AR03_txt_STR

štvrtok 22. februára 2024 7:44

AR03_txt_STR

O samonastavujúcom sa regulátore

Obsah

1 Konkrétny príklad

Princíp samonastavujúceho sa regulátora (Self-Tuning Regulator, i.e. STR) spočíva v tom, že v každej perióde vzorkovania sa identifikujú parametre modelu riadeného systému a následne, s využitím identifikovaných parametrov modelu, sa pomocou určitej metódy vypočítajú parametre regulátora.

Hneď v prvej vete sme použili pojem perióda vzorkovania. Je teda zrejmé, že celý riadiaci systém bude pracovať v diskrétnej časovej oblasti. Modelom riadeného systému bude ARX (Auto Regressive eXogenous) model. Štruktúra riadenia bude tzv. „trojzložková“. Tromi zložkami sú polynómy R, S a T. Tieto polynómy majú svoje koeficienty, ktoré sú zároveň parametrami riadiacej štruktúry, vlastne parametrami regulátora. Skonkretizujme teraz tento všeobecný popis.

1.1 Riadený systém

Riadený systém nech predstavuje prenosová funkcia v tvare

$$G(s) = \frac{0,15}{s^2 + 0,3s + 0,2} \tag{1}$$

Ide o prenosovú funkciu druhého rádu, ktorá má v čitateli polynóm nultého stupňa, teda nemá nuly.

1.2 ARX model

Nech modelom riadeného systému je ARX (AutoRegressive eXogenous) model. Vo všeobecnosti ARX model má tvar

$a_1 z^{-1} y$

$$A(z^{-1})y(k) = B(z^{-1})u(k) + \xi(k) \tag{2}$$

$\boxed{u(k)}$

kde

$$\begin{aligned} B(z^{-1}) &= b_1 z^{-1} + \ldots + b_{n_b} z^{-n_b} \\ A(z^{-1}) &= 1 + a_1 z^{-1} + \ldots + a_{n_a} z^{-n_a} \end{aligned} \tag{3}$$

a $\xi(k)$ predstavuje poruchy a šum. V ďalšom budeme uvažovať $\xi(k) = 0$. ARX model vyjadrený v tvare diferenčnej rovnice:

$$y(k) = -a_1 y(k-1) - \ldots - a_{n_a} y(k - n_a) + b_1 u(k-1) + \ldots + b_{n_b} u(k - n_b) \tag{4}$$

Nech modelom sústavy je diferenčná rovnica v tvare (4), kde hodnoty $n_a = 2$ a $n_b = 2$. Potom táto diferenčná rovnica má tvar

$-\Theta^T \omega$

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_1 u(k-1) + b_2 u(k-2) \tag{5}$$

V maticovom zápise:

$$\boxed{y(k) = h^{\mathsf{T}} \Theta} \tag{6}$$

$\Theta^T h$

kde $h^{\mathsf{T}} = \begin{bmatrix} -y(k-1) & -y(k-2) & u(k-1) & u(k-2) \end{bmatrix}$ a $\Theta = \begin{bmatrix} a_1 & a_2 & b_1 & b_2 \end{bmatrix}^{\mathsf{T}}$. Neznámymi parametrami modelu teda sú koeficienty a_1, a_2, b_1 a b_2.

1.3 Štruktúra riadenia

Štruktúra riadenia je zrejmá zo zápisu „trojzložkového" zákona riadenia

$$\begin{aligned} R(z^{-1})u(k) &= T(z^{-1})r(k) - S(z^{-1})y(k) \\ u(k) &= \frac{T(z^{-1})}{R(z^{-1})} r(k) - \frac{S(z^{-1})}{R(z^{-1})} y(k) \end{aligned} \tag{7}$$

kde R, S a T sú polynómy v tvare

$$\begin{aligned} R(z^{-1}) &= \boxed{1} + r_1 z^{-1} + r_2 z^{-2} + \ldots + r_{n_r} z^{-n_r} \\ S(z^{-1}) &= s_0 + s_1 z^{-1} + s_2 z^{-2} + \ldots + s_{n_s} z^{-n_s} \\ T(z^{-1}) &= t_0 + t_1 z^{-1} + t_2 z^{-2} + \ldots + t_{n_t} z^{-n_t} \end{aligned} \tag{8}$$

Ako už bolo uvedené, koeficienty polynómov sú parametrami regulátora. Počet parametrov regulátora závisí od stupňa jednotlivých polynómov. Pre tento príklad sú stupne polynómov nasledovné: $\boxed{n_r = 1}$, $n_s = 1$ a $n_t = 0$. Potom počet parametrov regulátora je $n_r + n_s + 1 + n_t + 1$. Teda $1 + 1 + 1 + 0 + 1 = 4$.

Zákon riadenia je možné zapísať aj v tvare diferenčnej rovnice

$$u(k) = -r_1 u(k-1) - s_0 y(k) - s_1 y(k-1) + t_0 r(k) \tag{9}$$

$u = \Theta^T \omega$

1.4 Výpočet parametrov regulátora

Metóda, pomocou ktorej sa v tomto prípade budú počítať parametre regulátora bude *Pole placement* – Metóda rozmiestňovania pólov uzavretého regulačného obvodu (URO). Ako už názov naznačuje, metóda spočíva v predpísaní rozmiestnenia pólov URO. Inými slovami, predpisujú sa korene charakteristického polynómu URO. Voľbou polohy pólov URO je možné zvoliť dynamické vlastnosti URO. Poloha pólov sa zadáva zvolením *želaného polynómu*, ktorý má také korene, aké si želáme. Ak napr. žiadame, aby URO mal dynamiku druhého rádu, tak želaný polynóm bude 2. stupňa.

Pripomeňme, že v tomto prípade ide o „diskrétny" polynóm, pretože riadiaci systém pracuje v diskrétnej oblasti, t.j. model sústavy je „diskrétny" a aj zákon riadenia je „diskrétny".

Parametre regulátora sa vypočítajú z porovnania koeficientov pri rovnakých mocninách charakteristického polynómu URO a želaného polynómu. Charakteristický polynóm URO sa bude skladať z polynómov modelu sústavy a zo zložiek regulátora, čo sú polynómy vystupujúce v zákone riadenia. Koeficienty polynómov modelu sústavy považujeme za známe, pretože ich získame identifikáciou. Koeficienty polynómov zo zákona riadenia – parametre regulátora sú neznáme. Získame ich riešením rovnice, kde na jednej strane je charakteristický polynóm URO a na druhej strane je želaný polynóm. Takáto rovnica sa nazýva *Diofantická rovnica*.

2 Identifikácia parametrov lineárneho modelu

Uvažujeme lineárny model v tvare (6). Parametre modelu budeme určovať *rekurzívnou metódou najmenších štvorcov*. Najskôr však odvodíme tzv. Off – line odhad parametrov modelu.

2.1 Metóda najmenších štvorcov

Predpokladajme, že sme spravili N experimentov – meraní. Napr.: na vstup sústavy sme priviedli „vhodný" signál a s nejakou periódou vzorkovania sme zaznamenávali hodnoty vstupného aj výstupného (zo sústavy) signálu. Teda máme nameraných N hodnôt vstupu, ku ktorým prislúcha N hodnôt výstupu.

V i-tom experimente sme namerali hodnotu výstupného signálu y_i. Nech model má odhadnuté „nejaké" parametre. Potom ak privedieme na vstup modelu hodnotu u_i, čo je nameraná hodnota vstupného signálu prislúchajúca k y_i, na výstupe modelu bude odhad $\hat{y}_i$. Tento odhad bude vo všeobecnosti rozdielny od nameranej hodnoty. Namiesto „nejakých" hodnôt parametrov modelu určme také, pre ktoré bude platiť, že suma štvorcov odchýlok vo všetkých nameraných bodoch bude minimálna.

Odchýlka v i-tom experimente je $e_i = y_i - \hat{y}_i$. Odhad výstupu v i-tom experimente je $\hat{y}_i = h_i^\mathsf{T}\Theta$, čo je rovnaký zápis ako v (6), len namiesto času k je použitý index i. Všetky odchýlky v N meraniach sú:

k

k

$$\begin{aligned} e_1 &= y_1 - h_1^\mathsf{T}\Theta \\ &\vdots \\ e_N &= y_N - h_N^\mathsf{T}\Theta \end{aligned} \qquad \boxed{\hat{y}_1 = h_1^T \Theta} \tag{10}$$

$\hat{y}(k) = h_1^T$

čo možno zapísať aj takto:

$[y(k-1)]$

$$e = y - \begin{bmatrix} h_1^\mathsf{T} \\ \vdots \\ h_N^\mathsf{T} \end{bmatrix} \Theta \tag{11}$$

$\hat{y}(k-1)$

Vektor h_1^T, za predpokladu, že začiatočný čas je $t_0 = 0$ a prvá vzorka vstupu a výstupu bola nameraná v čase $t(k=1) = 1 \cdot T_{vz}$ je $h_1^\mathsf{T} = [-y(0) \quad 0 \quad u(0) \quad 0]$. Analogicky $h_5^\mathsf{T} = h(5)^\mathsf{T} = [-y(4) \quad -y(3) \quad u(4) \quad u(3)]$.

Pre lepšiu názornosť nech je nameraných 5 vzoriek (a tiež hodnoty $y(0)$, $u(0)$).

Potom všetky odchýlky je možné zapísať takto:

$Ax - b = 0$

$Ax = b$

$$e = y - \begin{bmatrix} -y(0) & 0 & u(0) & 0 \\ -y(1) & -y(0) & u(1) & u(0) \\ -y(2) & -y(1) & u(2) & u(1) \\ -y(3) & -y(2) & u(3) & u(2) \\ -y(4) & -y(3) & u(4) & u(3) \end{bmatrix} \Theta \tag{12}$$

Všeobecný zápis

$$e = y - H\Theta \tag{13}$$

$e^\top e$

Ako už bolo uvedené, vektor Θ, čo je vektor parametrov modelu určíme tak, aby suma štvorcov odchýlok bola minimálna. Zaveďme účelovú funkciu v tvare

$$J(\Theta) = \frac{1}{2} e^\top e = \frac{1}{2}(y - H\Theta)^\top (y - H\Theta) \tag{14}$$

Je potrebné nájsť extrém tejto účelovej funkcie, čo znamená derivovať ju podľa vektora parametrov modelu.

$$J(\Theta) = \frac{1}{2}(y - H\Theta)^\top (y - H\Theta) = \frac{1}{2}\left(y^\top y - y^\top H\Theta - \Theta^\top H^\top y + \Theta^\top H^\top H\Theta\right) \tag{15}$$

S využitím rovnosti

$ax^2 = a2x$

$$\nabla_x \left(x^\top a\right) = \nabla_x \left(a^\top x\right) = a \tag{16}$$

$ax = a$

máme

$$\nabla_\Theta \left(y^\top y\right) = 0$$
$$\nabla_\Theta \left(y^\top H\Theta\right) = \left(y^\top H\right)^\top = H^\top y$$
$$\nabla_\Theta \left(\Theta^\top H^\top y\right) = H^\top y$$
$$\nabla_\Theta \left(\Theta^\top H^\top H\Theta\right) = \left(H^\top H\Theta + \left(\Theta^\top H^\top H\right)^\top\right) = H^\top H\Theta + H^\top H\Theta$$

$(y^\top H)^\top = H^\top y^{\top\top} = H^\top y$

teda

$$\begin{aligned} \nabla_\Theta (J) &= \frac{1}{2}\left(-H^\top y - H^\top y + H^\top H\Theta + H^\top H\Theta\right) \\ &= \frac{1}{2}\left(-2H^\top y + 2H^\top H\Theta\right) \\ &= -H^\top y + H^\top H\Theta \end{aligned} \tag{17}$$

V extréme je hodnota $\nabla_\Theta (J)$ nulová, teda

$$\begin{aligned} 0 &= -H^\top y + H^\top H\Theta \\ H^\top H\Theta &= H^\top y \end{aligned} \tag{18}$$

a nakoniec

$$\Theta = \left(H^\top H\right)^{-1} H^\top y \tag{19}$$

Rovnica (19) sa nazýva Gaussov vzorec.

Keďže $\nabla_\Theta \nabla_\Theta (J) = \left(H^\top H\right) = H^\top H$ a $H^\top H = R$ je kladne definitná, pretože H je nenulová a teda $H^\top H$ je symetrická a kladne definitná, potom nájdený extrém je minimum. Matica R sa nazýva informačná matica a $P = R^{-1}$ sa nazýva disperzná matica (alebo aj Kovariančná matica) s $(n_a + n_b)$ riadkami a $(n_a + n_b)$ stĺpcami.

Dosadením do Gaussovho vzorca získame Off–line odhad parametrov modelu.

2.2 Rekurzívna metóda najmenších štvorcov

Predpokladajme, že poznáme odhad parametrov modelu v predchádzajúcom kroku $(k-1)$ a chceme získať odhad parametrov modelu v aktuálnom kroku k. Ak poznáme odhad parametrov v kroku $(k-1)$, je zrejmé, že poznáme aj maticu $P(k-1)$.

Pre lepšiu názornosť nech situácia je takáto: Aktuálny krok je $k = 2$. V kroku $(k-1)$, teda v kroku $k = 1$ bola matica $P(k-1)$. Ak prejdeme z kroku $(k-1)$ do kroku k, znemená to pridať nový riadok matice H. Novým riadkom je vektor $h^\top(k)$. Pre krok k môžme písať Gausov vzorec:

$$\Theta(k) = \left(\begin{bmatrix} H^\top(k-1) & h(k) \end{bmatrix} \begin{bmatrix} H(k-1) \\ h^\top(k) \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} H^\top(k-1) & h(k) \end{bmatrix} y(k) \tag{20}$$

Odkiaľ matica $P(k)$ je

$$\begin{aligned} P(k) &= \left(\begin{bmatrix} H^\top(k-1) & h(k) \end{bmatrix} \begin{bmatrix} H(k-1) \\ h^\top(k) \end{bmatrix} \right)^{-1} \\ &= \left(H^\top(k-1)H(k-1) + h(k)h^\top(k) \right)^{-1} \\ &= \left(P(k-1)^{-1} + h(k)h^\top(k) \right)^{-1} \end{aligned} \tag{21}$$

V rovnici (21) sa vyskytuje inverzia matice, ktorej rozmer závisí od počtu parametrov modelu. Tento počet môže byť veľký, a inverzia takto veľkej matice môže byť nerealizovateľná. Použitie Woodburryho lemmy o inverzií matíc rieši tento problém. Platí (v tomto texte nebudeme uvádzať dôkaz Woodburryho lemmy)

$$(A + BD)^{-1} = A^{-1} - A^{-1}B\left(DA^{-1}B + 1\right)^{-1} DA^{-1} \tag{22}$$

Použitím (22) prejde (21) do tvaru

$$\begin{aligned} P(k) &= P(k-1) - P(k-1)h(k)\left(1 + h^\top(k)P(k-1)h(k)\right)^{-1} \cdot h^\top(k)P(k-1) \\ &= P(k-1) - Y(k)h^\top(k)P(k-1) \end{aligned} \tag{23}$$

kde

$$Y(k) = \frac{P(k-1)h(k)}{1 + h^\top(k)P(k-1)h(k)}$$

je tzv. „zosilnenie“. Rovnica (23) je rekurzívnym vzťahom pre výpočet novej matice P z predchádzajúcej matice P.

Odhad parametrov modelu v kroku k je

$$\Theta(k) = P(k)H^\top(k)y(k) = P(k)\sum_{i=1}^{k} h(i)y_i = P(k)\left(\sum_{i=1}^{k-1} h(i)y_i + h(k)y(k)\right) \tag{24}$$

Z (21) je zrejmé že platí

$$P^{-1}(k) = P(k-1) + h(k)h(k)^\top \tag{25a}$$

$$P^{-1}(k-1) = P^{-1}(k) - h(k)h^\top(k) \tag{25b}$$

Potom

$$\begin{aligned} \sum_{i=1}^{k-1} h^\top(i)y_i &= P^{-1}(k-1)\Theta(k-1) = \left(P^{-1}(k) - h(k)h^\top(k)\right)\Theta(k-1) \\ &= P^{-1}(k)\Theta(k-1) - h(k)h^\top(k)\Theta(k-1) \end{aligned} \tag{26}$$

čo umožní písať odhad parametrov modelu v kroku k v tvare

$$\begin{aligned} \Theta(k) &= P(k)\left(P^{-1}(k)\Theta(k-1) - h(k)h^\top(k)\Theta(k-1) + h(k)y(k)\right) \\ &= P(k)P^{-1}(k)\Theta(k-1) - P(k)h(k)h^\top(k)\Theta(k-1) + P(k)h(k)y(k) \\ &= I\Theta(k-1) + P(k)h(k)\left(y(k) - h^\top(k)\Theta(k-1)\right) \\ &= \Theta(k-1) + P(k)h(k)e(k) \end{aligned} \tag{27}$$

$h = \begin{bmatrix} y(k-1) \\ y(k-2) \\ \vdots \end{bmatrix}$

Tabuľka 1: Algoritmus rekurzívnej MNŠ

1.	Odchýlka (rezíduum)	$e(k) = y(k) - \boxed{h^{\mathsf{T}}(k)\Theta(k-1)}$
2.	Zosilnenie	$Y(k) = \dfrac{P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}$
3.	Kovariančná matica	$P(k) = \boxed{P(k-1)} - Y(k)h^{\mathsf{T}}(k)P(k-1)$
4.	Nový odhad parametrov	$\Theta(k) = \Theta(k-1) + Y(k)e(k)$

$\hat{y} = h(k)\Theta(k)$

Rovnicu (27) môžeme priviesť aj do iného tvaru. Platí:

$$
\begin{aligned}
\Theta(k) &= \Theta(k-1) + P(k)h(k)e(k) \\
&= \Theta(k-1) + \big(P(k-1) - Y(k)h^{\mathsf{T}}(k)P(k-1)\big)\, h(k)e(k) \\
&= \Theta(k-1) + \left(P(k-1)h(k) - \frac{P(k-1)h(k)h^{\mathsf{T}}(k)P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right) e(k) \\
&= \Theta(k-1) \\
&\quad + \left(\frac{P(k-1)h(k) + P(k-1)h(k)h^{\mathsf{T}}(k)P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right. \\
&\quad \left. - \frac{P(k-1)h(k)h^{\mathsf{T}}(k)P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right) e(k) \qquad (28) \\
&= \Theta(k-1) + \left(\frac{P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right) e(k) \\
&\quad + \left(\frac{P(k-1)h(k)h^{\mathsf{T}}(k)P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right) e(k) \\
&\quad - \left(\frac{P(k-1)h(k)h^{\mathsf{T}}(k)P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right) e(k) \\
&= \Theta(k-1) + \left(\frac{P(k-1)h(k)}{1 + h^{\mathsf{T}}(k)P(k-1)h(k)}\right) e(k)
\end{aligned}
$$

Potom

$$\Theta(k) = \Theta(k-1) + Y(k)e(k) \qquad (29)$$

Rovnica (29) je rekurzívnym vzťahom pre výpočet nových parametrov modelu z predchádzajúcich hodnôt parametrov modelu.

2.2.1 Súhrn

Algoritmus rekurzívnej metódy najmenších štvorcov je zhrnutý v Tabuľke 1.

2.2.2 Štart algoritmu

Disperzná matica má začiatočný tvar

$$P(0) = \alpha I \qquad (30)$$

kde I je jednotková matica rovnakého rozmeru ako P a α je reálne číslo napríklad z intervalu $\langle 10,\, 10^6 \rangle$.

Začiatočné hodnoty parametrov modelu môžu byť napríklad nulové, t.j. vektor Θ je nulový vektor príslušnej dĺžky:

$$\Theta(0) = 0 \qquad (31)$$

Nie je to však pravidlo a začiatočné hodnoty parametrov modelu môžu byť v princípe akékoľvek. Tieto hodnoty sa spravidla využívajú v ďalších výpočtoch a tak napríklad môže vzniknúť požiadavka, že niektorý z parametrov nemôže byť nulový (napr. pre delenie nulou) a podobne. Tiež môže existovať približný, hrubý odhad týchto hľadaných (identifikovaných) parametrov a tento je potom často výhodné použiť ako začiatočný.

Tabuľka 2: Algoritmus rekurzívnej MNŠ s exponenciálnym zabúdaním

1.	Odchýlka (rezíduum)	$e(k) = y(k) - h^{\mathsf{T}}(k)\Theta(k-1)$
2.	Zosilnenie	$Y(k) = \dfrac{P(k-1)h(k)}{\lambda + h^{\mathsf{T}}(k)P(k-1)h(k)}$
3.	Kovariančná matica	$P(k) = \dfrac{1}{\lambda}\left(P(k-1) - Y(k)h^{\mathsf{T}}(k)P(k-1)\right)$
4.	Nový odhad parametrov	$\Theta(k) = \Theta(k-1) + Y(k)e(k)$

2.2.3 Modifikácia algoritmu - zabúdanie

V účelovej funkcii (14) sa nepredpokladá žiadne váhovanie jednotlivých prvkov vektora odchýlok e. Všetky prvky majú rovnakú váhu. Z pohľadu rekurzívneho algoritmu to znamená, že najstaršie odchýlky majú rovnakú váhu ako najnovšie. Často však môže byť veľmi výhodné ak by novšie vzorky, teda novšie zistené odchýlky mali vyššiu váhu ako staršie. Inými slovami výhodné by bolo zabúdať na staršie odchýlky a uvažovať len tie novšie.

V účelovej funkcii (14) je možné uvažovať váhovaciu maticu W nasledovne:

$$J(\Theta) = \frac{1}{2}e^{\mathsf{T}}We \tag{32}$$

kde matica

$$W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & w_N \end{bmatrix} \tag{33}$$

umožní realizovať váhovanie jednotlivých štvorcov odchýlok.

Ak zvolíme váhovacie koeficienty ako $w_i = \lambda^{N-i}$ potom sa takého váhovanie nazýva exponenciálne zabúdanie. Číslo λ sa volí z intervalu $(0, 1)$, pričom typické hodnoty sú $\lambda = 0,99$ či $\lambda = 0,95$. Potom je zrejmé, že najnovšia vzorka, najnovší štvorec odchýlky, má váhu (je prenásobený číslom) $w_N = \lambda^{N-N} = 1$. Všetky ostatné váhovacie koeficienty majú nižšiu hodnotu (hodnota exponenciálne klesá ako sa zmenšuje poradové číslo i).

Ak aplikujeme ten istý postup pre získanie rekurzívneho algoritmu MNŠ ako v prípade bez exponenciálneho zabúdania, tak verzia so zabúdaním vedie na algoritmus sumarizovaný v tabuľke 2.

3 Cvičenie druhé

1. Zrealizujme priebežnú identifikáciu parametrov ARX modelu tak ako to predpokladá konkrétny príklad v časti 1. Vyskúšajte verziu bez exponenciálneho zabúdania a s exponenciálnym zabúdaním.

3.1 Ukážka simulačnej schémy pre simuláciu daného riadeného systému

Cieľom nasledujúceho je zostaviť (univerzálnu) simulačnú schému, do ktorej je možné následne doplniť v podstate akýkoľvek riadiaci systém.

Simulačná schéma v tomto prípade realizuje len simuláciu samotného riadeného systému. Vstupný signál riadeného systému je tu zvolený (daný vopred), nie je generovaný riadiacim systémom.

Riadený systém nech predstavuje prenosová funkcia v tvare

$$G(s) = \frac{0,15}{s^2 + 0,3s + 0,2} \tag{34}$$

Ide o prenosovú funkciu druhého rádu, ktorá má v čitateli polynóm nultého stupňa, teda nemá nuly.

Parametre riadeného systému vo forme vstupného vektora matice dynamiky potom sú:

Výpis kódu 1: Súbor ar03_pr01.py

```
A = np.array([[0, 1], [-0.2, -0.3]])
b = np.array([[0], [0.15]])
```

Funkcia, ktorá realizuje diferenciálne rovnice riadeného systému, nech je nasledovná:

Výpis kódu 2: Súbor ar03_pr01.py

```
def fcn_difRovnice(x, t, u):

    dotx = np.dot(A,x) + np.dot(b,u)

    return dotx
```

$\dot{x} = Ax + bu$

Simulačnú schému nech realizuje nasledujúca funkcia:

Výpis kódu 3: Súbor ar03_pr01.py

```
def fcn_simSch_01_zaklad(t_start, T_s, finalIndex, sig_u_ext):

    #-----------------------------------------
    t_log = np.zeros([finalIndex, 1])
    t_log[0,:] = t_start

    #-----------------------------------------
    x_0 = np.array([0, 0])

    x_log = np.zeros([finalIndex, len(x_0)])
    x_log[0,:] = x_0

    #-----------------------------------------


    #-----------------------------------------
    timespan = np.zeros(2)
    for idx in range(1, int(finalIndex)):

        timespan[0] = t_log[idx-1,:]
        timespan[1] = t_log[idx-1,:] + T_s

        u = sig_u_ext[idx-1,:]

        odeOut = odeint(fcn_difRovnice,
                        x_log[idx-1,:],
                        timespan,
                        args=(u,)
                        )

        x_log[idx,:] = odeOut[-1,:]
        t_log[idx,:] = timespan[-1]

    return [t_log, x_log]
```

Nastavenia potrebné pre samotnú simuláciu a vygenerovanie signálov, ktoré sa používajú pri simulácii (ktoré sú dopredu známe - dané):

Výpis kódu 4: Súbor ar03_pr01.py

```
# Nastavenia simulacie

sim_t_start = 0
sim_t_final = 200
sim_T_s = 0.1
sim_finalIndex = int(((sim_t_final - sim_t_start)/sim_T_s) + 1)
```

Pre simuláciu je potrebné vytvoriť vstupný signál $u(t)$ pre riadený systém. Nech je nasledovný:

Výpis kódu 5: Súbor ar03_pr01.py

```
tab_delt_u = np.array([
                      [0, 0],
                      [1, 1],
```

```
                              [50, 0],
                              [100, -1],
                              [150, 0],
                              ])


sig_delt_u = np.zeros([sim_finalIndex, 1])
for idx in range(sig_delt_u.shape[0]):
    lastValue = tab_delt_u[:,1][tab_delt_u[:,0]<=idx*sim_T_s ][-1]
    sig_delt_u[idx] = lastValue


sig_u_ext = sig_delt_u
```

Vstupný signál $u(t)$ je zobrazený na obr. 1.

Spustenie simulácie:

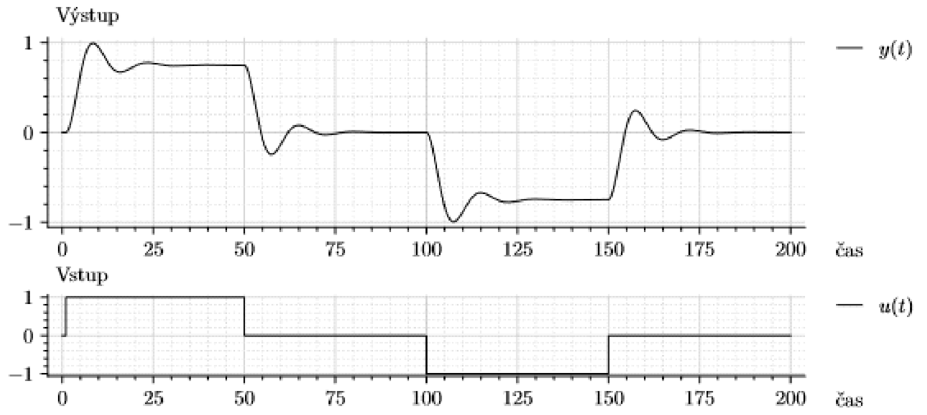
Výpis kódu 6: Súbor `ar03_pr01.py`

```
# Spustenie simulacie

t_log, x_log = fcn_simSch_01_zaklad(
                        sim_t_start,
                        sim_T_s,
                        sim_finalIndex,
                        sig_u_ext,
                        )
```

Nakreslenie obrázka (pre prehľadnosť je kód v samostatnom súbore):

Výpis kódu 7: Súbor `ar03_pr01.py`

```
# Obrazok

figName = 'figsc_ar03_fig01'
figNameNum = 1

exec(open('./figjobs/' + figName + '.py', encoding='utf-8').read())
```



Obr. 1

3.2 Doplnenie algoritmu RMNŠ do simulačnej schémy

V predchádzajúcom bola vytvorená simulačná schéma pre simuláciu uvažovaného riadeného systému. Tu je cieľom doplniť do simulačnej schémy algoritmus RMNŠ.

Pre úplnosť, ARX (AutoRegressive eXogenous) model, ktorý je identifikovaný (klasickou RMNŠ), je vo všeobecnosti nasledovný:

$$A(z^{-1})y(k) = B(z^{-1})u(k) + \xi(k) \tag{35}$$

kde

$$
\begin{aligned}
B(z^{-1}) &= b_1 z^{-1} + \ldots + b_{n_b} z^{-n_b} \\
A(z^{-1}) &= 1 + a_1 z^{-1} + \ldots + a_{n_a} z^{-n_a}
\end{aligned}
\tag{36}
$$

a $\xi(k)$ predstavuje poruchy a šum. V ďalšom budeme uvažovať $\xi(k) = 0$. ARX model vyjadrený v tvare diferenčnej rovnice:

$$
y(k) = -a_1 y(k-1) - \ldots - a_{n_a} y(k-n_a) + b_1 u(k-1) + \ldots + b_{n_b} u(k-n_b) \tag{37}
$$

Nech modelom riadeného systému je diferenčná rovnica v uvedenom tvare, kde hodnoty $n_a = 2$ a $n_b = 2$. Potom táto diferenčná rovnica má tvar

$$
y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_1 u(k-1) + b_2 u(k-2) \tag{38}
$$

V maticovom zápise:

$$
y(k) = h^\mathsf{T} \Theta \tag{39}
$$

kde $h^\mathsf{T} = \begin{bmatrix} -y(k-1) & -y(k-2) & u(k-1) & u(k-2) \end{bmatrix}$ a $\Theta = \begin{bmatrix} a_1 & a_2 & b_1 & b_2 \end{bmatrix}^\mathsf{T}$. Neznámymi parametrami modelu teda sú koeficienty a_1, a_2, b_1 a b_2.

Takpovediac diferenciálne rovnice riadeného systému ostavajú samozrejme rovnaké ako sme ich realizovali v predchádzajúcom, teda aj tu („na začiatku skriptu") platí výpis kódu 1 a 2.

Simulačnú schému nech realizuje nasledujúca funkcia:

Výpis kódu 8: Súbor ar03_pr02.py

```
def fcn_simSch_02_lenRMNS(t_start, T_s, finalIndex, sig_u_ext):

    #-------------------------------------------
    t_log = np.zeros([finalIndex, 1])
    t_log[0,:] = t_start

    #-------------------------------------------
    x_0 = np.array([0, 0])

    x_log = np.zeros([finalIndex, len(x_0)])
    x_log[0,:] = x_0

    #-------------------------------------------

    u_log = np.zeros([finalIndex, 1])

    #-------------------------------------------

    RMNS_theta_0 = np.array([[ 0.001],
                             [ 0.001],
                             [ 0.001],
                             [ 0.001]])

    RMNS_theta_log = np.zeros([finalIndex, len(RMNS_theta_0)])
    RMNS_theta_log[0,:] = RMNS_theta_0.reshape(1,-1)

    RMNS_P_0 = np.identity(4) * 10**6

    RMNS_P_log = np.zeros([finalIndex, RMNS_P_0.size])
    RMNS_P_log[0,:] = RMNS_P_0.reshape(1,-1)

    #----

    RMNS_y_predict_log = np.zeros([finalIndex, 1])

    #-------------------------------------------
    timespan = np.zeros(2)
    for idx in range(1, int(finalIndex)):

        timespan[0] = t_log[idx-1,:]
        timespan[1] = t_log[idx-1,:] + T_s

        odeOut = odeint(fcn_difRovnice,
                        x_log[idx-1,:],
```

t_log

$\hat{y}$-log

k=1

```
                        timespan,
                        args=(u_log[idx-1,:],)
                        )

        x_log[idx,:] = odeOut[-1,:]
        t_log[idx,:] = timespan[-1]

        #--------------------
        # ALGORITMUS RMNS
        y_k = x_log[idx,0]

        h_k = np.array([[-x_log[idx-1,0]],
                        [-x_log[idx-2,0]], # pozor na to idx-2 !!!
                        [u_log[idx-1,0]],
                        [u_log[idx-2,0]], # pozor na to idx-2 !!!
                        ])

        theta_km1 = RMNS_theta_log[idx-1,:].reshape(4,-1)
        P_km1 = RMNS_P_log[idx-1,:].reshape(4,4)

        #----------
        lambdaKoef = 1.0

        e_k = y_k - np.matmul(h_k.T, theta_km1)
        Y_k =  np.matmul(P_km1, h_k) / (lambdaKoef + np.matmul(np.matmul(h_k.T, P_km1), h_k))
        P_k = (1/lambdaKoef) * (P_km1 - np.matmul(np.matmul(Y_k, h_k.T), P_km1))
        theta_k = theta_km1 + Y_k * e_k

        #----

        RMNS_theta_log[idx,:] = theta_k.reshape(1,-1)
        RMNS_P_log[idx,:] = P_k.reshape(1,-1)

        #----------

        RMNS_y_predict_log[idx,:] = np.matmul(h_k.T, theta_km1)

        #--------------------
        u_log[idx,:] = sig_u_ext[idx-1,:]

    return [t_log, x_log, RMNS_y_predict_log, RMNS_theta_log]
```

V uvedenej simulačnej schéme je implementovaný RMNŠ algoritmus, ktorého výstupom je vektor parametrov `theta_k` a následne je tiež vypočítaná (v každom cykle) jednokroková predikcia výstupného signálu zapisovaná do vektora `RMNS_y_predict_log`.

Všimnime si tiež napríklad, že faktor zabúdania λ (premenná `lambdaKoef`) je nastavený na hodnotu $\lambda = 1$, teda algoritmus nevyužíva zabúdanie.

Nastavenia potrebné pre samotnú simuláciu a vygenerovanie signálov, ktoré sa používajú pri simulácii (ktoré sú dopredu známe - dané):

Výpis kódu 9: Súbor `ar03_pr02.py`

```
# Nastavenia simulacie

sim_t_start = 0
sim_t_final = 55
sim_T_s = 0.25
sim_finalIndex = int(((sim_t_final - sim_t_start)/sim_T_s) + 1)
```

Pre simuláciu je potrebné vytvoriť vstupný signál $u(t)$ pre riadený systém. Nech je nasledovný:

Výpis kódu 10: Súbor `ar03_pr02.py`

```
# Preddefinovane signaly

period_time = 40
period_tab = np.array([
                      [0, 1],
                      [10, 0],
                      [20, -1],
                      [30, 0],
                      ])

sig_vysl = np.zeros([sim_finalIndex, 1])

```

```
for period in range(int(sim_t_final/period_time) + 1):


    for idx in range( int((period*period_time)/sim_T_s), int((period*
    period_time + period_time)/sim_T_s)):

        lastValue = period_tab[:,1][(period_tab[:,0] + (period*
    period_time))<=idx*sim_T_s ][-1]
        try:
            sig_vysl[idx] = lastValue
        except:
            break

sig_u_ext = sig_vysl
```

Spustenie simulácie:

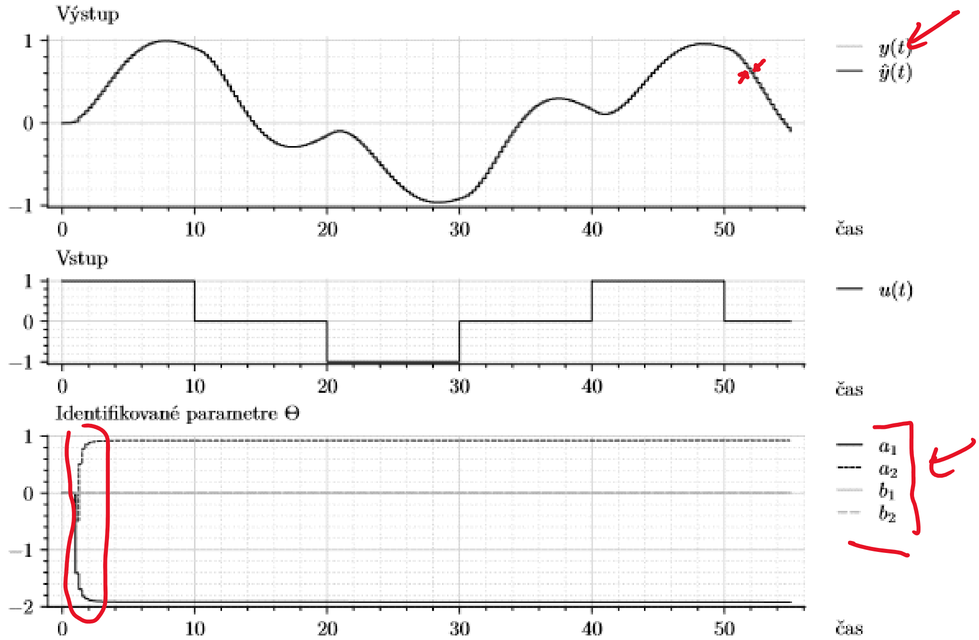
Výpis kódu 11: Súbor `ar03_pr02.py`

```
# Spustenie simulacie

t_log, x_log, RMNS_y_predict_log, RMNS_theta_log =
    fcn_simSch_02_lenRMNS(
    sim_t_start,
    sim_T_s,
    sim_finalIndex,
    sig_u_ext,
    )
```

Nakreslenie obrázku (pre prehľadnosť je kód v samostatnom súbore):

Výpis kódu 12: Súbor `ar03_pr02.py`

```
# Obrazok

figName = 'figsc_ar03_fig02'
figNameNum = 0

exec(open('./figjobs/' + figName + '.py', encoding='utf-8').read())

```



Obr. 2

Pre zaujímavosť, priebežne identifikované parametre Θ sú zapisované do poľa `RMNS_theta_log`. Posledný riadok v tomto poli je:

Výpis kódu 13: Súbor ar03_pr02.py

```
print(RMNS_theta_log[-1,:])
```

```
[-1.91569536  0.92772642  0.00456786  0.00445568]
```

3.3 Algoritmus RMNŠ pri zašumených dátach

V predchádzajúcom bola vytvorená simulačná schéma pre simuláciu uvažovaného riadeného systému a bol do nej doplnený algoritmus RMNŠ.

Výstupná veličina samotného simulovaného riadeného systému je, pochopiteľne, bez šumu. Tu je cieľom preskúmať ako je RMNŠ schopný vysporiadať sa s prítomnosťou šumu v dátach výstupnej veličiny.

Aj tu platí, že diferenciálne rovnice riadeného systému ostávajú samozrejme rovnaké ako sme ich realizovali v predchádzajúcom, teda aj tu („na začiatku skriptu“) platí výpis kódu 1 a 2.

Simulačnú schému nech realizuje nasledujúca funkcia:

Výpis kódu 14: Súbor ar03_pr03.py

```
# Simulacna schema:

def fcn_simSch_02_lenRMNS_noise(t_start, T_s, finalIndex, sig_u_ext,
    lambdaKoef):

    #-------------------------------------------
    t_log = np.zeros([finalIndex, 1])
    t_log[0,:] = t_start

    #-------------------------------------------
    x_0 = np.array([0, 0])

    x_log = np.zeros([finalIndex, len(x_0)])
    x_log[0,:] = x_0


    y_log_noise = np.zeros([finalIndex, 1])
    y_log_noise[0,0] = x_log[0,0]

    #-------------------------------------------

    u_log = np.zeros([finalIndex, 1])

    #-------------------------------------------

    RMNS_theta_0 = np.array([[ 0.001],
                             [ 0.001],
                             [ 0.001],
                             [ 0.001]])


    RMNS_theta_log = np.zeros([finalIndex, len(RMNS_theta_0)])
    RMNS_theta_log[0,:] = RMNS_theta_0.reshape(1,-1)

    RMNS_P_0 = np.identity(4) * 10**2

    RMNS_P_log = np.zeros([finalIndex, RMNS_P_0.size])
    RMNS_P_log[0,:] = RMNS_P_0.reshape(1,-1)

    RMNS_y_predict_log = np.zeros([finalIndex, 1])

    #-------------------------------------------
    timespan = np.zeros(2)
    for idx in range(1, int(finalIndex)):

        timespan[0] = t_log[idx-1,:]
        timespan[1] = t_log[idx-1,:] + T_s

        odeOut = odeint(fcn_difRovnice,
                        x_log[idx-1,:],
                        timespan,
                        args=(u_log[idx-1,:],)
                        )

```

```
        x_log[idx,:] = odeOut[-1,:]
        t_log[idx,:] = timespan[-1]

        # Tu sa umelo pridava sum k vystupnej velicine riadeneho systemu

        y_log_noise[idx,0] = x_log[idx,0] + np.random.normal(0, 0.1, size=1)

        #--------------------
        # ALGORITMUS RMNS

        # Pri RMNS sa vyuziva zasumena vystupna velicina

        y_k = y_log_noise[idx,0]

        h_k = np.array([[-y_log_noise[idx-1,0]],
                        [-y_log_noise[idx-2,0]], # pozor na idx-2 !!!
                        [u_log[idx-1,0]],
                        [u_log[idx-2,0]], # pozor na idx-2 !!!
                        ])

        theta_km1 = RMNS_theta_log[idx-1,:].reshape(4,-1)
        P_km1 = RMNS_P_log[idx-1,:].reshape(4,4)

        #----------
        e_k = y_k - np.matmul(h_k.T, theta_km1)
        Y_k =  np.matmul(P_km1, h_k) / (lambdaKoef + np.matmul(np.matmul(h_k.T, P_km1), h_k))
        P_k = (1/lambdaKoef) * (P_km1 - np.matmul(np.matmul(Y_k, h_k.T), P_km1))
        theta_k = theta_km1 + Y_k * e_k

        #----------
        RMNS_theta_log[idx,:] = theta_k.reshape(1,-1)
        RMNS_P_log[idx,:] = P_k.reshape(1,-1)

        RMNS_y_predict_log[idx,:] = np.matmul(h_k.T, theta_km1)

        #--------------------
        u_log[idx,:] = sig_u_ext[idx-1,:]

    return [t_log, x_log, RMNS_y_predict_log, RMNS_theta_log, y_log_noise]
```

V uvedenej simulačnej schéme je implementovaný RMNŠ algoritmus, ktorého výstupom je vektor parametrov `theta_k` a následne je tiež vypočítaná (v každom cykle) jednokroková predikcia výstupného signálu zapisovaná do vektora `RMNS_y_predict_log`.

Nastavenia potrebné pre samotnú simuláciu a vygenerovanie signálov, ktoré sa používajú pri simulácii (ktoré sú dopredu známe - dané):

Výpis kódu 15: Súbor `ar03_pr03.py`

```
# Nastavenia simulacie

sim_t_start = 0
sim_t_final = 250
sim_T_s = 0.1
sim_finalIndex = int(((sim_t_final - sim_t_start)/sim_T_s) + 1)
```

Pre simuláciu je potrebné vytvoriť vstupný signál $u(t)$ pre riadený systém. Nech je nasledovný:

Výpis kódu 16: Súbor `ar03_pr03.py`

```
# Preddefinovane signaly

period_time = 200
period_tab = np.array([
                      [0, 1],
                      [80, 0],
                      [120, -1],
                      [180, 0],
                      ])

sig_vysl = np.zeros([sim_finalIndex, 1])

for period in range(int(sim_t_final/period_time) + 1):
```

```
    for idx in range( int((period*period_time)/sim_T_s), int((period*
period_time + period_time)/sim_T_s)):

        lastValue = period_tab[:,1][(period_tab[:,0] + (period*
period_time))<=idx*sim_T_s ][-1]
        try:
            sig_vysl[idx] = lastValue
        except:
            break

sig_u_ext = sig_vysl
```

3.3.1 Bez zabúdania

Ďalším nastavením, špeciálne dôležitým v tomto príklade je koeficient zabúdania λ:

Výpis kódu 17: Súbor ar03_pr03.py

```
sim_lambdaKoef = 1.0
```

Pamätajme teda, že faktor zabúdania λ (premenná lambdaKoef) je tu nastavený na hodnotu $\lambda = 1$, teda algoritmus nevyužíva zabúdanie.
Spustenie simulácie:

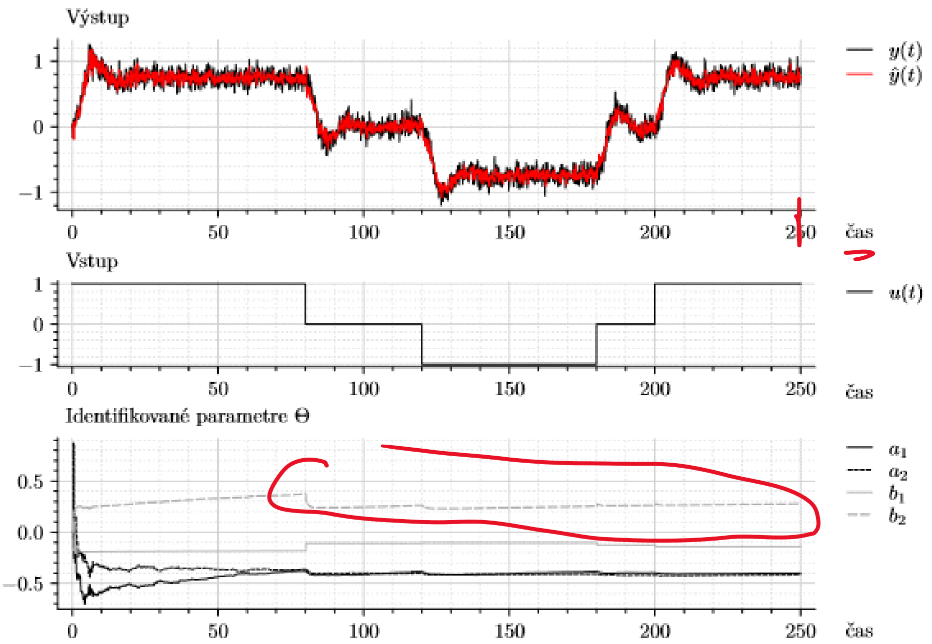
Výpis kódu 18: Súbor ar03_pr03.py

```
# Spustenie simulacie

t_log, x_log, RMNS_y_predict_log, RMNS_theta_log, y_log_noise =
    fcn_simSch_02_lenRMNS_noise(
    sim_t_start,
    sim_T_s,
    sim_finalIndex,
    sig_u_ext,
    sim_lambdaKoef
    )
```

Nakreslenie obrázka (pre prehľadnosť je kód v samostatnom súbore):

Výpis kódu 19: Súbor ar03_pr03.py

```
# Obrazok

figName = 'figsc_ar03_fig03'
figNameNum = 0

exec(open('./figjobs/' + figName + '.py', encoding='utf-8').read())
```



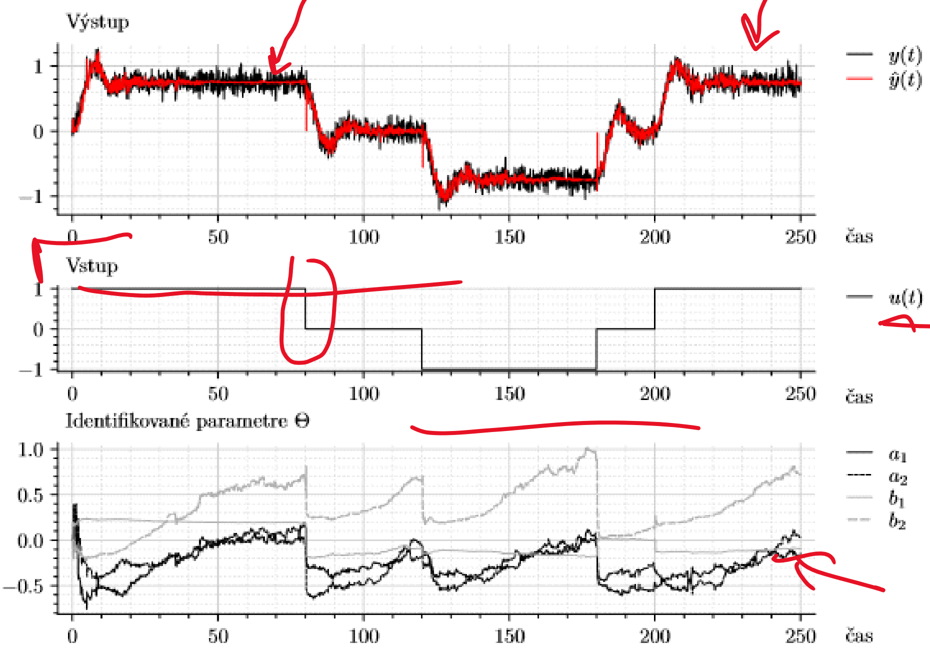
Obr. 3

3.3.2 So zabúdaním

Iná simulácia nech je s nasledovným koeficientom zabúdania:

Výpis kódu 20: Súbor ar03_pr03.py

```
sim_lambdaKoef = 0.987

# Spustenie simulacie

t_log, x_log, RMNS_y_predict_log, RMNS_theta_log, y_log_noise =
    fcn_simSch_02_lenRMNS_noise(
    sim_t_start,
    sim_T_s,
    sim_finalIndex,
    sig_u_ext,
    sim_lambdaKoef
    )

# Obrazok

figName = 'figsc_ar03_fig03'
figNameNum = 1

exec(open('./figjobs/' + figName + '.py', encoding='utf-8').read())
```

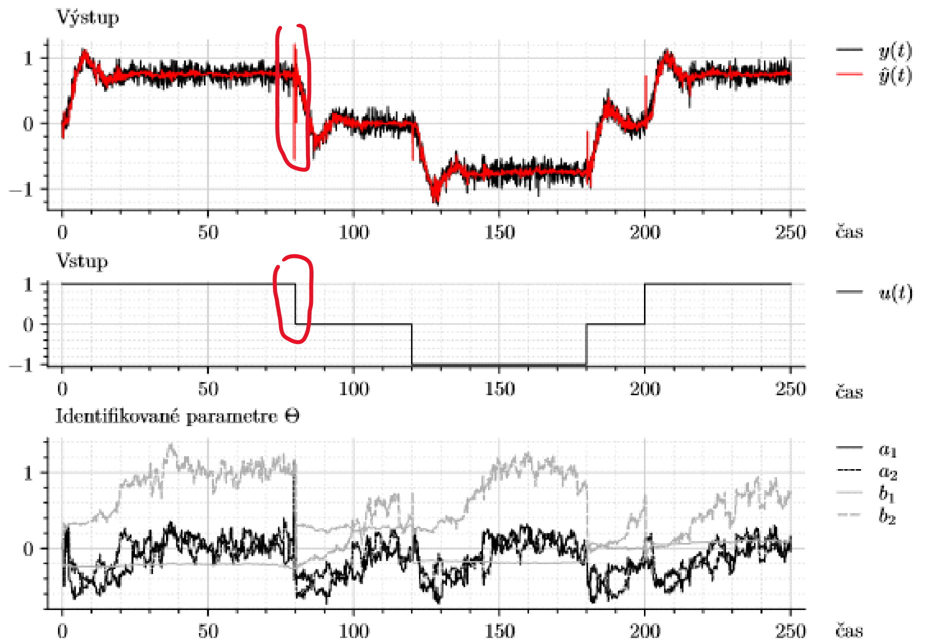


Obr. 4

Extrémnou voľbou koeficientu zabúdania pre tento prípad by bolo:

Výpis kódu 21: Súbor `ar03_pr03.py`

```
sim_lambdaKoef = 0.957

# Spustenie simulacie

t_log, x_log, RMNS_y_predict_log, RMNS_theta_log, y_log_noise =
    fcn_simSch_02_lenRMNS_noise(
    sim_t_start,
    sim_T_s,
    sim_finalIndex,
    sig_u_ext,
    sim_lambdaKoef
    )

# Obrazok

figName = 'figsc_ar03_fig03'
figNameNum = 2

exec(open('./figjobs/' + figName + '.py', encoding='utf-8').read())
```

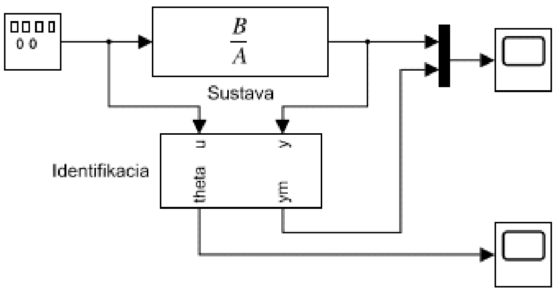


Obr. 5

3.4 Iná ukážka simulačnej schémy pre simuláciu RMNŠ

V tejto časti sa použije Simulink pre riešenie úloh ako v predchádzajúcom.

Majme sústavu, lepšie povedané dynamický systém (ktorý neskôr budeme riadiť), a k tomu istý blok, ktorého funkciou je realizovať priebežnú identifikáciu predmetného systému. Schematicky znázornené:



Obr. 6

Blok Identifikácia slúži na priebežnú identifikáciu a teda jeho hlavným výstupom sú parametre Θ a tiež sa uvažuje výstupná veličina modelu $\hat{y}$ (na obr. označená ako `ym`).

Pred spustením schémy (v Simulinku) je samozrejme potrebné inicializovať parametre riadeného systému (sústavy) a ako sa ukáže aj iné veci (v tomto prípade). Je to realizované v skripte:

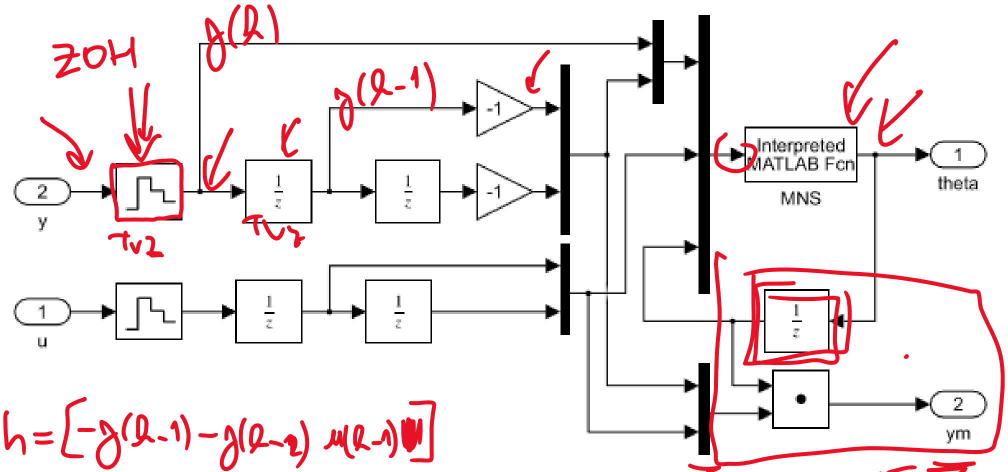
Výpis kódu 22: Súbor `ar03_spustima_RMNS.m`

```
clear all;
clc
```

z $(k-1)$ $y(k)$ k $\Theta(k)$ $u(k)$ K

```
global P

% Perioda vzorkovania
Tvz = 0.1;

% Identifikovana sustava
B = [0 0.15];
A = [1 0.3 0.2];

% Prepis do diskretneho tvaru (len tak pre zaujimavost...)
Gs = tf(B,A);
Gz = c2d(Gs,Tvz);
[Bz,Az] = tfdata(Gz,'v')

% Startovacia matica P
P = 10^6 * eye(4,4) ;
```

Samotný blok Identifikácia je realizovaný nasledovne:



$h = [-y(k-1) - y(k-2) \; u(k-1)]$

Obr. 7

$y_k = h^T \Theta$

Obsahuje vzorkovanie signálov (zero order hold) a oneskorovanie signálov (v zmysle z^{-1}). Tým je realizované získavanie tzv. signálneho vektora a podobne. Ďalej je súčasťou bloku funkcia, ktorá realizuje samotný algoritmus RMNŠ. Kód funkcie:

Výpis kódu 23: Súbor MNS.m

```
function odhadTheta = MNS(vst)
global P

P_n = P;

theta = vst(6:9);

h_n1 = [vst(2:5)];
y_n1 = vst(1);

e_n1 = y_n1 - h_n1' * theta;
Y_n1 = (P_n*h_n1)/(1+h_n1'*P_n*h_n1);
P_n1 = P_n - Y_n1*h_n1'*P_n;
odhadTheta = theta + Y_n1*e_n1;

P = P_n1;
```