

System Identification

Control Engineering EN, 3rd year B.Sc.
Technical University of Cluj-Napoca
Romania

Lecturer: Lucian Buşoniu



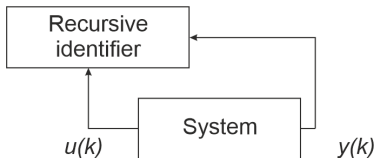
Part IX

Recursive identification methods

Table of contents

- 1 Introduction and motivation
- 2 Recursive least-squares and ARX
- 3 Recursive instrumental variables

Recursive identification: Idea



Recursive methods can work *online*, while the system is running.

At each step k , compute a *new parameter estimate* $\hat{\theta}(k)$ (= a new model), based on the previous estimate $\hat{\theta}(k-1)$ and newly available data $u(k)$, $y(k)$.

Remark: To contrast them with recursive identification, the previous methods, which used the whole data set at once, will be called *batch* identification.

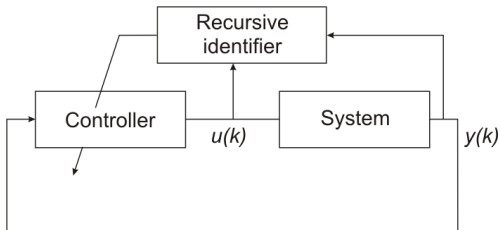
Motivation

Recursive methods:

- Require less memory, and less computation *for each update*, than the whole batch algorithm.
- ⇒ Easier to apply in real-time.
(total computation, after the entire dataset, may be larger)
- When properly modified, can deal with systems that change over time (*time-varying*).

Motivation (continued)

- If model is used to tune a controller, we obtain *adaptive control*:



Sketch of adaptive control based on recursive identification

initialize model and controller

loop at every step $k = 1, 2, \dots$

 apply $u(k)$ with current controller, measure $y(k)$

 update model using latest sample $u(k), y(k)$

 adapt/redesign controller using new model

end loop

Disadvantage

Recursive methods are usually approximations of batch techniques
⇒ guarantees more difficult to obtain.

Motivating example: Estimating a scalar

Recall scalar estimation. Model:

$$y(k) = b + e(k) = 1 \cdot b + e(k) = \varphi(k)\theta + e(k)$$

where $\varphi(k) = 1 \forall k$, $\theta = b$.

For the data points up to and including k :

$$y(1) = \varphi(1)\theta = 1 \cdot b$$

...

$$y(k) = \varphi(k)\theta = 1 \cdot b$$

After calculation, the solution of this system is:

$$\hat{\theta}(k) = \frac{1}{k} [y(1) + \dots + y(k)]$$

(estimate is the average of all measurements, filtering out the noise).

Estimating a scalar: Recursive formulation

Rewrite the formula:

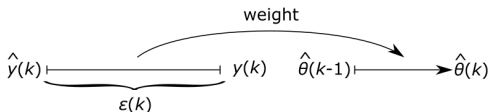
$$\begin{aligned}\hat{\theta}(k) &= \frac{1}{k}[y(1) + \dots + y(k)] \\ &= \frac{1}{k}[(k-1)\frac{1}{k-1}(y(1) + \dots + y(k-1)) + y(k)] \\ &= \frac{1}{k}[(k-1)\hat{\theta}(k-1) + y(k)]\end{aligned}$$

(already a recursive formula, but we go on to gain more intuition)

$$\begin{aligned}&= \frac{1}{k}[k\hat{\theta}(k-1) + y(k) - \hat{\theta}(k-1)] \\ &= \hat{\theta}(k-1) + \frac{1}{k}[y(k) - \hat{\theta}(k-1)]\end{aligned}$$

Estimating a scalar: Insight

$$\hat{\theta}(k) = \hat{\theta}(k-1) + \frac{1}{k} [y(k) - \hat{\theta}(k-1)]$$



Method has many of the features of general recursive techniques:

- Recursive formula: new estimate $\hat{\theta}(k)$ computed based on previous estimate $\hat{\theta}(k-1)$ and new data $y(k)$.
- $[y(k) - \hat{\theta}(k-1)]$ is a **prediction error** $\epsilon(k)$, since $\hat{\theta}(k-1) = \hat{b} = \hat{y}(k)$, a one-step-ahead prediction of the output.
- Update applies a correction proportional to $\epsilon(k)$, weighted by $\frac{1}{k}$:
 - When the error $\epsilon(k)$ is large (or small), a large (or small) adjustment is made.
 - The weight $\frac{1}{k}$ decreases with time k , so adjustments get smaller as $\hat{\theta}$ becomes better.

Table of contents

- 1 Introduction and motivation
- 2 Recursive least-squares and ARX
 - General recursive least-squares
 - Recursive ARX
 - Matlab example
- 3 Recursive instrumental variables

Recall: Least-squares regression

Model:

$$y(k) = \varphi^\top(k)\theta + e(k)$$

Dataset up to k gives a linear system of equations:

$$y(1) = \varphi^\top(1)\theta$$

$$y(2) = \varphi^\top(2)\theta$$

...

$$y(k) = \varphi^\top(k)\theta$$

After some linear algebra and calculus, the least-squares solution can be written:

$$\hat{\theta}(k) = \left[\sum_{j=1}^k \varphi(j)\varphi^\top(j) \right]^{-1} \left[\sum_{j=1}^k \varphi(j)y(j) \right]$$

Least-squares: Recursive formula

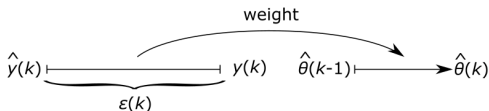
With notation $P(k) = \sum_{j=1}^k \varphi(j)\varphi^T(j)$:

$$\begin{aligned}
 \hat{\theta}(k) &= P^{-1}(k) \left[\sum_{j=1}^k \varphi(j)y(j) \right] \\
 &= P^{-1}(k) \left[\sum_{j=1}^{k-1} \varphi(j)y(j) + \varphi(k)y(k) \right] \\
 &= P^{-1}(k) \left[P(k-1)\hat{\theta}(k-1) + \varphi(k)y(k) \right] \\
 &= P^{-1}(k) \left[[P(k) - \varphi(k)\varphi^T(k)]\hat{\theta}(k-1) + \varphi(k)y(k) \right] \\
 &= \hat{\theta}(k-1) + P^{-1}(k) \left[-\varphi(k)\varphi^T(k)\hat{\theta}(k-1) + \varphi(k)y(k) \right] \\
 &= \hat{\theta}(k-1) + P^{-1}(k)\varphi(k) \left[y(k) - \varphi^T(k)\hat{\theta}(k-1) \right]
 \end{aligned}$$

Least-squares: Insight

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P^{-1}(k)\varphi(k) \left[y(k) - \varphi^T(k)\hat{\theta}(k-1) \right]$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + W(k)\varepsilon(k)$$



- Recursive formula: new estimate $\hat{\theta}(k)$ computed based on previous estimate $\hat{\theta}(k-1)$ and new data $y(k)$.
- $\left[y(k) - \varphi^T(k)\hat{\theta}(k-1) \right]$ is a prediction error $\varepsilon(k)$, since $\varphi^T(k)\hat{\theta}(k-1) = \hat{y}(k)$ is the one-step-ahead prediction using the previous parameter vector.
- $W(k) = P^{-1}(k)\varphi(k)$ is a weighting vector: elements of P grow large for large k , therefore $W(k)$ decreases.

Recursive matrix inversion

The previous formula requires matrix inverse $P^{-1}(k)$, which is computationally costly.

For P , an easy recursion exists: $P(k) = P(k-1) + \varphi(k)\varphi^T(k)$, but this does not help; the matrix must still be inverted.

The **Sherman-Morrison** formula gives a recursion for the inverse P^{-1} :

$$P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)\varphi(k)\varphi^T(k)P^{-1}(k-1)}{1 + \varphi^T(k)P^{-1}(k-1)\varphi(k)}$$

Exercise: Prove the Sherman-Morrison formula! (linear algebra)

Overall algorithm

Recursive least-squares (RLS)

initialize $\hat{\theta}(0)$, $P^{-1}(0)$

loop at every step $k = 1, 2, \dots$

measure $y(k)$, form regressor vector $\varphi(k)$

find prediction error $\varepsilon(k) = y(k) - \varphi^\top(k)\hat{\theta}(k-1)$

update inverse $P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)\varphi(k)\varphi^\top(k)P^{-1}(k-1)}{1 + \varphi^\top(k)P^{-1}(k-1)\varphi(k)}$

compute weights $W(k) = P^{-1}(k)\varphi(k)$

update parameters $\hat{\theta}(k) = \hat{\theta}(k-1) + W(k)\varepsilon(k)$

end loop

Each new model can then be used for some purpose, e.g. function approximation or adaptive control

Initialization

The RLS algorithm requires initial parameter $\hat{\theta}(0)$, and initial inverse $P^{-1}(0)$.

Typical choices without prior knowledge:

- $\hat{\theta}(0) = [0, \dots, 0]^T$ - a vector of n zeros.
- $P^{-1}(0) = \delta I$, with δ a large number such as 10^3 .
(An equivalent initial value of P would be $P(0) = \frac{1}{\delta} I$.)

Intuition: $P^{-1}(0)$ large means that initially the weights $W(k)$ are large and large updates are applied to $\hat{\theta}$, to learn fast from the samples.

Conversely, if good prior parameters $\hat{\theta}$ are available, $\hat{\theta}(0)$ can be initialized to them, and δ correspondingly decreased so that the updates start smaller.

Sanity check: Recovering the scalar case

$$y(k) = b + e(k) = \varphi(k)\theta + e(k), \text{ where } \varphi(k) = 1, \theta = b$$

Take $\hat{\theta}(0) = 0, P^{-1}(0) \rightarrow \infty$. We have with Sherman-Morrison:

$$P^{-1}(k) = P^{-1}(k-1) - \frac{(P^{-1}(k-1))^2}{1 + P^{-1}(k-1)} = \frac{P^{-1}(k-1)}{1 + P^{-1}(k-1)}$$

$$P^{-1}(1) = 1$$

$$P^{-1}(2) = \frac{1}{2}$$

...

$$P^{-1}(k) = \frac{1}{k}$$

Also, $\varepsilon(k) = y(k) - \hat{\theta}(k-1)$ and $W(k) = P^{-1}(k) = \frac{1}{k}$, leading to:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + W(k)\varepsilon(k) = \hat{\theta}(k-1) + \frac{1}{k}[y(k) - \hat{\theta}(k-1)]$$

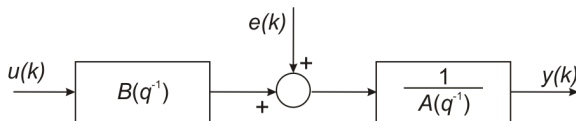
⇒ formula for the scalar case was recovered.

Table of contents

- 1 Introduction and motivation
- 2 Recursive least-squares and ARX
 - General recursive least-squares
 - Recursive ARX
 - Matlab example
- 3 Recursive instrumental variables

Recall: ARX model

$$A(q^{-1})y(k) = B(q^{-1})u(k) + e(k)$$



Linear regression representation:

$$\begin{aligned} y(k) &= -a_1y(k-1) - a_2y(k-2) - \dots - a_{na}y(k-na) \\ &\quad b_1u(k-1) + b_2u(k-2) + \dots + b_{nb}u(k-nb) + e(k) \\ &= [-y(k-1) \quad \dots \quad -y(k-na) \quad u(k-1) \quad \dots \quad u(k-nb)] \\ &\quad \cdot [a_1 \quad \dots \quad a_{na} \quad b_1 \quad \dots \quad b_{nb}]^T + e(k) \\ &=: \varphi^T(k)\theta + e(k) \end{aligned}$$

Regressor vector: $\varphi \in \mathbb{R}^{na+nb}$, previous output and input values.

Parameter vector: $\theta \in \mathbb{R}^{na+nb}$, polynomial coefficients a_1, a_2, \dots, a_{na}
and b_1, b_2, \dots, b_{nb} .

Recursive ARX

With the linear regression representation, recursive ARX is just an instantiation of RLS template.

Recursive ARX

initialize $\hat{\theta}(0)$, $P^{-1}(0)$

loop at every step $k = 1, 2, \dots$

measure $u(k)$, $y(k)$

form regressor vector

$$\varphi(k) = [-y(k-1), \dots, -y(k-na), u(k-1), \dots, u(k-nb)]^T$$

find prediction error $\varepsilon(k) = y(k) - \varphi^T(k)\hat{\theta}(k-1)$

$$\text{update inverse } P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)\varphi(k)\varphi^T(k)P^{-1}(k-1)}{1 + \varphi^T(k)P^{-1}(k-1)\varphi(k)}$$

compute weights $W(k) = P^{-1}(k)\varphi(k)$

update parameters $\hat{\theta}(k) = \hat{\theta}(k-1) + W(k)\varepsilon(k)$

end loop

Remark: Outputs more than na steps ago, and inputs more than nb steps ago, are not used so they can be forgotten, reducing memory usage.

Guarantees idea

With $\hat{\theta}(0) = 0$, $P^{-1}(0) = \delta I$ and $\delta \rightarrow \infty$, recursive ARX at step k is equivalent to running batch ARX on the dataset $u(1), y(1), \dots, u(k), y(k)$.

⇒ the same guarantees as batch ARX:

Theorem

Under appropriate assumptions (including the existence of true parameters θ_0), ARX identification is **consistent**: the estimated parameters $\hat{\theta}$ tend to the true parameters θ_0 as $k \rightarrow \infty$.

The condition on P is to ensure the inverses are the same as in the offline case.

Table of contents

- 1 Introduction and motivation
- 2 Recursive least-squares and ARX
 - General recursive least-squares
 - Recursive ARX
 - Matlab example
- 3 Recursive instrumental variables

System

To illustrate recursive ARX, we take a known system:

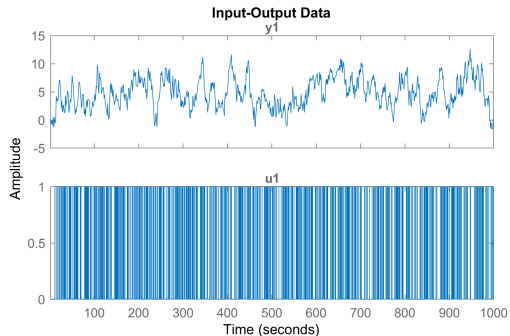
$$y(k) + ay(k - 1) = bu(k - 1) + e(k), \quad a = -0.9, b = 1$$

(Söderström & Stoica)

```
system = idpoly([1 -0.9], [0 1]);
```

Identification data obtained:

```
sim(system, u, 'noise');
```



Recursive ARX

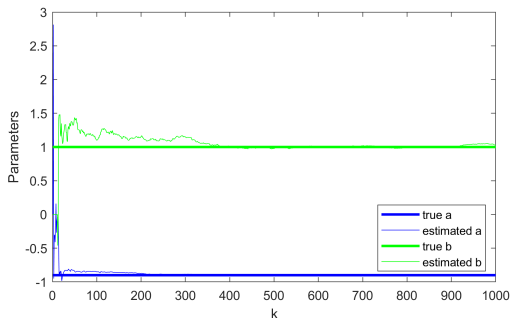
```
model = rarx(id, [na, nb, nk], 'ff', 1, th0, Pinv0);
```

Arguments:

- 1 Identification data.
- 2 Array containing the orders of A and B and the delay nk .
- 3 'ff', 1 selects the algorithm variant presented in this lecture.
- 4 $th0$ is the initial parameter value.
- 5 $Pinv0$ is the initial inverse $P^{-1}(0)$.

Results

- $na = nb = nk = 1$
- For all the experiments: $P^{-1}(0) = \delta I$ with $\delta = 1000$, and $\theta_0 = [0, 0]^T$



Conclusion: Algorithm converges to true parameter values

System outside model class

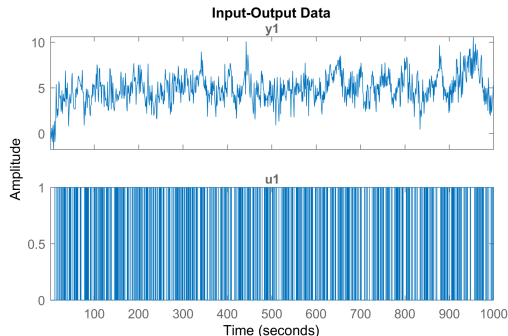
Take now an *output-error* system (so it is not ARX with white noise):

$$y(k) = \frac{bq^{-1}}{1 + fq^{-1}}u(k) + e(k), \quad f = -0.9, b = 1$$

(Söderström & Stoica)

```
system = idpoly([], [0 1], [], [], [1 -0.9]);
```

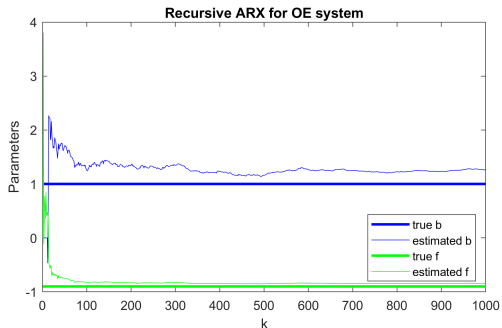
Identification data:



Results

$nf = nb = nk = 1$, so we attempt the ARX model:

$$y(k) + fy(k-1) = bu(k-1) + e(k)$$



Conclusion: Convergence to the true values is not attained! – due to colored noise (system not in model class). Just as in the IV lecture, the algorithm is inconsistent.

Other recursive PEM algorithms

Other recursive PEM methods available in Matlab, e.g.:

- ARMAX, `rarmax`
- output-error, `roe`
- generic prediction error method, `rpem`

Table of contents

- 1 Introduction and motivation
- 2 Recursive least-squares and ARX
- 3 Recursive instrumental variables**
 - Recursive IV method
 - Matlab example

Recall: Instrumental variable method

We'll start from the following formula to compute the IV-method parameter vector:

$$\hat{\theta} = \left[\frac{1}{N} \sum_{k=1}^N Z(k) \varphi^T(k) \right]^{-1} \left[\frac{1}{N} \sum_{k=1}^N Z(k) y(k) \right]$$

Regressor vector:

$$\varphi(k) = [-y(k-1), \dots, -y(k-na), u(k-1), \dots, u(k-nb)]^T,$$

instrument vector usually:

$$Z(k) = [-x(k-1), \dots, -x(k-na), u(k-1), \dots, u(k-nb)]^T.$$

Since instrument vector $Z(k)$ uncorrelated with the noise, IV methods can handle colored noise.

Recursive formulation (1)

Averaging by the number of data points can be removed, leading to:

$$\hat{\theta} = \left[\sum_{k=1}^N Z(k)\varphi^T(k) \right]^{-1} \left[\sum_{k=1}^N Z(k)y(k) \right]$$

Averaging was needed in batch IV since summations over many data points could grow very large, leading to numerical problems.

Recursive IV will, in the end, add terms one by one, and will therefore be numerically more stable.

Rewrite the equation for the recursive case:

$$\hat{\theta}(k) = P^{-1}(k) \left[\sum_{j=1}^k Z(j)y(j) \right]$$

where $P(k) = \sum_{j=1}^k Z(j)\varphi^T(j)$.

Recursive formulation (2)

With this definition of P , the recursive update is found similarly to RLS:

$$\begin{aligned}
 \hat{\theta}(k) &= P^{-1}(k) \left[\sum_{j=1}^k Z(j)y(j) \right] \\
 &= P^{-1}(k) \left[\sum_{j=1}^{k-1} Z(j)y(j) + Z(k)y(k) \right] \\
 &= P^{-1}(k) \left[P(k-1)\hat{\theta}(k-1) + Z(k)y(k) \right] \\
 &= P^{-1}(k) \left[[P(k) - Z(k)\varphi^T(k)]\hat{\theta}(k-1) + Z(k)y(k) \right] \\
 &= \hat{\theta}(k-1) + P^{-1}(k) \left[-Z(k)\varphi^T(k)\hat{\theta}(k-1) + Z(k)y(k) \right] \\
 &= \hat{\theta}(k-1) + P^{-1}(k)Z(k) \left[y(k) - \varphi^T(k)\hat{\theta}(k-1) \right]
 \end{aligned}$$

Recursive formulation (3)

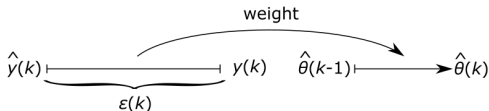
Final formula:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P^{-1}(k)Z(k) \left[y(k) - \varphi^T(k)\hat{\theta}(k-1) \right]$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + W(k)\varepsilon(k)$$

with the Sherman-Morrison update of the inverse:

$$P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)Z(k)\varphi^T(k)P^{-1}(k-1)}{1 + \varphi^T(k)P^{-1}(k-1)Z(k)}$$



Recursive IV: Overall algorithm

Recursive IV

initialize $\hat{\theta}(0)$, $P^{-1}(0)$

loop at every step $k = 1, 2, \dots$

measure $y(k)$, form regressor vector $\varphi(k)$ and instrument vector

$$Z(k) = [-x(k-1), \dots, -x(k-na), u(k-1), \dots, u(k-nb)]^T$$

find prediction error $\varepsilon(k) = y(k) - \varphi^T(k)\hat{\theta}(k-1)$

$$\text{update inverse } P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)Z(k)\varphi^T(k)P^{-1}(k-1)}{1 + \varphi^T(k)P^{-1}(k-1)Z(k)}$$

compute weights $W(k) = P^{-1}(k)Z(k)$

update parameters $\hat{\theta}(k) = \hat{\theta}(k-1) + W(k)\varepsilon(k)$

end loop

Table of contents

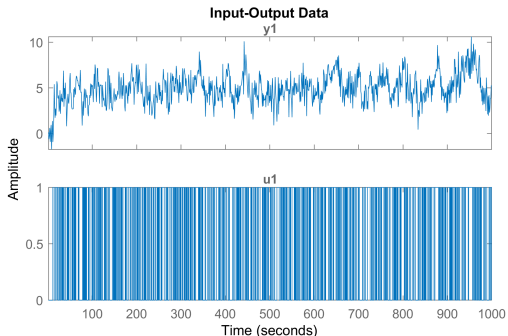
- 1 Introduction and motivation
- 2 Recursive least-squares and ARX
- 3 Recursive instrumental variables**
 - Recursive IV method
 - **Matlab example**

OE system

Take the same OE system we used in the ARX example:

$$y(k) = \frac{bq^{-1}}{1 + fq^{-1}}u(k) + e(k), \quad f = -0.9, b = 1$$

with the same dataset:

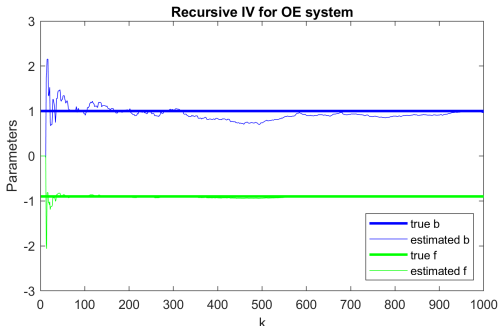


Results with recursive IV

We use the IV model with colored noise $v(k)$:

$$y(k) + fy(k - 1) = bu(k - 1) + v(k)$$

Instrument vector: $Z(k) = [u(k - 2), u(k - 1)]$.



Conclusion: Better than recursive ARX, parameters converge to their true values.

Summary

- Learn parameters incrementally, online: $\hat{\theta}(k)$ from $\hat{\theta}(k-1)$ and new data $u(k), y(k)$
- Produce a model early, easier to apply in real-time
- Motivating example: estimating a scalar from noisy measurements
- General linear regression, recursive variant:

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P^{-1}(k)\varphi(k) \left[y(k) - \varphi^T(k)\hat{\theta}(k-1) \right]$$

- Tuning knob: initial size δ of the inverse
- Application to ARX is immediate, leading to recursive ARX
- For IV some equations need to be revisited but the line is the same
- Matlab examples for both recursive ARX and recursive IV