Embedded Information Systems

5. Quantities and variables in real-time systems

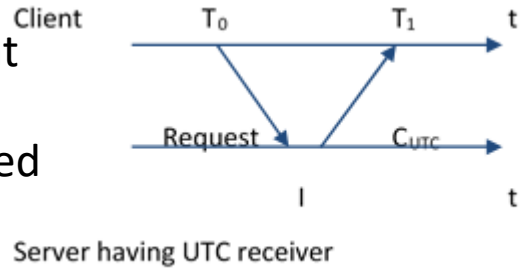
November 3, 2020

Reminder:

Clock synchronisation:

Berkeley algorithm: Active time server: regularly requests the clock values of the nodes, computes their average and resends them.

Cristian algorithm: Synchronization is initiated by the client at time T_0 by requesting a server, which has a **UTC** receiver. After the arrival of the request an interrupt routine is executed and the **UTC** radio is requested. Finally, the value of the **UTC** clock is sent to the client. The message arrives at T_1.

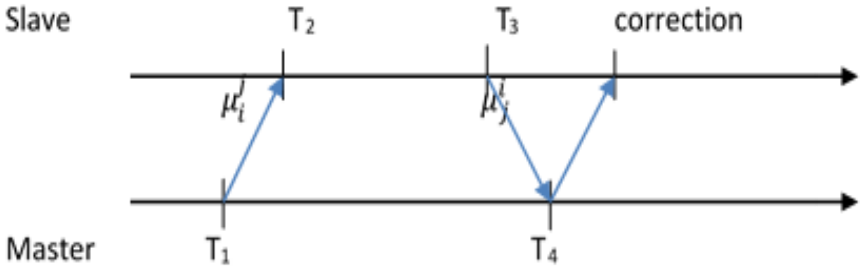


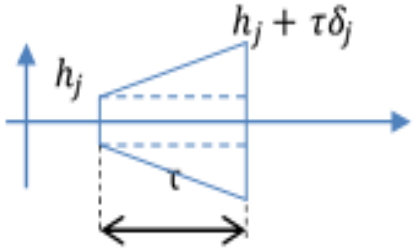
The received clock value must be corrected by the time needed for communication. If the communication requires nearly the same amount of time in both directions, then a good approximation of this correction is: $\sim \frac{T_1 - T_0 - I}{2}$.

Comment:

It might cause problems if $C_{UTC} + correction < T_1$, i.e., the clock of the client is to be reset to an earlier time. If the client clock is just timestamping subsequent events, it might happen that due to the reset a later event receives earlier time, i.e., seemingly changes the order. If such a situation is a real danger, then it is not allowed to reset the clock, only the slowing of the clock is permitted until its run will be synchronous with the **UTC** clock.

Master-slave algorithms:





Reminder:

Example: Tempo algorithms: master-slave synchronization in the distributed Berke

Master side:

The basic algorithm is repeated N times:

for *k*=1 to *N*

do

Initialization:

do

$$T_A \leftarrow C_i(now)$$

$$\forall j \neq i \text{ Send } T_A \text{ to } j$$

endo

Processing data received from the slaves:

$\forall j \neq i$:

do

$$d_2^j \leftarrow C_i(now) - T_B$$

$$\Delta_j(k) \leftarrow \left(d_1^j - d_2^j\right)/2$$

endo

endo ; *N differences are available for* $\forall j$*:*

$\forall j \neq i$:

do

$$\Delta_j = (1/N)\sum_{k=1}^{N} \Delta_j\,(k)$$

$$\text{Send}\left(\Delta_j\right) \text{ to j}$$

endo

Slave side:

do

$$d_1^j \leftarrow C_j(now) - T_A$$

$$T_B \leftarrow C_j(now)$$

Send (T_B, d_1^j) to *i*

endo

do

$$C_j(t) \leftarrow C_j(t) - \Delta_j$$

endo

Distributed clock synchronization algorithms

The major advantage of the distributed approach is the higher degree of fault tolerance it achieves. This achievement increases the cost mainly in communication rather than in special hardware. The load imposed on the communication network by the distributed approach is therefore expected to be higher than that of the MS approach.

In the distributed clock systems, all the time servers use uniform approach with the following characteristics:

- Each node polls the rest of the clocks or a subset of them.
- Each applies a specific algorithm to the responses of the poll.
- Each node updates the local clock accordingly.

I. A Fundamental Ordering Approach

Let us consider an ordering approach based on message timestamping with the following properties:

- The accuracy of clock *i* is bounded by a drift rate δ: $\forall t$: $\left|1 - \frac{d}{dt} C_i(t)\right| < \delta_i \ll 1$.
- The communication graph of the algorithm is closely connected (every vertex/node can send synchronization messages to the rest of the vertices/nodes) with a diameter *d* (minimum number of hopes).
- The network imposes an unpredictable (yet bounded) message delay *D*. In other words, $\mu < D < \eta$ holds, where μ and η are the lower and upper bounds on *D*.

Each clock implements the following algorithm:

- On every local clock event occurrence, increment the local clock $C_i(t) \leftarrow C_i(t) + 1$.

Reminder:

- Each node with a clock sends messages to the others at least every τ seconds. Each message includes its timestamp T_m.
- Upon reception of an external T_m, the receiver sets its clock $C_i(t) \leftarrow max(C_i(t), T_m + \mu)$.

The communication cost of one update of the whole network is $n(n-1)$ messages.

The correctness of each clock due to this synchronization algorithm is $\forall i$: $\forall j$: $|C_i(t) - C_j(t)| < d(2\delta\tau + \eta)$ for all t.

This algorithm achieves only the ordering goal, bounding clock differences between sites. The algorithm results in updates according to the fastest clock in the system, and not necessarily the most accurate one.

II. Minimize Maximum Error

Every clock i "knows" it is correct within the interval: $[C_i(t) - E_i(t), C_i(t) + E_i(t)]$ where $E_i(t)$ is a bound on the error of clock i.

The error interval is constructed from the following contributors:

- The error that comes into effect right on the clock reset time (ρ_i), as discretization and other constant errors (ε_i).
- The delay from the time this clock i is read until another clock j uses the readout for its update (μ_i^j).
- The degradation of time-counting that develops between consecutive resets (δ_i).

The algorithm consists of two rules: a **response rule** and a **synchronizer rule**.

A request is transmitted by the **synchronizer rule** at node j activates i's response.

Reminder:

Minimize Maximum Error:

Upon receiving a time Request from $j \neq i$:

do

$E_i(t) \leftarrow \varepsilon_i + (C_i(t) - \rho_i)\delta_i$

Send $(C_i(t), E_i(t))$ to j.

endo

Rule#1 (from the viewpoint of clock i).

At least once every τ time units:

$\forall j \neq i$: Request$\big(C_j(t), E_j(t)\big)$;

for $j \neq i$ do begin

Receive$\big(C_j(t), E_j(t)\big)$;

if $\big(C_j(t), E_j(t)\big)$ is consistent with $(C_i(t), E_i(t))$

then if $E_j(t) + (1 + \delta_i)\mu_j^i \leq E_i(t)$

then begin

$C_i(t) \leftarrow C_j(t)$

$\varepsilon_i \leftarrow E_j(t) + (1 + \delta_i)\mu_j^i$

$\rho_i \leftarrow C_j(t)$

end

else ignore it

end

endo

Rule#2

III. Intersection of time intervals

Reminder:

This algorithm also consists of two rules: a response rule and a synchronizer rule.
The response rule is identical to the response of the previous algorithm.
The synchronizer rule here is also periodic, performed at least every τ time units.
The first step of the synchronizer rule is a request for responses, which the algorithm sends to the rest of the nodes. The similarity to the previous algorithm ends here.
The second step of this algorithm is to receive all the responses.

Each response interval has a **left boundary** $L_j(t)$ and a right boundary $R_j(t)$ and the algorithm calculates both of them.

Then the algorithm selects the highest left boundary in the responses, α, and the lowest **right boundary**, β. If the responses are **consistent**, there must be a nonempty intersection of them all, and thus, $\alpha < \beta$. Otherwise the responses are considered inconsistent and therefore ignored.
If they are **consistent**, we can conclude that the real-time clock is within the interval $[\alpha, \beta]$.
Therefore, the algorithm sets its error to equal half this interval and the local clock to equal the interval's midpoint.

The first rule is exactly the same as previously.

The second rule:

At least once every τ time units:

$\forall j \neq i$: Request$\big(C_j(t), E_j(t)\big)$;
$\forall j \neq i$: Receive$\big(C_j(t), E_j(t)\big)$;
$\forall j \neq i\ L_j(t) \leftarrow \big(C_j(t) - E_j(t)\big)$; the left boundary
$\forall j \neq i$: $R_j(t) \leftarrow \big(C_j(t) + E_j(t)\big) + (1 + \delta_i)\mu_j^i$
the right boundary
$\alpha \leftarrow max\big(L_j\big); \beta \leftarrow min\big(R_j\big)$
if $\alpha < \beta$
then
$\varepsilon_i \leftarrow \frac{1}{2}(\beta - \alpha)$;
$C_i(t) \leftarrow \frac{1}{2}(\alpha + \beta)$;
$\rho_i \leftarrow \frac{1}{2}(\alpha + \beta)$
end
else ignore them all
endo

Reminder:

Comments:

(1) The intersection algorithm is superior in its accuracy, however, less robust. It may ignore the responses of all the participants because of an erroneous response from one participant.

(2) The communication demand of the distributed algorithm in case of *n* clocks is $2n(n-1)$ in every τ time units.

(3) The distributed algorithms require good knowledge of time.

Jitter of the synchronization message: *Jitter*: $e = d_{max} - d_{min}$

- at the application software level: $500\mu s$ …5ms
- at the kernel of the operating system: $10\mu s$ …$100\mu s$
- at hardware of the communication controller: <$10\mu s$.

The important role of the latency jitter ε for internal synchronization is emphasized by an impossibility result: It is not possible to internally synchronize the clocks of an ensemble consisting of *N* nodes to a better precision than $\Pi = e\left(1 - \frac{1}{N}\right)$.

IV. Fault-Tolerant-Average (FTA) algorithm:

In a system with *N* nodes *k* Byzantine faults should be tolerated.
The FTA algorithm is a one-round algorithm that works with inconsistent information, and bounds the error introduced by inconsistency. At every node, the *N* measured time differences between the node's clock and the clocks of all other nodes are collected (the node considers itself a member of the ensemble with time difference zero).

Reminder:

These time differences are sorted by size.

Then the k largest and the k smallest differences are removed (if the erroneous time value is either larger or smaller than the rest).
The remaining $N - 2k$ time differences are, by definition, within the precision window.

The average of these remaining time differences is the correction term for the node's clock.

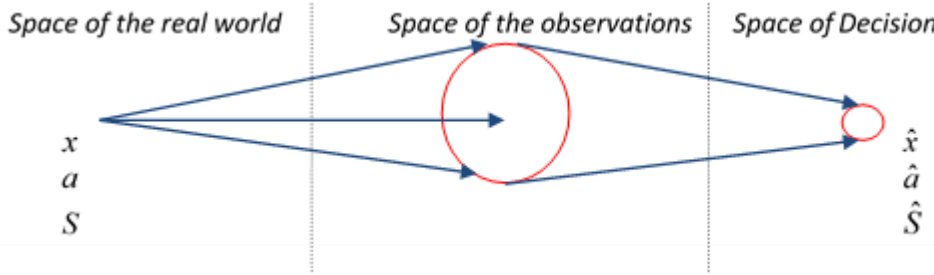
5. Quantities and variables in real-time systems

Known concepts: Real-time variable, real-time image, temporal accuracy, periodic update, observation: state observation, event observation, real-time object, permanence, action delay, idempotence, credibility (Byzantine errors), sphere of control, …

Modelling of the recipient environment: **What can not be avoided:**

The **cognition** of the recipient environment, and its **installation** into the software.

An important tool of this latter is the measurement process: which is an inherent part of the cognition process within which we are increasing and expanding our knowledge.



The figure helps the **interpretation.**

While taking measurement, we try to **grasp** the different **phenomena** of the real world.

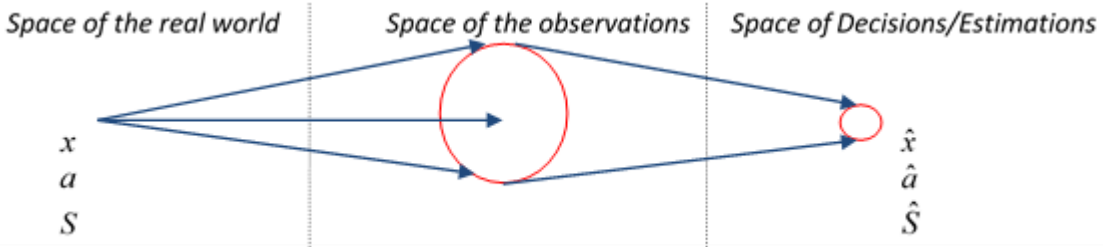
This is made preferably by quantities which show **stability.**

Obviously, such quantities are results of **abstractions.**

The following **quantities/features** play key role:

- ***state variables*** (*x*), the changes of which follow energy processes (voltage, pressure, temperature, speed, etc.) due to interactions;
- ***parameters*** (*a*), which characterize the strength of the interactions;
- ***structures*** (***S***), which describe the relations of the system components.

The ***Space of the real world*** is such an abstraction, where the values of the investigated features correspond to one point of the space.



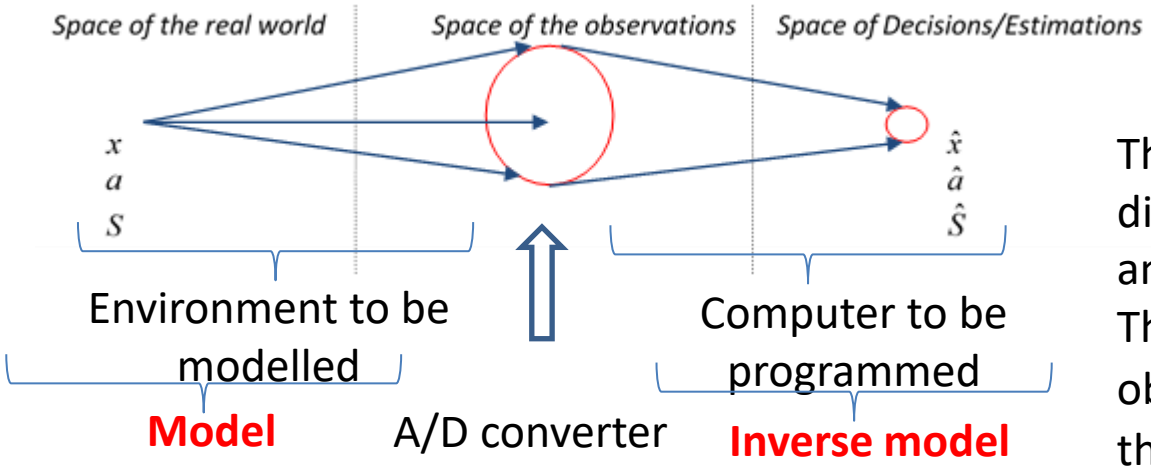
The coordinates of this points **are unknown** before the measurement.

It is well known, that due to measurement errors, only **an estimate** of the measurand can be provided.

Further difficulty, that there is no direct access to the quantity to be measured, only some kind of **indirect mapping** is possible.

This mapping is called **observation.**

The path from the quantity to be measured and the observation is called **measuring channel**.



Observation in case of deterministic channel:

The observed reality is described by a discrete model, and it is supposed to be an autonomous system.

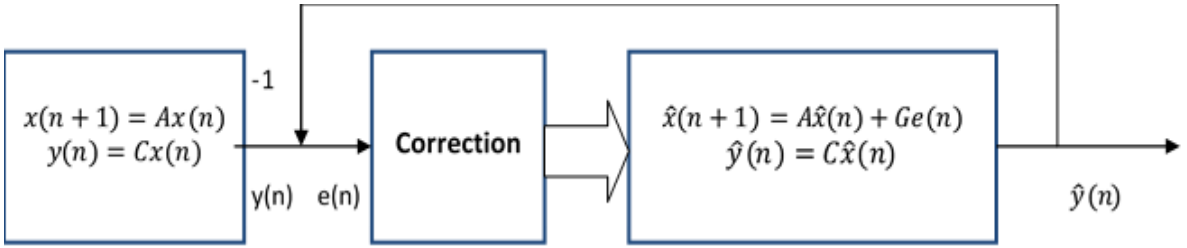
The state equations and the observation equation describing the reality and the observation:

$$x(n+1) = Ax(n), \quad dim[x(n)] = N, \quad dim[A] = N * N$$

$$y(n) = Cx(n) \quad dim[y(n)] = M, \quad M \leq N, \; dim[C] = M * N$$

Can we invert such a system? **Generally not!** When is it possible? If C^{-1} exists!

If C^{-1} **does not exist**, then something else is to do! The solution is a **simulator!**
The model should be built into the computer! This is controlled by the observed values!



Our aim is the estimation of the **state variable *x(n)*.**

The name of this device is: **observer,** which tries to produce **a copy** of the reality by following it thanks to a correction/training/adaptation mechanism, and gives **an estimate** of the value to be measured. This estimator $\hat{x}(n)$ **can be read** from the observer.

The state and the observation equations of the observer are:

$$\hat{x}(n+1) = A\hat{x}(n) + Ge(n)$$

$$\hat{y}(n) = C\hat{x}(n)$$

G: **correction matrix**; $dim[G] = N * M$, $e(n) = y(n) - \hat{y}(n)$

Matrix G should be designed to get: $\hat{x}(n) \rightarrow x(n)$.

The difference of the state variables is to be minimized:

$$x(n+1) - \hat{x}(n+1) = Ax(n) - A\hat{x}(n) - Ge(n) = (A - GC)(x(n) - \hat{x}(n)).$$

$$\varepsilon(n+1) = F\varepsilon(n)$$

$\varepsilon(n+1)$ F $\varepsilon(n)$

Is the state equation of the **error system**.

The design of matrix G: $\varepsilon(n) \xrightarrow[n\to\infty]{} 0$, therefore $\|\varepsilon(n+1)\| < \|\varepsilon(n)\|$, is forced, possibly for $\forall\, n$. F reduces the size of vector, i.e. it is „**contractive**". This property can be interpreted also in such a way that the internal energy of the error system is **dissipated**.

If this is the case in every step, then the decrease of the size of the error vector will be a **monotonic process.**

Special cases:

1. $F = A - GC = 0$. In this case $G = AC^{-1}$. This is possible if C is a square matrix, i.e. the observation has as many components as the state vector itself.

The equation can be solved in an explicite way! The system converges in one step!

2. $F^N = (A - GC)^N = 0$. In this case the error system converges in *N* steps:

$$x(N) - \hat{x}(N) = (A - GC)^N (x(0) - \hat{x}(0)) = 0$$

The matrices wth property $F^N = 0$ can be characterized by the fact that **all they eigenvalues are zero**.

Systems having state transition matrix of this property are of **finite impulse response** (**FIR**) systems, the initial error will disappear in finite steps.

3. If $F^N = (A - GC)^N \neq 0$, then the size of the state vector of a stabile error system will decrease **exponentially**. The error system is stable, if all its eigenvalue is within the unit circle. Systems having state transition matrix of this property have **infinite impulse response** (**IIR** systems), because the initial error **will disappear in infinite steps.**

Observation in the case of noisy observation channel:

In this case our expectation is not $\|\varepsilon(n)\| \xrightarrow[n\to\infty]{} 0$, but the trace of $E\{[\![\varepsilon(n)\varepsilon^T(n)]\!]\} \xrightarrow[n\to\infty]{} min$.

$$\varepsilon(n) = \begin{bmatrix} \varepsilon_0(n) \\ \varepsilon_1(n) \\ \vdots \\ \varepsilon_{N-1}(n) \end{bmatrix}$$

therefore $trace\{E[\varepsilon(n)\varepsilon^T(n)]\} = \sum_{k=0}^{N-1} \varepsilon_k^2(n)$.

Instead of $\varepsilon(n+1) = F\varepsilon(n)$

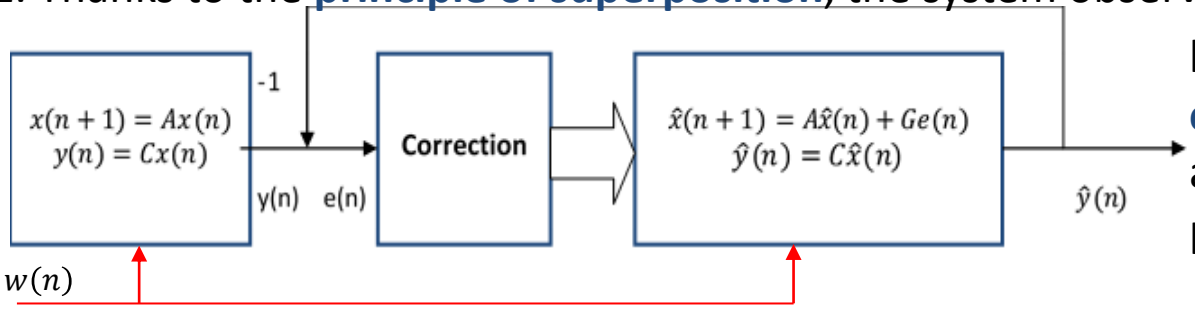
$$E[\varepsilon(n+1)\varepsilon^T(n+1)] = FE[\varepsilon(n)\varepsilon^T(n)]F^T$$

Is used for error system characterization.

Matrix $P=E[\varepsilon(n)\varepsilon^T(n)]$ plays a central role in the famous Kalman predictor and filter.

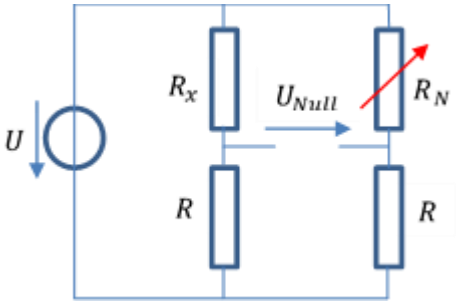
Comments:

1. Thanks to the **principle of superposition**, the system observed and the observer itself can have an additional, **common external excitation** without any change in the convergence properties.



2. The above introduced observer is called Luenberger observer. According to Luenberger **almost any system is an observer**. The only requirement is that the observer should be faster than the observed system; otherwise it will not be able to follow its changes.

3. The bridge-branch containing the **impedance to be measured** within an impedance measuring bridge implements the **physical model of the reality**, while the **tuneable bridge-branch** correspond to the model built into the **observer.**



The difference between the outputs of the voltage divider bridge-branches controls the correction mechanism. Finally the value of the unknown impedance will be computed from the correction value.

This setup, together with the operator responsible for tuning, implements an observer.

Example:

Given $A=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}; C=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$

How to set matrix G?

$$G=AC^{-1}=A=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

Example: Given $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$; $C = [1 \quad 1]$. How to set matrix G? $G = \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} = ?$

$GC = \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} [1 \quad 1] = \begin{bmatrix} g_0 & g_0 \\ g_1 & g_1 \end{bmatrix}$, $[A - GC] = \begin{bmatrix} 1 - g_0 & -g_0 \\ -g_1 & -1 - 1g_1 \end{bmatrix}$, based on $[A - GC]^2 = 0$

$$\begin{bmatrix} 1 - g_0 & -g_0 \\ -g_1 & -1 - g_1 \end{bmatrix} \begin{bmatrix} 1 - g_0 & -g_0 \\ -g_1 & -1 - g_1 \end{bmatrix} = \begin{bmatrix} 1 - 2g_0 + g_0^2 + g_0 g_1 & -g_0 + g_0^2 + g_0 + g_0 g_1 \\ -g_1 + g_1^2 + g_1 + g_0 g_1 & 1 + 2g_1 + g_1^2 + g_0 g_1 \end{bmatrix} =$$

$= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ Substituting expressions of the minor diagonal into the main diagonal we get:

$1 - 2g_0 = 0,$ $1 + 2g_1 = 0,$ where from: $g_0 = 0.5$ és $g_1 = -0.5$.

Checking: $\begin{bmatrix} 0.5 & -0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 \\ 0.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Example: Let us compute eigenvalues of matrix $[A - GC]$: $det[\lambda I - A + GC] = 0$

$$det \begin{bmatrix} \lambda - 0.5 & 0.5 \\ -0.5 & \lambda + 0.5 \end{bmatrix} = (\lambda - 0.5)(\lambda + 0.5) + 0.25 = \lambda^2 - 0.25 + 0.25 = 0.$$

Both eigenvalues are zero.

Comments:

1. This property is **valid** in every system capable to converge **in finite steps**.
2. The transfer function of such systems is a rational function having **all its poles at the origin**:

$$H(z) = a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N} = \frac{a_N + a_{N-1} z + a_{N-2} z^2 + \ldots + a_1 z^{N-1}}{z^N}$$

These are the so-called **Finite Impulse Response** (**FIR**) filters.

The time-domain equivalent: $y(n) = a_1 x(n-1) + a_2 x(n-2) + \ldots + a_N x(n-N)$,

where **due to computability reasons** only previous samples of $x(n)$ are used.

Example: The **condition for the eigenvalues** can be used to compute g_0 and g_1:

$$det[\lambda I - A + GC] = 0,\ \ det\begin{bmatrix}\lambda - 1 + g_0 & g_0 \\ g_1 & \lambda + 1 + g_1\end{bmatrix} = \lambda^2 + \lambda(g_0 + g_1) + g_0 - g_1 - 1 =$$

$= \lambda^2 = 0.$ Where from: $g_0 + g_1 = 0$ and $g_0 - g_1 = 1,$ thus: $g_0 = 0.5$ és $g_1 = -0.5.$

It happens that the dynamics of recipient environment is not known:
The state equation is not known or $\boxed{A = I.}$ In this case we have only observations.

Linear Least Squares (LS) Estimation: No a priori information is available neither from the parameter to be measured, nor from the channel characteristics/noise. Let us assume that the observation equation is linear: $y(n) = Cx(n) + w(n);\ w(n)$ is the observation noise. We assume that the unknown $x(n)$ takes value $\hat{x}(n)$, and we set up **the model of the observation.** We compare this with the observation, and we are looking for the best setting of $\hat{x}(n)$ assuming squared error: $J\big(x(n), \hat{x}(n)\big) = \big(y(n) - C\hat{x}(n)\big)^T\big(y(n) - C\hat{x}(n)\big) =$

$= y(n)^T y(n) - y(n)^T C\hat{x}(n) - \hat{x}(n)^T C^T y(n) + \hat{x}(n)^T C^T C\hat{x}(n) =$

$= y(n)^T y(n) - 2\hat{x}(n)^T C^T y(n) + \hat{x}(n)^T C^T C\hat{x}(n)$ The minimum of which is given by:

$$\left.\frac{\partial J(x(n), \hat{x}(n))}{\partial \hat{x}(n)}\right|_{\hat{x}(n)=\hat{x}_{LS}} = 0 \qquad -2C^T y(n) + 2C^T C\hat{x}(n) = 0, \qquad \boxed{\hat{x}(n) = [C^T C]^{-1} C^T y(n)}$$

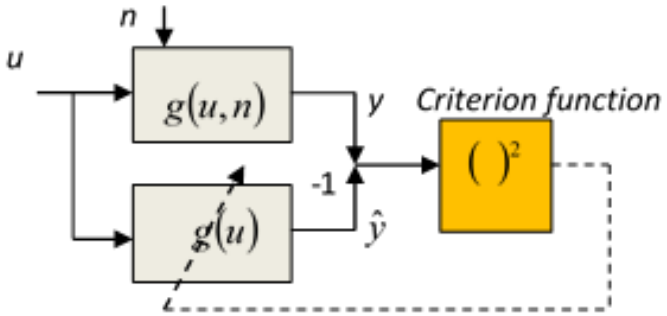
Due to $A = I$ here $x(n)$ is practically a constant parameter.
We take more observations, and the (noisy) observed values are collected in vector $y(n)$.
If $C = [1 \quad 1 \quad \cdots \quad 1]^T$, then $\boxed{\hat{x}_{LS} = \frac{1}{N}\sum_{k=0}^{N-1} y_k(n)}$ i.e. the result is simple **linear averaging.**

Model fitting:

In the case of LS estimators, we do not have prior information about parameter to be measured, therefore what we do is **model fitting**.

The problem of model fitting is manifold.

A classical version is **regression calculus**: The determination of a possibly deterministic relation of independent and dependent variables can be considered as a special case of model fitting.



In the figure below the function to be modelled $y = g(u,n)$ has two types of independent variables: the one denoted by $u(n)$, is known and **can be influenced**, while the other, denoted by $n(n)$, is unknown, and **cannot be influenced.** This latter is typically a noise process, or disturbance modelled as a noise process.

For modelling we use a "tuneable" function $\hat{y} = \hat{g}(u)$ the free parameters of which are tuneable. We aim at applying such setting of parameters which is optimal in some sense.

Typically quadratic criterion is applied:

$$J = E\{(y-\hat{y})^T(y-\hat{y})\}$$

Linear regression: The function to be fitted is a scalar linear function $\hat{g}(u) = a_0 + a_1 u$, the parameters of which are set to minimize $E\left\{\left(y-\hat{g}(u)\right)^2\right\}$.

To get the minimum of $J = E\{(y-a_0-a_1u)^2\}$ we take the derivative to a_0 and a_1 :

$$\frac{\partial J}{\partial a_0} = -2E\{(y-a_0-a_1u)\} = -2(E\{y\}-a_0-a_1E\{u\}) = 0$$

$$\frac{\partial J}{\partial a_1} = -2E\{u(y-a_0-a_1u)\} = -2(E\{uy\}-a_0E\{u\}-a_1E\{u^2\}) = 0$$

Polynomial regression: Important property that it is linear in its parameters.

$$\hat{g}(u) = \sum_{k=0}^{N} a_k u^k$$

We prefer **models linear in their parameters**, because in case of squared error criterion, finding the optimum requires the solution of a set of **linear equations**.

Linear regression based on measured data:

The above development can be carried out also for the case where no a priori information is available. In this case $y_n = a_0 + a_1 u_n + w_n$,

which is the model of the observation at the n-th time instant.

Let us take N observations! The input and the observed samples are ordered into vectors.

$$\boldsymbol{z} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & u_0 \\ 1 & u_1 \\ \vdots & \vdots \\ 1 & u_{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} + \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix}$$

$$y(n) = C * x(n) + w(n)$$

We recognize that this structure is the same as the model of the LS estimation!

$$\hat{x}(n) = [C^T C]^{-1} C^T y(n)$$

$$[\boldsymbol{C}^T \boldsymbol{C}] = \begin{bmatrix} N & \sum_{n=0}^{N-1} u_n \\ \sum_{n=0}^{N-1} u_n & \sum_{n=0}^{N-1} u_n^2 \end{bmatrix},$$

$$\boldsymbol{C}^T \boldsymbol{z} = \begin{bmatrix} \sum_{n=0}^{N-1} y_n \\ \sum_{n=0}^{N-1} u_n y_n \end{bmatrix},$$

$$\begin{bmatrix} \hat{a}_0 \\ \hat{a}_1 \end{bmatrix} = \frac{1}{\frac{1}{N}\sum_{n=0}^{N-1} u_n^2 - \left(\frac{1}{N}\sum_{n=0}^{N-1} u_n\right)^2} \begin{bmatrix} \frac{1}{N}\sum_{n=0}^{N-1} u_n^2 & -\frac{1}{N}\sum_{n=0}^{N-1} u_n \\ -\frac{1}{N}\sum_{n=0}^{N-1} u_n & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{N}\sum_{n=0}^{N-1} y_n \\ \frac{1}{N}\sum_{n=0}^{N-1} u_n y_n \end{bmatrix},$$

We recognize the approximations of the statistical characterizations!