System Identification

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

Lecturer: Lucian Buşoniu

Part VII

Prediction error methods

Table of contents

1. Model structures

2. General prediction error methods

3. Solving the optimization problem

Classification

Recall **Types of models** from Part I:

1. Mental or verbal models
2. Graphs and tables (nonparametric)
3. Mathematical models, with two subtypes:
   - First-principles, analytical models
   - Models from system identification

Like ARX, general prediction error methods (PEM) produce *parametric*, polynomial models.

Table of contents

1. Model structures
2. General prediction error methods
3. Solving the optimization problem

Motivation

General PEM can be seen as an extension of ARX to significantly more general model structures, and are therefore able to identify more classes of systems.

To clarify, we first introduce the general class of models to which PEM can be applied.

General model structure

$$y(k) = G(q^{-1})u(k) + H(q^{-1})e(k)$$

where G and H are discrete-time *transfer functions* – ratios of polynomials. Signal $e(t)$ is zero-mean white noise.

By placing the common factors of the denominators of G and H in $A(q^{-1})$, we get the more detailed form:

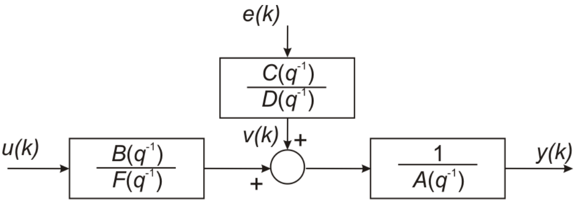
$$y(k) = \frac{B(q^{-1})}{A(q^{-1})F(q^{-1})}u(k) + \frac{C(q^{-1})}{A(q^{-1})D(q^{-1})}e(k)$$
$$A(q^{-1})y(k) = \frac{B(q^{-1})}{F(q^{-1})}u(k) + \frac{C(q^{-1})}{D(q^{-1})}e(k)$$

where A, B, C, D, F are all polynomials, of orders na, nb, nc, nd, nf:

$$A = 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na}$$
$$B(q^{-1}) = b_1 q^{-1} + \ldots + b_{nb} q^{-nb} \qquad F(q^{-1}) = 1 + f_1 q^{-1} + \ldots + f_{nf} q^{-nf}$$
$$C(q^{-1}) = 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc} \qquad D(q^{-1}) = 1 + d_1 q^{-1} + \ldots + d_{nd} q^{-nd}$$

General model structure (continued)

$$\boxed{A(q^{-1})y(k) = \frac{B(q^{-1})}{F(q^{-1})}u(k) + \frac{C(q^{-1})}{D(q^{-1})}e(k)}$$

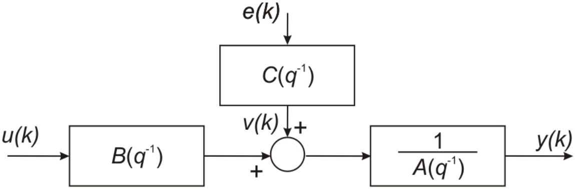


Very general form, all other linear forms are special cases of this. Not for practical use, but to describe algorithms in a generic way. In practice, we use one of the special cases, as exemplified next.

ARMAX model structure

Setting $F = D = 1$ (i.e. orders $nf = nd = 0$), we get:

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



Name: AutoRegressive, Moving Average (referring to noise model) with eXogenous input (dependence on *u*)

ARMAX: explicit form

$$\begin{aligned} A(q^{-1})y(k) &= B(q^{-1})u(k) + C(q^{-1})e(k) \\ A(q^{-1}) &= 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na} \\ B(q^{-1}) &= b_1 q^{-1} + \ldots + b_{nb} q^{-nb} \\ C(q^{-1}) &= 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc} \end{aligned}$$

$$\begin{aligned} y(k) &+ a_1 y(k-1) + \ldots + a_{na} y(k-na) \\ &= b_1 u(k-1) + \ldots + b_{nb} u(k-nb) \\ &+ e(k) + c_1 e(k-1) + \ldots + c_{nc} e(k-nc) \end{aligned}$$

with parameter vector:

$$\theta = [a_1, \ldots, a_{na}, b_1, \ldots, b_{nb}, c_1, \ldots, c_{nc}]^\top \in \mathbb{R}^{na+nb+nc}$$

Special case of ARMAX: ARX

Setting $C = 1$ in ARMAX ($nc = 0$), we get:

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

precisely the ARX model we worked with before.

Compared to ARX, ARMAX can model more intricate disturbances ($C(q^{-1})e(k)$ instead of $e(k)$, which is often assumed to be zero-mean white-noise).

Recall: FIR special case of ARX

Further setting $A = 1$ ($na = 0$) in ARX, we get:

$$
\begin{aligned}
y(k) = B(q^{-1})u(k) + e(k) &= \sum_{j=1}^{nb} b_j u(k-j) + e(k) \\
&= \sum_{j=0}^{M-1} h(j)u(k-j) + e(k)
\end{aligned}
$$

the FIR model.

Overall relationship

$$\text{General Form} \supset \text{ARMAX} \supset \text{ARX} \supset \text{FIR}$$

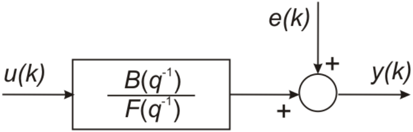
- ARMAX to ARX: Less freedom in modeling disturbance.
- ARX to FIR: More parameters required.

Output error

Other model forms are possible that are not special cases of ARMAX, e.g. Output Error, OE:

$$\boxed{y(k) = \frac{B(q^{-1})}{F(q^{-1})} u(k) + e(k)}$$

obtained for $na = nc = nd = 0$, i.e. $A = C = D = 1$.



This corresponds to simple, additive measurement noise on the output (the “output error”).

Exercise: What is the explicit form of the OE model?

Table of contents

1. Model structures
2. General prediction error methods
   - Stepping stones: ARX and ARMAX
   - General case
   - Matlab example
   - Theoretical guarantees
3. Solving the optimization problem

ARX reinterpreted as a PEM

1. Compute predictions at each step, $\widehat{y}(k) = \varphi^\top(k)\theta$ given parameters θ.
2. Compute prediction errors at each step, $\varepsilon(k) = y(k) - \widehat{y}(k)$.
3. Find a parameter vector θ minimizing criterion $V(\theta) = \frac{1}{N}\sum_{k=1}^{N} \varepsilon^2(k)$.

The procedure above is just a reinterpretation, equivalent to the algorithm discussed in the ARX lecture.

Prediction error methods are obtained by extending this procedure to general model structures.

ARX reinterpreted as a PEM (continued)

Remarks:

- For ARX, we already know how to minimize the MSE (from linear regression); for more general models new methods will be introduced.
- ARX predictor $\widehat{y}(k)$ is chosen to achieve the error $\varepsilon(k) = e(k)$, equal to the noise. We will aim to achieve the same error in the general PEM, intuitively because we cannot do better.
  Note different meanings of $\varepsilon(k)$ (prediction error) and $e(k)$ (noise)
- The prediction error is for ARX just a rearrangement of the equation $y(k) = \varphi^\top(k)\theta + \varepsilon(k) = \widehat{y}(k) + \varepsilon(k)$.

Intermediate step: PEM for 1st order ARMAX

To make things easier to follow, we first derive PEM for a 1st order ARMAX.

Looking at slide ARMAX: explicit form, the 1st order ARMAX can be written as follows:

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

1st order ARMAX: predictor

To achieve error $e(k)$, the predictor must be:

$$\widehat{y}(k) = -ay(k-1) + bu(k-1) + ce(k-1) \tag{7.1}$$

This depends on unknown noise $e(k-1)$. However, we derive a *dynamical*, recursive predictor formula that does not.

$$\widehat{y}(k-1) = -ay(k-2) + bu(k-2) + ce(k-2) \tag{7.2}$$

From Eqn. (7.1) $+c$ Eqn. (7.2):

$$\begin{aligned}
&\widehat{y}(k) + c\widehat{y}(k-1) \\
&= -ay(k-1) + bu(k-1) + ce(k-1) \\
&\quad + c(-ay(k-2) + bu(k-2) + ce(k-2)) \\
&= -ay(k-1) + bu(k-1) + ce(k-1) \\
&\quad + c(-ay(k-2) + bu(k-2) + ce(k-2) + e(k-1) - e(k-1)) \\
&= -ay(k-1) + bu(k-1) + ce(k-1) + cy(k-1) - ce(k-1) \\
&= (c-a)y(k-1) + bu(k-1)
\end{aligned}$$

1st order ARMAX: predictor (continued)

Final recursion:

$$\widehat{y}(k) = -c\widehat{y}(k-1) + (c-a)y(k-1) + bu(k-1)$$

Requires initialization at $\widehat{y}(0)$; this initial value is usually taken 0.

1st order ARMAX: prediction error

Since we want to minimize the prediction errors $\varepsilon(k)$, we need a way to compute them as well.

Similar dynamics (recursion):

$$\begin{aligned}\varepsilon(k) &= y(k) - \widehat{y}(k)\\ \varepsilon(k-1) &= y(k-1) - \widehat{y}(k-1)\end{aligned}$$

$$\begin{aligned}\Rightarrow \varepsilon(k) + c\varepsilon(k-1) &= y(k) + cy(k-1) - (\widehat{y}(k) + c\widehat{y}(k-1))\\ &= y(k) + cy(k-1) - ((c-a)y(k-1) + bu(k-1))\\ &= y(k) + ay(k-1) - bu(k-1)\end{aligned}$$

$$\Rightarrow \varepsilon(k) = -c\varepsilon(k-1) + y(k) + ay(k-1) - bu(k-1)$$

Requires initialization of $\varepsilon(0)$, usually taken 0.

1st order ARMAX: Finding the parameters

Once a procedure to compute the errors is available, the parameters θ are found by minimizing criterion $V(\theta) = \frac{1}{N}\sum_{k=1}^{N} \varepsilon^2(k)$. This may require multiple evaluations of error signal $\varepsilon(k)$, for multiple values of the parameters θ.

(We do not yet go into specific computational methods to solve the error minimization problem. We will study them in detail in the next section.)

Finally, once an estimate $\widehat{\theta}$ of the optimum is found, the predictor formula is applied to compute the model outputs $\hat{y}(k)$.

Table of contents

1. Model structures

2. General prediction error methods
   - Stepping stones: ARX and ARMAX
   - General case
   - Matlab example
   - Theoretical guarantees

3. Solving the optimization problem

Recall: General model structure

$$y(k) = G(q^{-1})u(k) + H(q^{-1})e(k)$$

where G and H are discrete-time transfer functions:

$$y(k) = \frac{B(q^{-1})}{A(q^{-1})F(q^{-1})}u(k) + \frac{C(q^{-1})}{A(q^{-1})D(q^{-1})}e(k)$$

Prediction error

We start by deriving $e(k)$.

$$\begin{aligned} y(k) &= G(q^{-1})u(k) + H(q^{-1})e(k) \\ \Rightarrow e(k) &= H^{-1}(q^{-1})(y(k) - G(q^{-1})u(k)) \end{aligned}$$

where $H^{-1} = \frac{A(q^{-1})D(q^{-1})}{C(q^{-1})}$ is the inverse of polynomial fraction H.

The predictor will be derived so that the prediction error $\varepsilon(k) = y(k) - \widehat{y}(k) = e(k)$. So, the same formula can also be used to compute $\varepsilon(k)$:

$$\boxed{\varepsilon(k) = H^{-1}(q^{-1})(y(k) - G(q^{-1})u(k))}$$

This is a dynamical system that can be simulated to compute $\varepsilon(k)$.

Predictor

To achieve error $e(k)$, the predictor dynamics must be:

$$\begin{aligned}\widehat{y}(k) &= y(k) - e(k) = Gu(k) + He(k) - e(k) = Gu(k) + (H-1)e(k)\\ &= Gu(k) + (H-1)H^{-1}(y(k) - Gu(k))\\ &= Gu(k) + (1-H^{-1})(y(k) - Gu(k))\\ &= Gu(k) + (1-H^{-1})y(k) - Gu(k) + H^{-1}Gu(k)\\ &= \boxed{(1-H^{-1})y(k) + H^{-1}Gu(k)}\end{aligned}$$

where we skipped argument q^{-1} to make the equations readable.

Remark: In order to have a *causal* predictor, that only depends on past values of the output and input, we require $G(0) = 0$ and $H(0) = 1$.

Finding the parameters

Once a procedure to compute the predictors and errors is available, the parameters θ are found by minimizing criterion $V(\theta) = \frac{1}{N} \sum_{k=1}^{N} \varepsilon^2(k)$.

Again, linear regression will not work in general, and later we go into the computation methods.

Specializing the framework to ARX

It is instructive to see how the formulas simplify in the ARX case. Rewriting ARX in the general model template:

$$y(k) = Gu(k) + He(k) = \frac{B}{A}u(k) + \frac{1}{A}e(k)$$

We have:

$$\begin{aligned}
H^{-1} &= A \Rightarrow 1 - H^{-1} = 1 - A, \quad H^{-1}G = B \\
\widehat{y}(k) &= (1 - H^{-1})y(k) + H^{-1}Gy(k) = (1 - A)y(k) + Bu(k) \\
&= (-a_1 q^{-1} - \ldots - a_{na} q^{-na})y(k) + (b_1 q^{-1} + \ldots b_{nb} + q^{-nb})u(k) \\
&= \varphi(k)\theta \\
\varepsilon(k) &= H^{-1}(y(k) - Gu(k)) = Ay(k) - Bu(k) \\
&= y(k) - (1 - A)y(k) - Bu(k) = y(k) - \widehat{y}(k)
\end{aligned}$$

which is therefore equivalent to the ARX formulation.

Specializing the framework to 1st order ARMAX

Exercise

Plug in the polynomials for 1st order ARMAX and verify that you obtain the same dynamics (recursions) as before for the predictor and the error.

Table of contents

1. Model structures

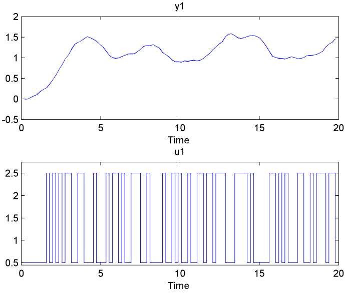
2. General prediction error methods
   - Stepping stones: ARX and ARMAX
   - General case
   - Matlab example
   - Theoretical guarantees

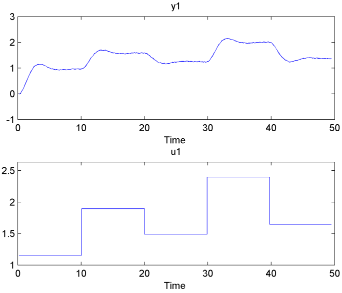
3. Solving the optimization problem

Experimental data

Consider again the experimental data on which ARX was applied.

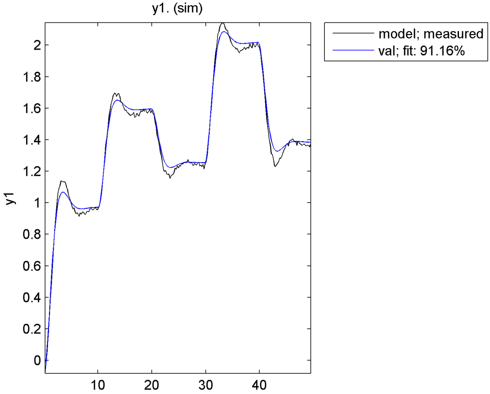
`plot(id);` and `plot(val);`





Recall: ARX result

Assuming the system is second-order and without time delay, we take $na = 2, nb = 2, nk = 1$.



Results are quite bad.

Identifying an ARMAX model

```
mARMAX = armax(id, [na, nb, nc, nk]);
```

Arguments:

1. Identification data.
2. Array containing the orders of *A*, *B*, *C* and the *delay nk*.

Like for ARX, structure includes the explicit minimum delay *nk* between inputs and outputs.

$$\begin{aligned} y(k) &+ a_1 y(k-1) + a_2 y(k-2) + \ldots + a_{na} y(k-na) \\ &= b_1 u(k-nk) + b_2 u(k-nk-1) + \ldots + b_{nb} u(k-nk-nb+1) \\ &+ e(k) + c_1 e(k-1) + c_2 e(k-2) + \ldots + c_{nc} e(k-nc) \end{aligned}$$

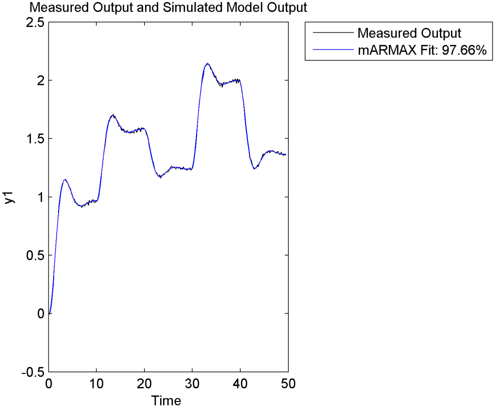
$$A(q^{-1})y(k) = B(q^{-1})u(k-nk) + C(q^{-1})e(k), \text{ where:}$$

$$\begin{aligned} A(q^{-1}) &= (1 + a_1 q^{-1} + a_2 q^{-2} + \ldots + a_{na} q^{-na}) \\ B(q^{-1}) &= (b_1 + b_2 q^{-1} + b_{nb} q^{-nb+1}) \\ C(q^{-1}) &= (1 + c_1 q^{-1} + c_2 q^{-2} + \ldots + c_{nc} q^{-nc}) \end{aligned}$$

Remark: As for ARX, the theoretical structure is obtained by setting $nk = 1$ (and to represent $nk > 1$ in the theoretical structure, change *B* like in the ARX Matlab example).

ARMAX model

Considering the system is 2nd order with no time delay, take $na = 2$, $nb = 2$, $nc = 2$, $nk = 1$. Validation: `compare(val, mARMAX);`

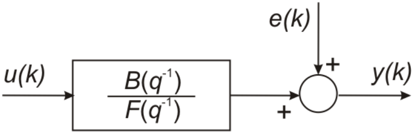


In contrast to ARX, results are good. Flexible noise model pays off.

Identifying an OE model

Recall OE model structure:

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



Identifying an OE model (continued)

```
mOE = oe(id, [nb, nf, nk]);
```

Arguments:

1. Identification data.
2. Array containing the orders of B, F, and the delay nk.

$$y(k) = \frac{B(q^{-1})}{F(q^{-1})} u(k - nk) + e(k), \text{ where:}$$

$$B(q^{-1}) = (b_1 + b_2 q^{-1} + b_{nb} q^{-nb+1})$$

$$F(q^{-1}) = (1 + f_1 q^{-1} + f_2 q^{-2} + \ldots + f_{nf} q^{-nf})$$

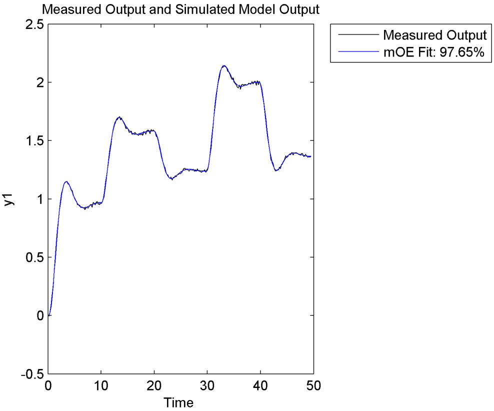
Explicit formula:

$$\begin{aligned} y(k) &+ f_1 y(k-1) + f_2 y(k-2) + \ldots + f_{nf} y(k - nf) \\ =&b_1 u(k - nk) + b_2 u(k - nk - 1) + \ldots + b_{nb} u(k - nk - nb + 1) \\ &+ e(k) + f_1 e(k-1) + f_2 e(k-2) + \ldots + f_{nf} e(k - nf) \end{aligned}$$

Remark: Like before, can transform into theoretical structure by setting $nk = 1$ (or changing B if $nk > 1$).

OE model

Considering the system is second-order with no time delay, we take $nb = 2, nf = 2, nk = 1$. Validation: `compare(val, mOE);`



Results as good as ARMAX. System turns out to obey both model structures. Question: What is the true structure then?

Table of contents

1 Model structures

2 General prediction error methods

- Stepping stones: ARX and ARMAX
- General case
- Matlab example
- Theoretical guarantees

3 Solving the optimization problem

Preliminaries: Vector derivative and Hessian

Consider *any function* $V(\theta)$, $V : \mathbb{R}^n \to \mathbb{R}$. Then:

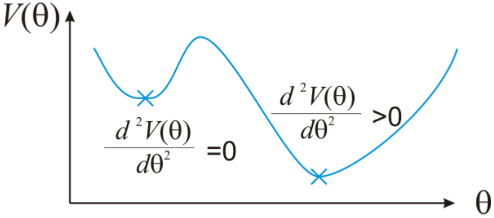
$$\frac{dV}{d\theta} = \begin{bmatrix} \frac{\partial V}{\partial \theta_1} \\ \frac{\partial V}{\partial \theta_2} \\ \vdots \\ \frac{\partial V}{\partial \theta_n} \end{bmatrix}, \quad \frac{d^2 V}{d\theta^2} = \begin{bmatrix} \frac{\partial^2 V}{\partial \theta_1^2} & \frac{\partial^2 V}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 V}{\partial \theta_1 \theta_n} \\ \frac{\partial^2 V}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 V}{\partial \theta_2^2} & \cdots & \frac{\partial^2 V}{\partial \theta_2 \theta_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 V}{\partial \theta_n \partial \theta_1} & \frac{\partial^2 V}{\partial \theta_n \theta_2} & \cdots & \frac{\partial^2 V}{\partial \theta_n^2} \end{bmatrix}$$

Assumptions

Assumptions (simplified)

1. Signals $u(k)$ and $y(k)$ are stationary stochastic processes.
2. The input signal $u(k)$ has a sufficiently high order of persistent excitation.
3. The Hessian $\frac{d^2V}{d\theta^2}$ is nonsingular at the minimum points of V.

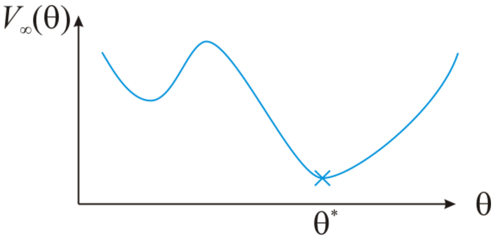
Recall $V(\theta) = \frac{1}{N}\sum_{k=1}^{N} \varepsilon^2(k)$, the MSE. Assumption 3 ensures V is not “flat” around minima.



Guarantee

Theorem 1

Define the limit $V_\infty(\theta) = \lim_{N\to\infty} V(\theta)$. Given Assumptions 1–3, the identification solution $\widehat{\theta} = \arg\min_\theta V(\theta)$ converges to a minimum point θ^* of $V_\infty(\theta)$ as $N \to \infty$.



Remark: This is a type of consistency guarantee, in the limit of infinitely many data points.

Further assumptions to strengthen guarantee

Assumptions (simplified)

4. The true system satisfies the model structure chosen. This means there exists at least one θ_0 so that for any input $u(k)$ and the corresponding output $y(k)$ of the true system, we have:

$$y(k) = G(q^{-1}; \theta_0)u(k) + H(q^{-1}; \theta_0)e(k)$$

   with $e(k)$ *white noise*.

5. The input $u(k)$ is independent from the noise $e(k)$ (the experiment is performed in open loop).

Additional guarantee

Theorem 2

Under Assumptions 1-5, $\widehat{\theta}$ converges to a true parameter vector θ_0 as $N \to \infty$.

Remark: Also a consistency guarantee. Theorem 1 guaranteed a minimum-error solution, whereas Theorem 2 additionally says this solution corresponds to the true system, *if* the system satisfies the model structure.

Table of contents

Optimization problem

Objective of identification procedure: Minimize mean squared error

$$V(\theta) = \frac{1}{N}\sum_{k=1}^{N} \varepsilon(k)^2$$

where $\varepsilon(k)$ are the prediction errors. In the general case:

$$\varepsilon(k) = H^{-1}(q^{-1})(y(k) - G(q^{-1})u(k))$$

Solution: $\widehat{\theta} = \arg\min_{\theta} V(\theta)$

So far we took this solution for granted and investigated its properties. While in ARX linear regression could be applied to find $\widehat{\theta}$, in general this does not work. Main implementation question:

How to solve the optimization problem?

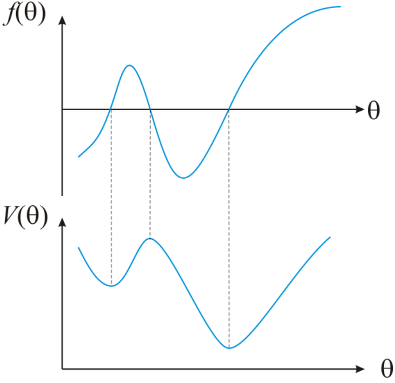
Minimization via derivative root

Consider first the scalar case, $\theta \in \mathbb{R}$.

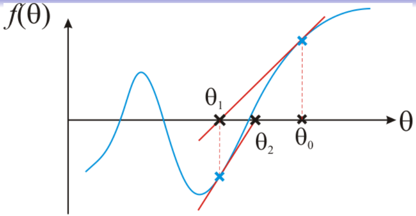
Idea: at any minimum, the derivative $f(\theta) = \frac{dV}{d\theta}$ is zero. So, find a root of $f(\theta)$.

Remarks:

- Care must be taken to find a minimum and not a maximum or inflexion. This can be checked with the second derivative, $\frac{d^2 V}{d\theta^2} = \frac{df}{d\theta} > 0$.
- We may also find a local minimum which is larger (worse) than the global one.



Newton’s method for root finding



- Start from some initial point θ_0.
- At iteration ℓ, next point $\theta_{\ell+1}$ is the intersection between abscissa and tangent at f in current point θ_ℓ. By geometry arguments:

$$\boxed{\theta_{\ell+1} = \theta_\ell - \frac{f(\theta_\ell)}{\frac{df(\theta_\ell)}{d\theta}}}$$

Remarks:

- Notation $\frac{df(\theta_\ell)}{d\theta}$ means the value of derivative $\frac{df}{d\theta}$ at point θ_ℓ.
- The slope of the tangent is $\frac{df(\theta_\ell)}{d\theta}$.
- $\theta_{\ell+1}$ is the “best guess” for root given current point θ_ℓ.

Newton's method for the optimization problem

- Replace $f(\theta)$ by $\frac{dV}{d\theta}$ to get back to optimization problem:

$$\theta_{\ell+1} = \theta_\ell - \frac{\frac{dV(\theta_\ell)}{d\theta}}{\frac{d^2V(\theta_\ell)}{d\theta^2}}$$

- Extend to $\theta \in \mathbb{R}^n$. Recall that $\frac{dV}{d\theta}$ is now an n-vector, and $\frac{d^2V}{d\theta^2}$ an $n \times n$-matrix (the Hessian). The proper extension is:

$$\theta_{\ell+1} = \theta_\ell - \left[\frac{d^2V(\theta_\ell)}{d\theta^2}\right]^{-1} \frac{dV(\theta_\ell)}{d\theta}$$

- Add a step size $\alpha_\ell > 0$. Final formula:

$$\boxed{\theta_{\ell+1} = \theta_\ell - \alpha_\ell \left[\frac{d^2V(\theta_\ell)}{d\theta^2}\right]^{-1} \frac{dV(\theta_\ell)}{d\theta}}$$

Remark:

- The stepsize helps with the convergence of the method, e.g. when V is noisy.

Stopping criterion

Algorithm can be stopped:

- When the difference between consecutive parameter vectors is small, e.g. $\max_{i=1}^{n} |\theta_{i,\ell+1} - \theta_{i,\ell}|$ smaller than some preset threshold.

or

- When the number of iterations ℓ exceeds a preset maximum.

Computing the derivatives

$$V(\theta) = \frac{1}{N} \sum_{k=1}^{N} \varepsilon(k)^2$$

Keeping in mind that $\varepsilon(k)$ depends on θ, from matrix calculus:

$$\frac{dV}{d\theta} = \frac{2}{N} \sum_{k=1}^{N} \varepsilon(k) \frac{d\varepsilon(k)}{d\theta}$$

$$\frac{d^2 V}{d\theta^2} = \frac{2}{N} \sum_{k=1}^{N} \frac{d\varepsilon(k)}{d\theta} \left[\frac{d\varepsilon(k)}{d\theta} \right]^\top + \frac{2}{N} \sum_{k=1}^{N} \varepsilon(k) \frac{d^2\varepsilon(k)}{d\theta^2}$$

where:

- $\frac{d\varepsilon(k)}{d\theta}$ is the vector derivative and $\frac{d^2\varepsilon(k)}{d\theta^2}$ the Hessian of $\varepsilon(k)$.
- $\frac{d\varepsilon(k)}{d\theta}[\frac{d\varepsilon(k)}{d\theta}]^\top$ is an $n \times n$ matrix.

Gauss-Newton

Ignore the second term in the Hessian of V and just use the first term:

$$\mathcal{H} = \frac{2}{N} \sum_{k=1}^{N} \frac{d\varepsilon(k)}{d\theta} \left[\frac{d\varepsilon(k)}{d\theta} \right]^{\top}$$

leading to the Gauss-Newton algorithm:

$$\theta_{\ell+1} = \theta_{\ell} - \alpha_{\ell} \mathcal{H}^{-1} \frac{dV(\theta_{\ell})}{d\theta}$$

Motivation:

- Quadratic form of $\mathcal{H}$ gives better algorithm behavior.
- Simpler computation.

The details of how to compute $\frac{d\varepsilon(k)}{d\theta}$ depend on the model structure chosen.

Example: 1st order ARMAX

Recall model and prediction error for 1st order ARMAX:

$$\begin{aligned} y(k) &= -ay(k-1) + bu(k-1) + ce(k-1) + e(k) \\ \varepsilon(k) &= -c\varepsilon(k-1) + y(k) + ay(k-1) - bu(k-1) \end{aligned}$$

We need $\frac{d\varepsilon(k)}{d\theta} = [\frac{\partial\varepsilon(k)}{\partial a}, \frac{\partial\varepsilon(k)}{\partial b}, \frac{\partial\varepsilon(k)}{\partial c}]^\top$. Differentiating second equation:

$$\begin{aligned} \frac{\partial\varepsilon(k)}{\partial a} &= -c\frac{\partial\varepsilon(k-1)}{\partial a} + y(k-1) \\ \frac{\partial\varepsilon(k)}{\partial b} &= -c\frac{\partial\varepsilon(k-1)}{\partial b} - u(k-1) \\ \frac{\partial\varepsilon(k)}{\partial c} &= -c\frac{\partial\varepsilon(k-1)}{\partial c} - \varepsilon(k-1) \end{aligned}$$

So, $\frac{\partial\varepsilon(k)}{\partial a}, \frac{\partial\varepsilon(k)}{\partial b}, \frac{\partial\varepsilon(k)}{\partial c}$ are dynamical signals! They can be computed using the recursions above, starting e.g. from 0 initial values.

Example: 1st order ARMAX (continued)

Finally, the overall algorithm is implemented as follows:

```
initialize $\theta_0$, iteration counter $\ell = 0$
**repeat**
    given current value of parameter vector $\theta_\ell$,
        apply recursions above to find $\frac{d\varepsilon(k)}{d\theta}$, $k = 1, \dots, n$
    plug $\frac{d\varepsilon(k)}{d\theta}$ into equations for $\frac{dV}{d\theta}$, $\frac{d^2V}{d\theta^2}$
    apply Newton (or Gauss-Newton) update formula to find $\theta_{\ell+1}$
    increment counter: $\ell = \ell + 1$
**until** $\theta_{\ell+1} - \theta_\ell$ is small enough, or maximum $\ell$ was reached
```