon(n) &= y(n) - \varphi^T(n)\hat{\theta}(n-1) \\
K(n) &= P(n)\varphi(n) = \frac{P(n-1)\varphi(n)}{\lambda + \varphi^T(n)P(n-1)\varphi(n)} \\
P(n) &= \frac{1}{\lambda}\left(P(n-1) - \frac{P(n-1)\varphi(n)\varphi^T(n)P(n-1)}{\lambda + \varphi^T(n)P(n-1)\varphi(n)}\right)
\end{aligned}$$

Phi	Φ
epsilon	ϵ
lambda	λ
theta	θ
varphi	φ

By setting $\lambda = 1$ we return to the conventional least squares.

Recall that $y(n)$ is the current output, $\varphi(n)$ contains all the past values of input and output.

$$\varphi^T(n) = \begin{pmatrix} -y(n-1) & -y(n-2) & \dots & -y(n-Na) & u(n-1) & u(n-2) & \dots & u(n-Nb) \end{pmatrix}$$

It is convenient to set the initial value of `theta` to zeros and the initial value of `P` to `LN * I`, where `I` is the identity matrix and `LN` is a large number.

Skeleton code for recursive least squares estimate (see crib.m)

```
% A skeleton for a recursive identification algorithm. Since the assignment
% assumes an ARX model this code does likewise.
% You need to set Na, Nb (number of a and b coefficients) and LN (large
% number).  Nc is not needed if this is an ARX model. You also need to supply
% the algorithm!
% The code assumes that the data is in a vector y and the input is in a vector u

theta_nminus1=zeros(Na+Nb,1); % Initialise the estimate of theta to zero
P_nminus1=LN.*eye(Na+Nb);  % Initialise P where LN is a large number
% Theta=[]; % history of theta starts here
% Step through data and for each new point generate a new estimate
for n=1:10 % Change 10 to length(y) once you have the code working
% set py to the previous Na y values
  py=zeros(1,Na);
  for i=n-1:-1:n-Na
    if i>0 py(n-i)=y(i);   end
  end
% set pu to the previous Nb u values
  pu=zeros(1,Nb);
  for i=n-1:-1:n-Nb
    if i>0 pu(n-i)=u(i);   end
  end
% Construct varphi from py' and pu'
% varphi= ...
% Use varphi(n), y(n) theta(n-1) and P(n-1) to iterate the next estimate
% epsilon= ...
% P= P_nminus1 - ...
% K= ...
% theta=theta_nminus1 + ...
```

```
% get ready for the new iteration
  theta_nminus1=theta;
  P_nminus1=P;
% To get a history of theta set Theta=[Theta; theta']
end % and so it ends
% If you have recorded parameter evolution you can plot with
% plot(Theta)
```

WSH, May 1996