Realizace rekurzivních metod identifikace v prostředí SCILAB&XCos

Bc. Jakub Hašek

Diplomová práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Ing. Jakub Hašek**
Osobní číslo:	**A16126**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Automatické řízení a informatika**
Forma studia:	**kombinovaná**
Téma práce:	**Realisace rekurzivních metod identifikace v prostředí SCILAB&XCos**
Téma anglicky:	**The Implementation of Recursive Identification Methods in the SCILAB&XCos Environment**

Zásady pro vypracování:

1. **Proveďte literární rešerši na zadané téma.**
2. **V teoretické části uveďte přehled základních lineárních dynamických modelů a rekurzivních identifikačních metod, které slouží pro odhad jejich parametrů.**
3. **V praktické části vytvořte knihovnu rekurzivních identifikačních metod v programovém prostředí SCILAB&XCOS. Vybrané rekurzivní metody identifikace s vhodnými faktory zapomínání realizujte jako samostatné uživatelské bloky.**
4. **Využití knihovny demonstrujte na vhodně zvolených příkladech.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **AFFOUF, M. Scilab by Example: for Beginners and Experienced Users. CreateSpace Independent Publishing Platform, 2012. ISBN 978-1479203444.**
2. **BOBÁL, Vladimír. Identifikace systémů. Zlín: Univerzita Tomáše Bati ve Zlíně, 2009. ISBN 978-80-7318-888-7.**
3. **LJUNG, Lennart. System identification: theory for the user. 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, c1999. ISBN 978-0136566953.**
4. **NOSKIEVIČ, Petr. Modelování a identifikace systémů. Ostrava: Montanex, 1999. ISBN 80-7225-030-2.**
5. **E. WELLSTEAD AND M. B. ZARROP. Self-tuning system: control and signal processing. Reprint. Chichester: Wiley, 1991. ISBN 0471930547.**
6. **RAMACHANDRAN, Hema a ACHUTHSANKAR S. NAIR. Scilab: (a free software to Matlab). New Delhi: S. Chand & Company, 2012. ISBN 9788121939706.**
7. **SHETH, Tejas. Scilab: A Practical Introduction to Programming and Problem Solving. CreateSpace Independent Publishing Platform, 2016. ISBN 978-1539027843.**
8. **SÖDERSTRÖM, Torsten a Petre STOICA. System identification. New York [u.a.]: Prentice-Hall, 1994. ISBN 0131276069. ROUX, Philippe. Scilab from Theory to Practice – I. Fundamentals. xxx: Editions D-Booker, 2016. ISBN 978-2822702935.**

Vedoucí diplomové práce: **Ing. Petr Navrátil, Ph.D.**
Ústav řízení procesů

Datum zadání diplomové práce: **15. prosince 2017**

Termín odevzdání diplomové práce: **25. května 2018**

Ve Zlíně dne 15. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor;
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15.5.2018 .....

podpis diplomanta

ABSTRAKT

Cílem této diplomové práce je implementace knihovny rekurzivních identifikačních algoritmů v programu SCILAB, konkrétně pro grafickou nadstavbu XCos. V teoretické části práce jsou shrnuty základní teoretické metody rekurzivních identifikačních algoritmů, které jsou následně v praktické části implementovány do jednotlivých bloků knihovny.

Diplomová práce si klade za cíl rozšířit komunitu kolem open source programu SCILAB a vytvořit alternativu pro sice oblíbený, ale komerčně pojatý software MATLAB.

Klíčová slova: SCILAB, Xcos, rekurzivní identifikační algoritmy, manuál, ARX model, ARMAX model, RLS metoda

ABSTRACT

The aim of this diploma thesis is the implementation of a recursive identification algorithm library in the SCILAB software, especially for graphic interface Xcos. The theoretical part describes basic theoretical methods of recursive identification algorithms, which are subsequently implemented in the practical blocks into individual blocks of the library.

This diploma thesis wants also support the open source SCILAB community and create an alternative to commercial software MATLAB.

Keywords: SCILAB, Xcos, recursive estimation algorithms, user's guide, ARX model, ARMAX model, RLS method

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Petru Navrátilovi Ph.D. za odborné vedení, za připomínky a čas věnovaný mé práci.

OBSAH

ABSTRAKT **5**

ÚVOD **9**

I TEORETICKÁ ČÁST **11**

1 VOLBA MODELU **12**

1.1 ARX model 14

1.2 ARMAX model 15

1.3 OE model 16

2 KRITERIA KVALITY IDENTIFIKACE **17**

2.1 Chyba vstupu 17

2.2 Chyba výstupu 18

2.3 Chyba rovnice 18

3 METODY IDENTIFIKACE **19**

3.1 Metoda nejmenších čtverců (LS) 19

3.2 Rekurzivní metoda nejmenších čtverců (RLS) 21

3.3 Rekurzivní metoda nejmenších čtverců (RLS) s faktorem zapomínání 24

3.3.1 Konstantní exponenciální zapomínání 24
3.3.2 Proměnlivé exponenciální zapomínání 25
3.3.3 Směrové zapomínání 25
3.3.4 Adaptivní směrové zapomínání 26
3.3.5 Maticové exponenciální zapomínání 26
3.3.6 Konstantní stopa 27

3.4 Rozšířená rekurzivní metoda nejmenších čtverců (RELS) 28

3.5 Rekurzivní metoda instrumentální proměnné (RIV) 30

3.6 Rozšířená rekurzivní metoda instrumentální proměnné (ERIV) 32

3.7 Rekurzivní metoda predikčních chyb (RPEM) 33

3.7.1 Metoda RPEM pro model ARMAX 33
3.7.2 Metoda RPEM pro model OE 36

II PRAKTICKÁ ČÁST **38**

4 SCILAB **39**

4.1 Xcos 41

4.2 Realizace nového bloku v Xcosu....43
4.3 Realizace knihovny rekurzivních identifikačních algoritmů v prostředí Scilab Xcos....49
4.4 Příklady použití bloku knihovny....52
ZÁVĚR....56
SEZNAM POUŽITÉ LITERATURY....58
SEZNAM OBRÁZKŮ....60

ÚVOD

Identifikace modelů dynamických systémů je důležitou součástí teorie automatického řízení, která umožňuje lepší prostudování a pochopení pozorovaného dynamického procesu a konečně je nezbytná pro návrh efektivního řízení. Metody identifikace systémů však nenachází uplatnění pouze striktně v teorii automatického řízení, ale také v oblasti zpracování signálů a v širším pojetí všude, kde dochází k průzkumu dynamických systémů jako například v chemii, biologii, ale třeba i ekonomii a statistické analýze.

Je však důležité si uvědomit, že vytvořený model je vždy určitým stupněm aproximace pozorovaného procesu, proto i metody identifikace využívají v obecné rovině různých přístupů. Metody identifikace můžeme dle těchto přístupů dělit na analytické a experimentální. Analytický přístup je však často, vzhledem ke složitosti systémů, velmi nevhodný. Naproti tomu experimentální přístup vychází z pozorování systému a jeho výstupů jako reakce na různé vstupy. Tyto vstupy mohou být časově determinované nebo častěji stochastické, které umožňují lépe identifikovat systémy ovlivněné poruchovými veličinami. V jednodušším případě naměříme ucelenou sadu dat sestávajících se vstupů a odpovídajících výstupů a na této konečné množině provedeme jednorázovou identifikaci. Reálné, často nestacionární procesy (s měnícím se výstupem při konstantním vstupu), si však vyžádali použití průběžným identifikačních algoritmů, které jsou schopné v průběhu identifikace měnit parametry modelovaného systému. Tyto metody prošly rozvojem zejména v souvislosti s adaptivními metodami řízení. Při těchto metodách je potom identifikovaný model systému v podstatě součástí regulační struktury.

Nejrozšířenějšími metodami identifikace jsou experimentální stochastické metody založené na regresní analýze, konkrétně metoda nejmenších čtverců (RLS) včetně jejích modifikací a rozšíření, metoda instrumentální proměnné (RIV) a další. K rozvoji těchto metod došlo i v souvislosti s překotným vývojem výpočetní techniky v posledních dekádách, která umožnila efektivní a rychlé řešení numerických problémů. Při těchto typech analýzy je však nutné znát dopředu přesně strukturu modelu s ohledem na uspořádání vstupů, výstupů a poruchových veličin. Nejčastěji používanými modely jsou modely ARX (AutoRegressive with eXogenous input), ARMAX (AutoRegressive Moving Average with eXogenous input) a OE (Output Error)

Cílem této diplomové práce je implementace vybraných metod identifikace v podobě knihovny pro volně šiřitelný program SCILAB a jeho grafického rozhraní XCos. SCILAB je volně šiřitelný program určený obecně pro numerické výpočty. Kromě využití v oblasti klasické matematiky nachází uplatnění v aplikovaných oborech jako zpracování signálů, statická analýza, analýza a simulace systémů a mnoho dalších. Hlavní výhoda programu spočívá v nekomerčním pojetí, kterým se vymezuje proti ostatním kvalitním matematickým softwarům, zejména pak programům Matlab, Wolfram Mathematica, GNU Octave a dalším.

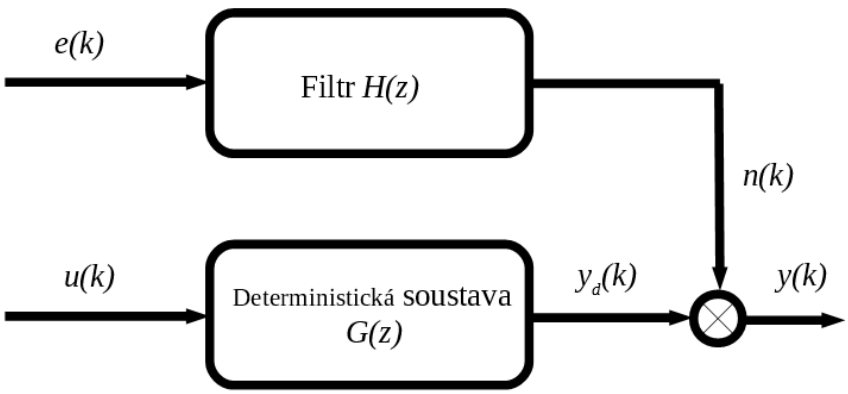
I. TEORETICKÁ ČÁST

1 VOLBA MODELU

Jednotlivé metody identifikace dynamických systému vychází z exaktních matematických odvození, která vychází z konkrétně zvolených modelů systému. V případě experimentální identifikace musíme tedy znát dopředu strukturu vlastního procesu, včetně poruchových veličin, abychom byli schopni zvolit konkrétní dynamický diskrétní model systému.

Z hlediska chování procesu můžeme modely rozdělit na deterministické a stochastické. Pokud budeme uvažovat deterministický lineární systém, pak jednotlivým vstupním signálům odpovídají v daném časovém okamžiku přesně určené hodnoty výstupního signálu. Takto popsaný model lze považovat za deterministický. Často lze ovšem výstupní signál přiřadit ke vstupnímu pouze s určitou pravděpodobností, což je způsobeno vnějšími rušivými vlivy nebo chybami v měření vstupů a výstupů. Takovéto modely lze chápat jako stochastické. Pokud tedy budeme uvažovat, že deterministický model obsahuje nulové vnější rušící vlivy, pak tuto skupinu lze považovat za podmnožinu stochastických modelů, proto modely reálných procesů popisujeme jako diskrétní lineární stochastické modely [1].

Obecný stochastický model procesu popisuje tedy vztah mezi vstupem, výstupem a poruchovou veličinou dle následujícího obrázku (Obr. 1)



Obr. 1: Blokové schéma modelu obecného stochastického procesu

Lze tedy říct, že obecný stochastický výstup se skládá z deterministické části, kterou tvoří deterministická soustava *G(z)* s obecným vstupem *u(k)* a stochastické části, kterou tvoří obecný náhodný signál *n(k)*. Tento signál můžeme však chápat jako signál bílého šumu *e(k)* filtrovaný přes filtr *H(z)*.

Pro blokové schema dle obrázku (Obr. 1) zapíšeme tedy rovnici:

$$y(k)=G(z)u(k)+H(z)e(k) \tag{1}$$

Diskrétní přenosové funkce *G(z)* a *H(z)* můžeme zapsat ve tvaru polynomů:

$$G(z)=\frac{B(z^{-1})}{A(z^{-1})F(z^{-1})}, H(z)=\frac{C(z^{-1})}{A(z^{-1})D(z^{-1})} \tag{2}$$

Výsledná rovnice stochastického obecného modelu má tedy tvar:

$$y(k)=\frac{B(z^{-1})}{A(z^{-1})F(z^{-1})}u(k)+\frac{C(z^{-1})}{A(z^{-1})D(z^{-1})}e(k) \tag{3}$$

Definici polynomů $A(z^{-1})$, $B(z^{-1})$, $C(z^{-1})$, $D(z^{-1})$ a $F(z^{-1})$ určuje jednotlivé typy modelů. Polynomy mají tvar:

$$A(z^{-1})=1+a_1 z^{-1}+...+a_{na} z^{-n_a} \tag{4}$$

$$B(z^{-1})=b_1 z^{-1}+...+b_{na} z^{-n_b} \tag{5}$$

$$C(z^{-1})=1+c_1 z^{-1}+...+c_{na} z^{-n_c} \tag{6}$$

$$D(z^{-1})=1+d_1 z^{-1}+...+d_{na} z^{-n_d} \tag{7}$$

$$F(z^{-1})=1+f_1 z^{-1}+...+f_{na} z^{-n_f} \tag{8}$$

Pro stupně polynomů platí $n_f>n_b$, $n_d>n_c$.

Mezi nejčastěji používané typy modelů patří ARX a ARMAX modely. Jejich společnou vlastností je stejná dynamická část *1/A(z)* v přenosových funkcích deterministické a stochastické složky *G(z)* a *H(z)*. Další významnou skupinou jsou modely založené na chybě výstupu jako například model typu OE, které předpokládají nezávislou deterministickou a stochastickou složku *G(z)* a *H(z)* [12].

Základní typy modelů:

- **ARX** - AutoRegressive with eXogenous input, podmínky: *C=D=F=1*

$$y(k)=\frac{B(z^{-1})}{A(z^{-1})}u(k)+\frac{1}{A(z^{-1})}e(k) \tag{9}$$

- **AR** - AutoRegressive, podmínky: *B=0, C=D=1*

$$y(k)=\frac{1}{A(z^{-1})D(z^{-1})}e(k) \tag{10}$$

- **ARMA** - AutoRegressive Moving Average, podmínky: *B=0, D=1*

$$y(k)=\frac{C(z^{-1})}{A(z^{-1})}e(k) \tag{11}$$

- **ARMAX** - AutoRegressive Moving Average with eXogenous input, podmínky: *D=F=1*

$$y(k)=\frac{B(z^{-1})}{A(z^{-1})}u(k)+\frac{C(z^{-1})}{A(z^{-1})}e(k) \tag{12}$$

- **OE** - Output Error, podmínky: *A=C=D=1*

$$y(k)=\frac{B(z^{-1})}{F(z^{-1})}u(k)+e(k) \tag{13}$$

- **BJ** – Box-Jenkins, podmínky: *A=1*

$$y(k)=\frac{B(z^{-1})}{F(z^{-1})}u(k)+\frac{C(z^{-1})}{D(z^{-1})}e(k) \tag{14}$$

- **ARIMAX** - AutoRegressive Moving Average with Integrator, podmínky: $D=1-z^{-1}$, *F=1*

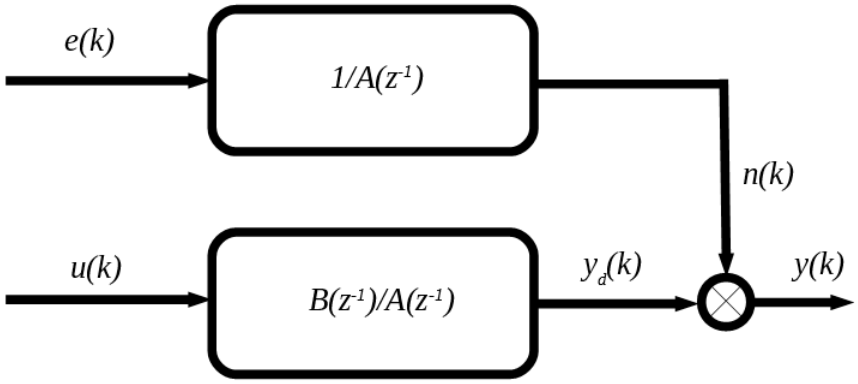
$$y(k)=\frac{B(z^{-1})}{A(z^{-1})}u(k)+\frac{C(z^{-1})}{A(z^{-1})(1-z^{-1})}e(k) \tag{15}$$

- **FIR** - Finite Impulse Response, podmínky: *A=F=C=D=1*

$$y(k)=B(z^{-1})u(k)+e(k) \tag{16}$$

1.1 ARX model

Model typu ARX (AutoRegressive with eXogenous input) předpokládá ovlivnění poruchovou veličinou typu bílého šumu. Pro polynomy *C,D* a *F* platí: $C(z^{-1})=D(z^{-1})=$ $=F(z^{-1})=1$. Blokové schema má potom tvar podle obrázku (Obr. 2).



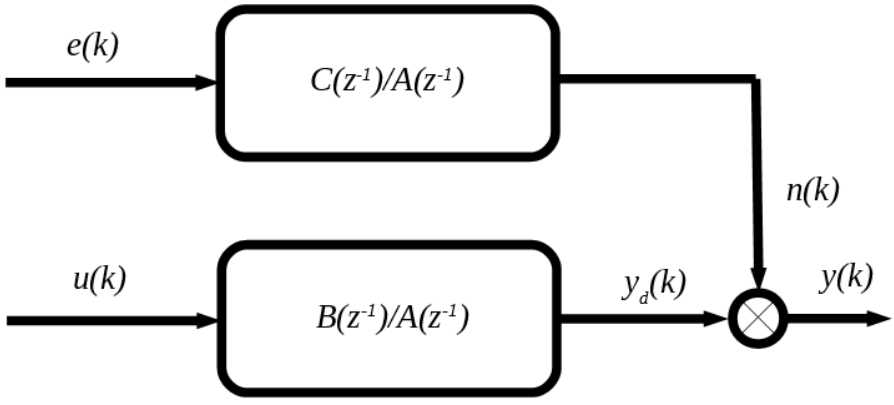
Obr. 2: Blokové schéma ARX modelu

Výslednou rovnici ARX modelu zapíšeme jako:

$$A(z^{-1})y(k)=B(z^{-1})u(k)+e(k) \tag{17}$$

1.2 ARMAX model

Model typu ARMAX (AutoRegressive Moving Average with eXogenous input) už předpokládá, že poruchová veličina typu bílého šumu je nějak modifikována filtrem *H(z)*. Pro polynomy *D* a *F* platí: $D(z^{-1})=F(z^{-1})=1$. Blokové schema má potom tvar podle obrázku (Obr. 3). Je dobré si uvědomit, že odhad parametrů polynomu *C(z)* bývá obtížný vzhledem k tomu, že vstupní poruchová veličina *e(k)* je z podstaty neměřitelná.



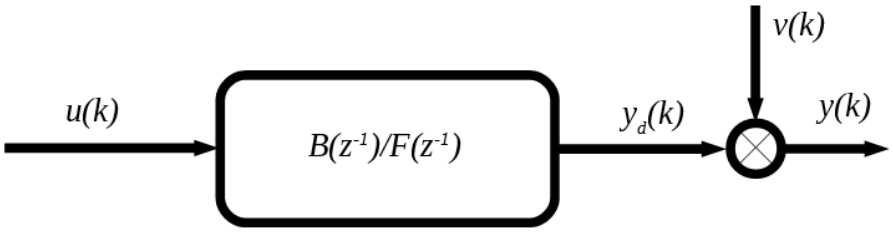
Obr. 3: Blokové schéma ARMAX modelu

Výslednou rovnici ARX modelu zapíšeme jako:

$$A(z^{-1})y(k)=B(z^{-1})u(k)+C(z^{-1})e(k) \tag{18}$$

1.3 OE model

Model typu OE (Output Error) patří do skupiny modelů založených na chybě výstupu. Stochastická složka je reprezentována přímo poruchovou veličinou dle obrázku (Obr. 4). Pro polynomy *A, C* a *D* platí: $A(z^{-1})=C(z^{-1})=D(z^{-1})=1$.



Obr. 4: Blokové schéma OE modelu

Výslednou rovnici OE modelu zapíšeme jako:

$$F(z^{-1})y(k)=B(z^{-1})u(k)+v(k) \tag{19}$$

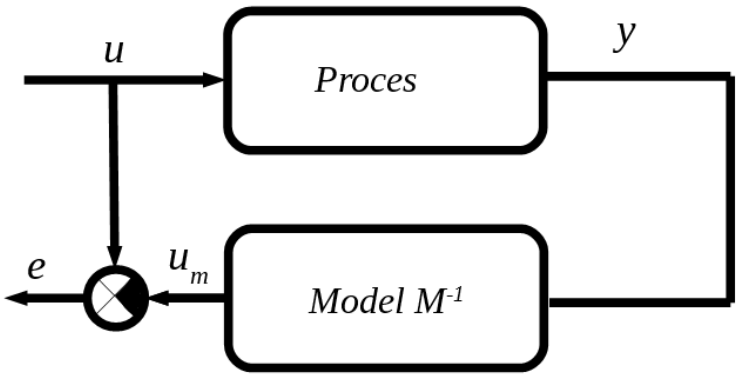
2 KRITERIA KVALITY IDENTIFIKACE

Většina metod identifikace systémů vychází z minimalizace nějaké obecné kriteriální ztrátové funkce *J*. Tato funkce může být různá, ale ukázalo se výhodné sestavit ji tak, aby se výsledný model co nejvíce blížil identifikovanému procesu při zachování odolnosti vůči rušivým vlivům.

Parametry kriteriální funkce musí být pozorovatelné a měřitelné, proto se v metodách rekurzivní identifikace používají kriteriální funkce založené na chybě vstupu, chybě výstupu nebo nejčastěji na chybě rovnice [9].

2.1 Chyba vstupu

Odchylka *e* je definována na straně vstupního signálu, který porovnáváme se vstupním signálem z modelu podle obrázku (Obr. 5). Použití této chyby předpokládá, že je model systému invertovatelný.



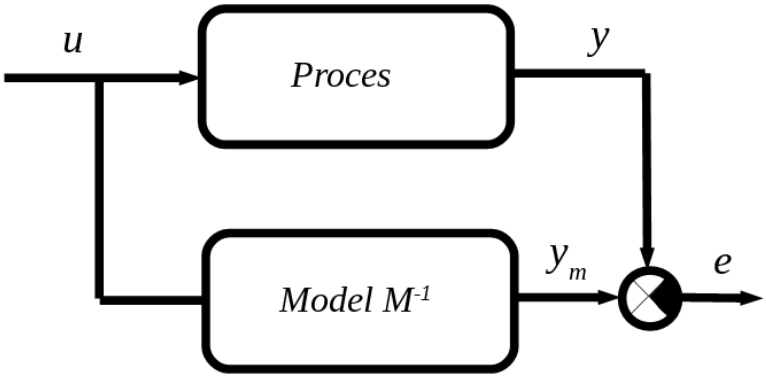
Obr. 5: Blokové schema-chyba vstupu

Pro chybu vstupu bude platit:

$$e = u - u_m = u - M^{-1} y \tag{20}$$

2.2 Chyba výstupu

Odchylka *e* je definována na straně výstupního signálu, který porovnáváme s výstupním signálem z modelu podle obrázku (Obr. 6).



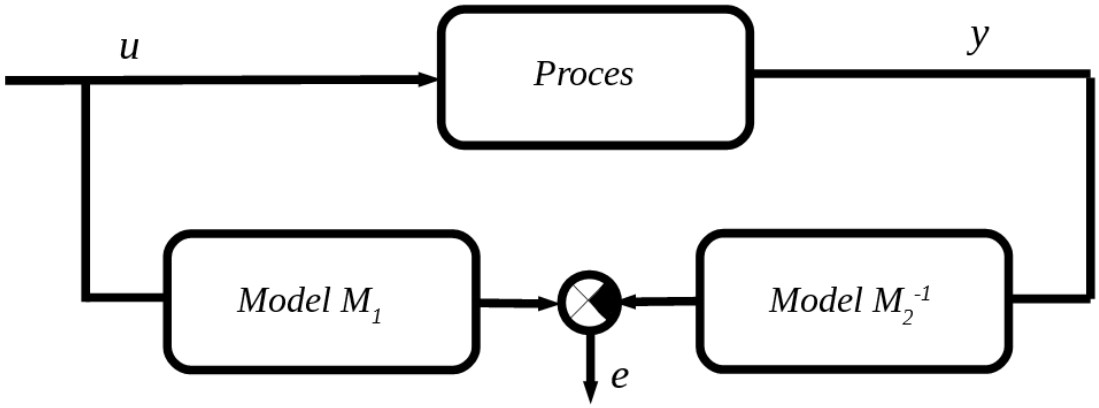
Obr. 6: Blokové schema-chyba výstupu

Pro chybu výstupu bude platit:

$$e = y - y_m = y - M\,u \tag{21}$$

2.3 Chyba rovnice

Chyba rovnice předpokládá rozpad modelu na invertovatelnou a přímou část. Je vhodná pro definování modelů ve tvaru přenosových funkcí, kde výsledný výstup je dán přímou modifikací vstupů a invertovanou modifikací výstupů podle obrázku (Obr. 7).



Obr. 7: Blokové schema-chyba rovnice

Pro chybu rovnice bude platit:

$$e = M_1^{-1}\,y - M_2 u \tag{22}$$

3 METODY IDENTIFIKACE

3.1 Metoda nejmenších čtverců (LS)

Metoda nejmenších čtverců (least square) patří mezi základní metody regresní analýzy. Mezi její největší výhody patří relativní jednoduchost a vysoká modifikovatelnost. Odvození této metody lze najít v řadě zdrojů [1], [6], [3]. Metoda nejmenších čtverců je založena na minimalizaci kriteriální funkce, kterou v případě identifikace systémů definujeme jako rozdíl naměřených a modelových hodnot:

$$e = y - y_m = y - \sum_{i=1}^{r} a_i f_i(k) \tag{23}$$

, kde a_i jsou hledané parametry modelu, které získáme minimalizací kvadratického kriteria chyby $e(k)$.

$$J = \sum_{k=1}^{N} e^2(k) = \sum_{k=1}^{N} \left[y(k) - a_i f_i(k) \right]^2 \tag{24}$$

Rozepsáním výrazu (23) dostaneme systém N rovnic (pro všechny pozorování) o r neznámých (pro hledané parametry):

$$\begin{array}{c} y(1) = a_1 f_1(1) + \ldots + a_r f_r(1) + e(1) \\ y(2) = a_1 f_1(2) + \ldots + a_r f_r(2) + e(2) \\ \vdots \\ y(N) = a_1 f_1(N) + \ldots + a_r f_r(N) + e(N) \end{array} \tag{25}$$

Pokud vektory ze soustavy rovnic (25) označíme jako:

$$\begin{array}{c} y^T = y(1) + y(2) + \ldots + y(N) \\ e^T = e(1) + e(2) + \ldots + e(N) \\ \theta^T = a_1 + a_2 + \ldots + a_r \vdots \end{array} \tag{26}$$

Funkční hodnoty ze soustavy rovnic (25) přepíšeme do matice jako:

$$F = \begin{bmatrix} f_1(1) + f_2(1) + \ldots + f_r(1) \\ f_1(2) + f_2(2) + \ldots + f_r(2) \\ \vdots \\ f_1(N) + f_2(N) + \ldots + f_r(N) \end{bmatrix} \tag{27}$$

Soustavu rovnic (25) můžeme vektorově zapsat jako:

$$y = F\theta + e \tag{28}$$

Rovnici pro chybu (23) potom přepíšeme do tvaru:

$$e = y - F\theta \tag{29}$$

Vektorově zapíšeme skalární kriterium jako:

$$J = e^T e = (y - F\theta)^T (y - F\theta) \rightarrow min \tag{30}$$

Pro minimalizaci tohoto kriteria bude platit, že parciální derivace podle jednotlivých proměnných musí být rovny nule.

$$\frac{\partial J}{\partial \theta} = 0 \tag{31}$$

Pro derivaci součinu vektorů můžeme zapsat:

$$\frac{\partial x_T y}{\partial a} = \frac{\partial x_T}{\partial a} y + \frac{\partial y_T}{\partial a} x \tag{32}$$

Aplikací věty o derivaci součinu vektorů na výraz (30) dostaneme:

$$\frac{\partial J}{\partial \hat{\theta}} = \frac{\partial (y - F\hat{\theta})^T}{\partial \hat{\theta}} (y - F\hat{\theta}) + \frac{\partial (y - F\hat{\theta})^T}{\partial \hat{\theta}} (y - F\hat{\theta}) = -2F^T (y - F\hat{\theta}) = 0 \tag{33}$$

Z výrazu (33) získáme:

$$F^T (y - F\hat{\theta}) = 0 \tag{34}$$

Z výrazu (34) vyjádříme vektor hledaných parametrů modelu a dostaneme výsledný tvar pro odhad parametrů modelu:

$$\hat{\theta} = (F^T F)^{-1} F^T y \tag{35}$$

, kde výraz $(F^T F)^{-1}$ nazýváme kovarianční maticí C. Z rovnice (35) vyplývá, že kovarianční matice nesmí být singulární, takže $det\ C \neq 0$.

3.2 Rekurzivní metoda nejmenších čtverců (RLS)

Rekurzivní identifikační algoritmy mají tu výhodu, že při pořízení nových naměřených údajů není nutné znovu opakovat celý jednorázový výpočet, ale pouze novou korekci hledaných parametrů z *k-1* kroku. Uplatňují se tedy při identifikaci proměnlivých systémů nebo v adaptivních metodách řízení [8]. Základní metodou rekurzivní identifikace je právě rekurzivní metoda nejmenších čtverců (recursive least square) aplikovaná na ARX model. Odvození vztahu pro výpočet odhadu vektoru parametrů Θ vychází ze vztahu (35). Pokud uvažujeme, že vztahy pro RLS platí pro první až *k-1* pozorování, pak měřením *k*-tého pozorování upravíme matici soustavy rovnic *F* (27) na:

$$F(k)=\begin{bmatrix} f_1(1)+f_2(1)+\ldots+f_r(1) \\ \vdots \\ f_1(k-1)+f_2(k-1)+\ldots+f_r(k-1) \\ f_1(k)+f_2(k)+\ldots+f_r(k) \end{bmatrix} \tag{36}$$

, kde poslední řádek označíme jako vektor $\Phi^T(k)$.

$$F(k)=\begin{bmatrix} F(k-1) \\ \phi^T(k) \end{bmatrix} \tag{37}$$

Rovnici pro výstup systému v *k*-tém kroku můžeme tedy vektorově zapsat jako:

$$y(k)=\theta^T\phi(k)+e(k) \tag{38}$$

Přepsáním základní rovnice pro výpočet metody nejmenších čtverců (LS) (35) pro *k*-tý krok dostaneme:

$$\hat{\theta}(k)=\left(F^T(k)F(k)\right)^{-1}F^T(k)\,y(k) \tag{39}$$

Pro kovarianční matici bude dosazením za *F(k)* z rovnice (36) platit:

$$C(k)=\left(F^T(k)F(k)\right)^{-1}=\left[\begin{bmatrix} F^T(k-1) & \phi(k) \end{bmatrix}\begin{bmatrix} F(k-1) \\ \phi^T(k) \end{bmatrix}\right]^{-1} \tag{40}$$

, po úpravě:

$$C(k)=\left[F^T(k-1)F(k-1)+\phi(k)\phi^T(k)\right]^{-1} \tag{41}$$

Dosazením kovarianční matice pro *k-1* krok do rovnice (41) dostaneme:

$$C(k)=\left[C^{-1}(k-1)+\phi(k)\phi^{T}(k)\right]^{-1} \tag{42}$$

Inverzi matice ve výrazu (42) můžeme provést pomocí věty o inverzi matic:

$$\left[A+BCB^{T}\right]^{-1}=A^{-1}-A^{-1}B\left[C^{-1}+B^{T}A^{-1}B\right]^{-1}B^{T}A^{-1} \tag{43}$$

Rovnici (43) aplikujeme na výraz (42) podle pravidel:

$$A=C^{-1}(k-1), B=\phi(k), C=1 \tag{44}$$

Výraz (42) přepíšeme jako:

$$C(k)=C(k-1)\left[I-\phi(k)\left(1+\phi^{T}(k)C(k-1)\phi(k)\right)^{-1}\phi^{T}(k)C(k-1)\right] \tag{45}$$

Tím dostáváme rovnici pro výpočet kovarianční matice v *k*-tém kroku při použití minulých hodnot. Z rovnice (39) pro výpočet odhadu parametrů zbývá určit součin členů $F^{T}(k).y(k)$ jako:

$$\hat{\theta}(k)=C(k)\left[\begin{bmatrix}F^{T}(k-1) & \phi(k)\end{bmatrix}\begin{bmatrix}y(k-1)\\ y(k)\end{bmatrix}\right]=C(k)\left[F^{T}(k-1)y(k-1)+\phi(k)y(k)\right] \tag{46}$$

Pokud zavedeme chybu predikce v následujícím tvaru:

$$\hat{e}(k)=y(k)-\phi^{T}(k)\theta(k-1) \tag{47}$$

Potom můžeme rovnici (46) přepsat dosazením za *y(k)* jako:

$$\hat{\theta}(k)=C(k)\left[F^{T}(k-1)y(k-1)+\phi(k)\phi(k)^{T}\hat{\theta}(k-1)+\phi(k)\hat{e}(k)\right] \tag{48}$$

Za $F^{T}(k-1).y(k-1)$ dosadíme z rovnice (39) uvažované pro *k-1* krok. Za *C(k)* dosadíme z rovnice (42) a výraz upravíme na:

$$\hat{\theta}(k)=\left[C^{-1}(k-1)+\phi(k)\phi^{T}(k)\right]^{-1}\left[C^{-1}(k-1)\hat{\theta}(k-1)+\phi(k)\phi(k)^{T}\hat{\theta}(k-1)\right]+C(k)\phi(k)\hat{e}(k) \tag{49}$$

Po vykrácení ve výrazu (49) konečně dostáváme vztah pro výpočet odhadu parametrů systému na základě hodnot z minulého kroku a přírůstku závislém buď na předchozích hodnotách nebo na měřených vstupech a výstupech:

$$\hat{\theta}(k)=\hat{\theta}(k-1)+C(k)\phi(k)\hat{e}(k) \tag{50}$$

Algoritmus výpočtu rekurzivní metody nejmenších čtverců (RLS)

Postup pro *k*-tý krok:

1. určení regresoru dat ϕ v *k*-tém kroku z měřených vstupů a výstupů

$$\phi^T(k)=\left[-y(k-1),\ldots,-y(k-n_a),u(k-1),\ldots,u(k-n_b)\right] \tag{51}$$

2. výpočet chyby predikce

$$\hat{e}(k)=y(k)-\phi^T(k)\theta(k-1) \tag{52}$$

3. výpočet kovarianční matice v *k*-tém kroku

$$C(k)=C(k-1)-\frac{C(k-1)\phi(k)\phi^T(k)C(k-1)}{1+\phi^T(k)C(k-1)\phi(k)} \tag{53}$$

4. výpočet odhadu parametrů modelu v *k*-tém kroku

$$\hat{\theta}(k)=\hat{\theta}(k-1)+C(k)\phi(k)\hat{e}(k) \tag{54}$$

3.3 Rekurzivní metoda nejmenších čtverců (RLS) s faktorem zapomínání

Rekurzivní identifikační algoritmy předpokládají v principu nasazení pro dynamicky se měnící systémy. Proto se dá předpokládat, že starší data reprezentují model hůře než data novější. Řešení tohoto problému umožňují modifikace rekurzivních identifikačních algoritmů pomocí různých metod zapomínání.

Výpočet odhadu parametrů v rovnici (54) je v podstatě přírůstkový algoritmus, kde ke starým datům přičítáme odchylku násobenou zesílením, které reprezentuje kovarianční matice. Problémem je konvergence odhadu parametrů, která vede na snižování tohoto zesílení do té míry, že přestane být citlivé na změny parametrů modelu. Metody zapomínání v rekurzivních identifikačních algoritmech se snaží udržet faktor zesílení dostatečně velký pro sledování změn systému.

3.3.1 Konstantní exponenciální zapomínání

Pro tento typ zapomínání modifikujeme kriteriální funkci (24) na:

$$J=\sum_{k=1}^{N}\lambda e^{2}(k) \tag{55}$$

Vztah pro výpočet kovarianční matice (53) se po odvození upraví na [8]:

$$C(k)=\frac{1}{\lambda}\left(C(k-1)-\frac{C(k-1)\phi(k)\phi^{T}(k)C(k-1)}{\lambda+\phi^{T}(k)C(k-1)\phi(k)}\right) \tag{56}$$

Faktor zapomínání λ se volí v rozmezí intervalu <0,1>, obvykle blíže horní meze intervalu (s hodnotami $\lambda\approx 0{,}95\text{-}0{,}99$). Je zřejmé, že pro hodnotu $\lambda=1$ algoritmus degraduje na klasickou rekurzivní identifikaci metodou nejmenších čtverců (RLS). Pro nižší hodnoty dochází k narušování konvergence kovarianční matice a algoritmus je stále citlivý na změny parametrů, ale snižuje se tím dosažitelnost ustáleného stavu. Tyto jevy působí do jisté míry protichůdně, proto byly vyvinuty algoritmy s proměnlivým faktorem zapomínání [10].

3.3.2 Proměnlivé exponenciální zapomínání

Proměnlivé exponenciální zapomínání počítá faktor λ ve výpočetních krocích základního algoritmu tak, aby se faktor zapomínání blížil asymptoticky k jedné a tím byla zajištěna konvergence výsledných parametrů. Po dosažení hledaných parametrů může ovšem algoritmus rychle degradovat na standardní rekurzivní metodu nejmenších čtverců (RLS), proto je vhodný zejména pro neměnné systémy s neurčeným přibližným odhadem parametrů, které s touto metodou rychle konvergují. Základní rovnice zapomínání v *k*-tém kroku má tvar:

$$\lambda(k)=\lambda_0\lambda(k-1)+1-\lambda_0 \tag{57}$$

Počáteční podmínky pro λ, λ_0 volíme obvykle v rozmezí <0,95;0,99>. Kovarianční matici vypočítáme podle vzorce (56).

3.3.3 Směrové zapomínání

Exponenciální zapomínání má tu nevýhodu, že probíhá ve všech směrech algoritmu stejně, tzn. i v těch směrech odkud nepřicházejí nová data. V důsledku toho může docházet k růstu hodnot kovarianční matice a tím k nestabilitě celého algoritmu. Proto byla navržena metoda směrového zapomínání, které upravuje kovarianční matici a zapomíná pouze ve směru, ve kterém dochází ke změně hodnot. V ostatních směrech algoritmus nadále konverguje beze změn. Tato úprava zabrání růstu hodnot kovarianční matice. Hodnoty faktoru směrového zapomínání určíme podle následující rovnice [14]:

$$\epsilon(k-1)=\lambda-\frac{1-\lambda}{\phi^T(k)C(k-1)\phi(k)} \tag{58}$$

Kovarianční matice se potom vypočte podle rovnice:

$$C(k)=C(k-1)-\frac{C(k-1)\phi(k)\phi^T(k)C(k-1)}{\epsilon(k-1)+\phi^T(k)C(k-1)\phi(k)} \tag{59}$$

Hodnoty faktoru zapomínání λ volíme v rozmezí intervalu <0,1>, obvykle blíže horní meze intervalu (s hodnotami $\lambda\approx 0{,}95\text{-}0{,}99$).

3.3.4 Adaptivní směrové zapomínání

Faktor zapomínání λ u směrového zapomínání lze určovat adaptivně v každém kroku, což opět zlepšuje vlastnosti celého algoritmu při změnách parametrů systému. Rovnice adaptivního směrového zapomínání potom mají tvar [5]:

$$\epsilon(k)=\varphi(k)-\frac{1-\varphi(k)}{\xi(k-1)} \tag{60}$$

$$C(k)=C(k-1)-\frac{C(k-1)\phi(k)\phi^T(k)C(k-1)}{\epsilon(k-1)+\phi^T(k)C(k-1)\phi(k)} \tag{61}$$

,kde

$$\xi(k-1)=\phi^T(k)C(k-1)\phi(k) \tag{62}$$

Faktor zapomínání *φ(k)* určíme jako:

$$\varphi(k)=\left\{1+(1+\rho)\left[\ln\left(1+\xi(k-1)\right)\right]+\left[\frac{(\upsilon(k-1)+1)\eta(k-1)}{1+\xi(k-1)+\eta(k-1)}-1\right]\frac{\xi(k-1)}{1+\xi(k-1)}\right\}^{-1} \tag{63}$$

,kde

$$\eta(k)=\frac{\hat{e}^2(k)}{\lambda(k)} \tag{64}$$

$$\upsilon(k)=\varphi(k)\left[\upsilon(k-1)+1\right] \tag{65}$$

$$\lambda(k)=\varphi(k)\left[\lambda(k-1)+\frac{\hat{e}^2(k)}{1+\xi(k-1)}\right] \tag{66}$$

Počáteční podmínky je možné nastavit blízko hodnot $\varphi(0)=1$, $\lambda(0)=0{,}001$, $\upsilon(0)=10^{-6}$, $\rho=0{,}99$ [1]. Případně je možné počáteční podmínky volit empiricky.

3.3.5 Maticové exponenciální zapomínání

V případě změn jednotlivých parametrů sytému rozdílnou rychlostí je výhodné přiřadit každému parametru jinou váhu v algoritmu zapomínání. Tento problém řeší maticové exponenciální zapomínání, které distribuuje jednotlivé váhy faktoru zapomínání parametrům pomocí matice.

Výpočet kovarianční matice je potom dán vztahem [6]:

$$C(k)=\Lambda(k-1)\left(I-\frac{\phi(k)\phi^T(k)\Lambda(k-1)}{1+\phi^T(k)\Lambda(k-1)\phi(k)}\right) \tag{67}$$

Modifikovaná kovarianční matice Λ se určí jako:

$$\Lambda(k-1)=\Omega\, C(k-1)\,\Omega^T \tag{68}$$

Vlastní matici vah faktorů zapomínání pro jednotlivé parametry Ω vyjádříme:

$$\Omega=\begin{bmatrix}\frac{1}{\sqrt{\lambda_1}} & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \frac{1}{\sqrt{\lambda_n}}\end{bmatrix} \tag{69}$$

Faktory λ_1 až λ_n jsou jednotlivé váhy přidělené hledaným parametrům systému. Výpočet vektoru odhadu hledaných parametrů upravíme na:

$$\hat{\theta}(k)=\hat{\theta}(k-1)+\frac{\Lambda(k-1)\phi(k)}{1+\phi^T(k)\Lambda(k-1)\phi(k)}\hat{e}(k) \tag{70}$$

3.3.6 Konstantní stopa

Výše zmiňované algoritmy se snaží nějakým způsobem narušovat velikost prvků kovarianční matice, tak aby hodnoty nepoklesly do stavu, kdy algoritmus není schopen reagovat na změny. Na druhou stranu se může stát, že prvky kovarianční matice porostou do numericky nestabilních hodnot. Proto byl modifikován algoritmus konstantního exponenciálního zapomínání, tak aby kovarianční matice měla konstantní stopu. Základní kovarianční matice se vypočte jako u exponenciálního zapomínání:

$$\bar{C}(k)=\frac{1}{\lambda}\left(C(k-1)-\frac{C(k-1)\phi(k)\phi^T(k)C(k-1)}{\lambda+\phi^T(k)C(k-1)\phi(k)}\right) \tag{71}$$

Takto získaná hodnota se dále modifikuje, tak aby výsledná matice měla konstantní stopu.

$$C(k)=c_1\frac{\bar{C}(k)}{tr\left(\bar{C}(k)\right)}+c_2 I \tag{72}$$

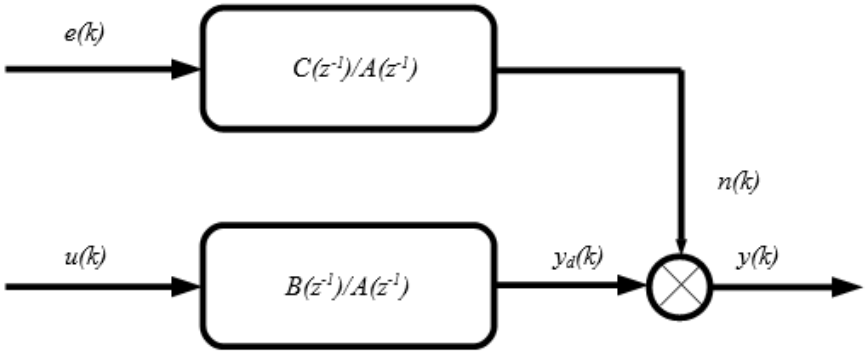
Kde hodnoty c_1 a c_2 jsou definovány jako kladné podle c_1/c_2 =10000.

3.4 Rozšířená rekurzivní metoda nejmenších čtverců (RELS)

Standardní rekurzivní metoda nejmenších čtverců předpokládá ovlivnění systému bílým šumem podle rovnice (73) ARX modelu.

$$A(z^{-1})y(k)=B(z^{-1})u(k)+n(k) \tag{73}$$

Pro systémy na které působí šumy barevného charakteru je výhodnější použít ARMAX model (Obr. 8).



Obr. 8: Blokové schema ARMAX modelu pro metodu RELS

Působící šum filtrujeme filtrem charakterizovaným v rovnici (74) polynomem *C*.

$$A(z^{-1})y(k)=B(z^{-1})u(k)+C(z^{-1})n(k) \tag{74}$$

Při identifikaci takovýchto systémů může být samozřejmě použita nijak nemodifikovaná rekurzivní metoda nejmenších čtverců (RLS), ale obecně bude dávat horší výsledky než její rozšíření (RELS) pro model ARMAX.

Modifikace rekurzivní metody nejmenších čtverců spočívá v rozšíření vektoru hledaných parametrů systému Θ o neznámé parametry filtru reprezentovaného polynomem $C(z^{-1})$. O stejný počet hodnot je nutné rozšířit regresor dat Φ a to ideálně hodnotami šumu $n(k-1), \ldots, n(k-n_C)$. Poruchový šum $n(k)$ je ovšem z podstaty věci nepozorovatelný, proto existuje řada přístupů jak data tohoto šumu v rovnicích nahradit [12]. Nejčastěji se využívá náhrady pomocí chyby predikce nebo residuí.

Chyba predikce

Počítá se z odhadu parametrů modelu v *(k-1)* kroku *Θ(k-1)*.

$$\hat{e}(k)=y(k)-\phi^T(k)\theta(k-1) \tag{75}$$

Reziduum

Počítá se z odhadu parametrů modelu v *k*-tém kroku *Θ(k)*.

$$\eta(k)=y(k)-\phi^T(k)\theta(k) \tag{76}$$

Výpočetní algoritmus se po naplnění vektorů nijak neliší od algoritmu pro rekurzivní metodu nejmenších čtverců.

Algoritmus výpočtu rozšířené rekurzivní metody nejmenších čtverců (RELS)

Postup pro *k*-tý krok:

1. určení regresoru dat ϕ v *k*-tém kroku z měřených vstupů a výstupů

$$\phi^T(k)=\left[-y(k-1),\ldots,-y(k-n_a),u(k-1),\ldots,u(k-n_b)\right] \tag{77}$$

2. výpočet chyby predikce

$$\hat{e}(k)=y(k)-\phi^T(k)\theta(k-1) \tag{78}$$

3. výpočet kovarianční matice v *k*-tém kroku

$$C(k)=C(k-1)-\frac{C(k-1)\phi(k)\phi^T(k)C(k-1)}{1+\phi^T(k)C(k-1)\phi(k)} \tag{79}$$

4. výpočet odhadu parametrů modelu v *k*-tém kroku

$$\hat{\theta}(k)=\hat{\theta}(k-1)+C(k)\phi(k)\hat{e}(k) \tag{80}$$

Samozřejmě na toto rozšíření lze aplikovat stejné modifikace zapomínání jako na klasickou rekurzivní identifikaci metodou nejmenších čtverců.

3.5 Rekurzivní metoda instrumentální proměnné (RIV)

Klasická rekurzivní metoda nejmenších čtverců (RLS) předpokládá ovlivnění systému poruchovou veličinou typu bílého šumu. Jestliže má poruchová veličina převažující vlastnosti barevného šumu, tak metody založené na prosté metodě nejmenších čtverců pro ARX model nemusí dávat konsistentní výsledky. Metoda instrumentální proměnné (Recursive Instrumental Variables) identifikuje systém zašuměný poruchovou veličinou povahy barevného šumu, ale uvažuje model systému typu ARX. Metoda vychází z pozměněné kriteriální funkce (30), kde neminimalizujeme kvadrát chyby predikce *e*, ale součin chyby predikce a tzv. instrumentální proměnné *Z*, která by měla být nezávislá na šumu ovlivňujícím systém [8].

$$J = z\,e = z\left(y - F\,\theta\right) \rightarrow min \tag{81}$$

Algoritmus výpočtu rekurzivní metody instrumentální proměnné (RIV)

Postup pro *k*-tý krok:

1. určení regresoru dat ϕ v *k*-tém kroku z měřených vstupů a výstupů, určení instrumentální proměnné *z* (viz. dále)

$$\phi^T(k) = \left[-y(k-1), \ldots, -y(k-n_a), u(k-1), \ldots, u(k-n_b)\right] \tag{82}$$

2. výpočet chyby predikce

$$\hat{e}(k) = y(k) - \phi^T(k)\,\theta(k-1) \tag{83}$$

3. výpočet kovarianční matice v *k*-tém kroku

$$C(k) = C(k-1) - \frac{C(k-1)\,z(k)\,\phi^T(k)\,C(k-1)}{1 + \phi^T(k)\,C(k-1)\,z(k)} \tag{84}$$

4. výpočet odhadu parametrů modelu v *k*-tém kroku

$$\hat{\theta}(k) = \hat{\theta}(k-1) + C(k)\,\phi(k)\,\hat{e}(k) \tag{85}$$

Volba instrumentální proměnné *z(k)*

Jak již bylo zmíněno, účelem instrumentální proměnné je potlačení vlivu barevného šumu na výpočet odhadu parametrů systému. Instrumentální proměnnou je možné sestavit dvěma

základními přístupy.

- Instrumentální proměnná nezávislá na modelu

  Pro sestavení vektoru dat se použijí pouze vstupy do systému, které nejsou šumem ovlivněny. Hloubka paměti dat musí pokrýt počet hledaných parametrů.

$$z(k)^T=\left[u(k-1),\ldots,u(k-n_a-n_b)\right] \tag{86}$$

- Instrumentální proměnná závislá na modelu

  Vektor dat má podobný tvar jako regresor (82) s tím rozdílem, že jako hodnoty výstupů jsou použity predikce výstupů.

$$z(k)^T=\left[-\hat{y}(k-1),\ldots,-\hat{y}(k-n_a),u(k-1),\ldots,u(k-n_b)\right] \tag{87}$$

  , kde

$$\hat{y}(k)=\frac{\hat{B}(z^{-1})}{\hat{A}(z^{-1})}u(k) \tag{88}$$

Predikované hodnoty *y* by měly být přepočítávány v každém kroku podle aktuálních odhadů parametrů modelu. To je ovšem značně výpočetně náročné, proto je možné vypočítat z aktuálních hodnot odhadu parametrů vždy jen odhad výstupu v aktuálním kroku *k* a pro hodnoty z kroků *(k-i)* použít hodnoty z předchozích rekurzí.

$$\hat{y}(k)=\hat{b_1}(k)u(k-1)+\ldots+\hat{b_{nb}}(k)u(k-n_b)-\hat{a}_1(k)\hat{y}(k-1),\ldots,-\hat{a_{na}}(k)\hat{y}(k-n_a) \tag{89}$$

Pokud jsou počáteční odhady parametrů modelu nepřesné, tak je možné pro odhad parametrů použít klasickou rekurzivní metodu nejmenších čtverců.

Na metodu instrumentální proměnné je možné aplikovat stejné modifikace algoritmu jako na rekurzivní metodu nejmenších čtverců.

3.6 Rozšířená rekurzivní metoda instrumentální proměnné (ERIV)

Rozšíření metody instrumentální proměnné zlepšuje celkovou konvergenci rekurzivního algoritmu. Její základ spočívá v rozšíření vektoru instrumentální proměnné na více hodnot než je počet neznámých parametrů. Tyto nesouhlasné dimenze vektorů regresoru a instrumentální proměnné si však vyžadují podstatnou úpravu algoritmu, i když celkové řešení vychází stále z minimalizace pomocí metody nejmenších čtverců [4].

Algoritmus výpočtu rozšířené rekurzivní metody instrumentální proměnné (ERIV)

Postup pro *k*-tý krok:

1. určení regresoru dat ϕ v *k*-tém kroku z měřených vstupů a výstupů, určení instrumentální proměnné *z* (viz. dále)

$$\phi^T(k)=\left[-y(k-1),\ldots,-y(k-n_a),u(k-1),\ldots,u(k-n_b)\right] \tag{90}$$

2. určení a výpočet pomocných vektorů a matic *v, w, Λ* a *Φ*

$$v(k)=\begin{bmatrix} z^T(k)r(k-1) \\ y(k) \end{bmatrix} \tag{91}$$

$$w(k)=R^T(k-1)z(k) \tag{92}$$

$$\Phi(k)=\left[w(k)\phi(k)\right] \tag{93}$$

$$\Lambda(k)=\begin{bmatrix} -z^T(k)z(k) & 1 \\ 1 & 0 \end{bmatrix} \tag{94}$$

3. výpočet matice *L* v *k*-tém kroku

$$L(k)=P(k-1)\Phi(k)\left(\Lambda(k)+\Phi^T(k)P(k-1)\Phi(k)\right)^{-1} \tag{95}$$

4. výpočet odhadu parametrů modelu v *k*-tém kroku

$$\hat{\theta}(k)=\hat{\theta}(k-1)+L(k)\left(v(k)-\Phi^T(k)\hat{\theta}(k-1)\right) \tag{96}$$

5. výpočet pomocných vektorů a matic *R, r* a *P* pro další krok

$$R(k)=R(k-1)+z(k)\phi^T(k) \tag{97}$$

$$r(k)=r(k-1)+z(k)y(k) \tag{98}$$

$$P(k)=P(k-1)-L(k)\Phi^{T}(k)P(k-1) \tag{99}$$

Hloubka paměti vektoru instrumentální proměnné *z* je volitelná, teoreticky by měla při zvětšování zlepšovat konvergenci algoritmu, ale při výrazně větší velikosti než je velikost regresoru se může negativně projevovat stárnutí dat.

3.7 Rekurzivní metoda predikčních chyb (RPEM)

Rekurzivní metoda predikčních chyb (Recursive prediction error method) je v podstatě univerzální metodou ke všem dříve uvedeným metodám. Základním principem určení hledaných parametrů je optimalizace pomocí nelineární Gauss-Newtonovy metody nejmenších čtverců [12], [7]. Nelineární je tato metoda vzhledem k vektoru hledaných parametrů.

3.7.1 Metoda RPEM pro model ARMAX

Tato metoda je v podstatě zobecněním jednorozměrné Newtonovy optimalizační metody tečen pro vícerozměrný prostor. Proto pro vyhledávání extrémů ztrátové funkce využívá gradientu, který bude v tomto případě definován jako vektor *ψ(k)* parciálních derivací odhadů výstupu podle jednotlivých parametrů:

$$\psi(k)=\frac{\partial\hat{y}(k)}{\partial\Theta(k)}=\left[\frac{\partial\hat{y}(k)}{\partial\Theta_1(k)}\frac{\partial\hat{y}(k)}{\partial\Theta_2(k)}\cdots\frac{\partial\hat{y}(k)}{\partial\Theta_n(k)}\right]^{T} \tag{100}$$

Rovnici pro odhad výstupů modelu ARMAX můžeme zapsat jako:

$$\hat{y}(k)=\frac{B(z^{-1})}{C(z^{-1})}u(k)+\left(1-\frac{A(z^{-1})}{C(z^{-1})}\right)y(k) \tag{101}$$

Pro odhad chyby predikce platí:

$$\hat{e}(k)=y(k)-\hat{y}(k) \tag{102}$$

Kombinací rovnic (101) a (102) dostaneme rovnici:

$$\hat{e}(k)=\frac{A(z^{-1})}{C(z^{-1})}y(k)+\frac{B(z^{-1})}{C(z^{-1})}u(k) \tag{103}$$

Polynomy $A(z^{-1}),B(z^{-1})$ a $C(z^{-1})$ jsou ve tvaru:

$$A(z^{-1})=1+a_1 z^{-1}+...+a_{na} z^{-n_a} \tag{104}$$

$$B(z^{-1})=b_1 z^{-1}+...+b_{nb} z^{-n_b} \tag{105}$$

$$C(z^{-1})=1+c_1 z^{-1}+...+c_{nc} z^{-n_c} \tag{106}$$

Pro určení gradientu $\psi(k)$ upravíme rovnici (101) na:

$$C(z^{-1})\hat{y}(k)=B(z^{-1})u(k)+\left(C(z^{-1})-A(z^{-1})\right)y(k) \tag{107}$$

Derivací rovnice (107) podle jednotlivých odhadů parametrů modelu dostaneme složky gradientu:

$$\frac{\partial \hat{y}(k)}{\partial a_i}=-\frac{1}{C(z^{-1})}y(k-i) \tag{108}$$

$$\frac{\partial \hat{y}(k)}{\partial b_i}=\frac{1}{C(z^{-1})}u(k-i) \tag{109}$$

Derivaci podle parametrů filtru c_i dostaneme derivováním rovnice (107) podle pravidel derivování pro součin funkcí:

$$\hat{y}(k-i)+C(z^{-1})\frac{\partial \hat{y}(k)}{\partial c_i}=y(k-i) \tag{110}$$

$$\frac{\partial \hat{y}(k)}{\partial c_i}=\frac{1}{C(z^{-1})}\left(y(k-i)-\hat{y}(k-i)\right)=\frac{1}{C(z^{-1})}\hat{e}(k-i) \tag{111}$$

Jednotlivé složky gradientu z rovnic (108), (109) a (111) můžeme přepsat jako modifikaci standardních složek regresoru pomocí filtru $1/C(z^{-1})$:

$$y^F(k-i)=\frac{1}{C(z^{-1})}y(k-i) \tag{112}$$

$$u^F(k-i)=\frac{1}{C(z^{-1})}u(k-i) \tag{113}$$

$$\hat{e}^F(k-i)=\frac{1}{C(z^{-1})}\hat{e}(k-i) \tag{114}$$

Gradient $\psi(k)$ potom můžeme zapsat jako:

$$\psi(k)=\left[-y^F(k-1),...,-y^F(k-n_a),u^F(k-1),...,u^F(k-n_b),\hat{e}^F(k-1),...,\hat{e}^F(k-n_c)\right]^T \tag{115}$$

Jednotlivé složky gradientu z rovnice (115) vypočteme jako:

$$y^F(k)=y(k)-\hat{c}_1(k)y^F(k-1)-...-\hat{c}_n(k)y^F(k-n) \tag{116}$$

$$u^F(k)=u(k)-\hat{c}_1(k)u^F(k-1)-\ldots-\hat{c}_n(k)u^F(k-n) \tag{117}$$

$$\hat{e}^F(k)=\hat{e}(k)-\hat{c}_1(k)\hat{e}^F(k-1)-\ldots-\hat{c}_n(k)\hat{e}^F(k-n) \tag{118}$$

Kde chybu predikce určíme podle rovnice:

$$\hat{e}(k)=y(k)-\phi^T(k)\theta(k-1) \tag{119}$$

Chybu predikce vypočítanou z odhadu parametrů modelu *(k-1)* kroku *Θ(k-1)* lze případně nahradit reziduem vypočítaným z odhadu parametrů modelu *k*-tého kroku *Θ(k)* podle rovnice (120).

$$\eta(k)=y(k)-\phi^T(k)\theta(k) \tag{120}$$

Gradient bude potom ve tvaru:

$$\psi(k)=\left[-y^F(k-1),\ldots,-y^F(k-n_a),u^F(k-1),\ldots,u^F(k-n_b),\eta^F(k-1),\ldots,\eta^F(k-n_c)\right]^T \tag{121}$$

Filtrovanou hodnotu rezidua *η(k)* určíme jako:

$$\eta^F(k)=\eta(k)-\hat{c}_1(k)\eta^F(k-1)-\ldots-\hat{c}_n(k)\eta^F(k-n_c) \tag{122}$$

Algoritmus výpočtu rekurzivní metody predikčních chyb (RPEM)

Postup pro *k*-tý krok:

1. určení regresoru dat ψ v *k*-tém kroku z měřených vstupů, výstupů a chyby predikce,

$$\psi(k)=\left[-y^F(k-1),\ldots,-y^F(k-n_a),u^F(k-1),\ldots,u^F(k-n_b),\hat{e}^F(k-1),\ldots,\hat{e}^F(k-n_c)\right]^T \tag{123}$$

2. výpočet chyby predikce

$$\hat{e}(k)=y(k)-\phi^T(k)\theta(k-1) \tag{124}$$

3. výpočet kovarianční matice v *k*-tém kroku

$$P(k)=P(k-1)-\frac{P(k-1)\psi(k)\psi^T(k)P(k-1)}{1+\psi^T(k)P(k-1)\psi(k)} \tag{125}$$

4. výpočet odhadu parametrů modelu v *k*-tém kroku

$$\hat{\theta}(k)=\hat{\theta}(k-1)+P(k)\phi(k)\hat{e}(k) \tag{126}$$

3.7.2 Metoda RPEM pro model OE

Identifikace modelu OE je odlišná v tom, že nevyžaduje hodnoty výstupu procesu. Celý algoritmus identifikace pracuje pouze s predikovanými hodnotami výstupů a konvergence je zajištěna až porovnáváním chyby výstupu.

Rovnici pro odhad výstupů modelu OE zapíšeme jako:

$$\hat{y}(k)=\frac{B(z^{-1})}{F(z^{-1})}u(k) \tag{127}$$

Pro odhad chyby predikce platí z rovnic (102) a (127):

$$\hat{e}(k)=y(k)-\frac{B(z^{-1})}{F(z^{-1})}u(k) \tag{128}$$

Polynomy $B(z^{-1})$ a $F(z^{-1})$ jsou ve tvaru:

$$B(z^{-1})=b_1 z^{-1}+\ldots+b_{nb}z^{-n_b} \tag{129}$$

$$F(z^{-1})=1+f_1 z^{-1}+\ldots+f_{nf}z^{-n_f} \tag{130}$$

Pro určení gradientu $\psi(k)$ upravíme rovnici (127) na:

$$F(z^{-1})\hat{y}(k)=B(z^{-1})u(k) \tag{131}$$

Derivací rovnice (131) podle jednotlivých odhadů parametrů modelu dostaneme složky gradientu:

$$\frac{\partial \hat{y}(k)}{\partial f_i}=-\frac{1}{F(z^{-1})}\hat{y}(k-i) \tag{132}$$

$$\frac{\partial \hat{y}(k)}{\partial b_i}=\frac{1}{F(z^{-1})}u(k-i) \tag{133}$$

Jednotlivé složky gradientu z rovnic (132) a (133) můžeme přepsat jako modifikaci standardních složek regresoru pomocí filtru $1/F(z^{-1})$:

$$\hat{y}^F(k-i)=\frac{1}{F(z^{-1})}\hat{y}(k-i) \tag{134}$$

$$u^F(k-i)=\frac{1}{F(z^{-1})}u(k-i) \tag{135}$$

Gradient $\psi(k)$ potom můžeme zapsat jako:

$$\psi(k)=\left[-\hat{y}^F(k-1),\ldots,-\hat{y}^F(k-n_a),u^F(k-1),\ldots,u^F(k-n_b)\right]^T \tag{136}$$

Jednotlivé složky gradientu z rovnice (136) vypočteme jako:

$$\hat{y}^F(k)=\hat{y}(k)-\hat{f}_1(k)\hat{y}^F(k-1)-\ldots-\hat{f}_n(k)\hat{y}^F(k-n) \tag{137}$$

$$u^F(k)=u(k)-\hat{f}_1(k)u^F(k-1)-\ldots-\hat{f}_n(k)u^F(k-n) \tag{138}$$

Algoritmus výpočtu rekurzivní metody predikčních chyb (RPEM)

Postup pro *k*-tý krok:

1. určení regresoru dat ψ v *k*-tém kroku z měřených vstupů, výstupů a chyby predikce,

$$\psi(k)=\left[-\hat{y}^F(k-1),\ldots,-\hat{y}^F(k-n_a),u^F(k-1),\ldots,u^F(k-n_b)\right]^T \tag{139}$$

2. výpočet chyby predikce

$$\hat{e}(k)=y(k)-\phi^T(k)\theta(k-1) \tag{140}$$

3. výpočet kovarianční matice v *k*-tém kroku

$$P(k)=P(k-1)-\frac{P(k-1)\psi(k)\psi^T(k)P(k-1)}{1+\psi^T(k)P(k-1)\psi(k)} \tag{141}$$

4. výpočet odhadu parametrů modelu v *k*-tém kroku

$$\hat{\theta}(k)=\hat{\theta}(k-1)+P(k)\phi(k)\hat{e}(k) \tag{142}$$

II. PRAKTICKÁ ČÁST

4 SCILAB

Program Scilab byl vytvořen v devadesátých letech minulého století francouzskými vědeckými institucemi INRIA (The French National Institute for Research in Computer Science and Control) a ENPC (École Nationale des Ponts et Chaussées) primárně v jazycích C, C++, Java a Fortran. Tento program navazuje na software Blaise, který v osmdesátých letech vyvíjela firma CACSD (Computer Aided Control System Design) jako výpočetní program pro úlohy z oblasti automatického řízení. Později program změnil název na Basile, který distribuoval zprostředkovaně již zmíněný institut INRIA. V roce 1994 byla uvedena první verze programu Scilab 1.1. V roce 2003 vytvořil institut INRIA Scilab konsorcium za účelem lepší podpory programu, následně došlo v roce 2008 k jeho rozšíření. Od roku 2010 je vydavatelem programu společnost Scilab Enterprises.

Aktuálně (2018) je k dispozici verze Scilab 6.0. Základní licence umožňuje bezplatné použití, rozšiřující moduly a případná podpora mohou být i komerční, častěji jsou však volně šiřitelné. Od verze 5 je šířen pod licencí CeCILL, která, jak uvádí Free Software Foundation, je kompatibilní s rodinou GPL licencí.

V současné době je Scilab dostupný pro následující platformy: GNU/Linux, Mac OS X a Windows Vista/7/8/10. Základní informace a verze programu ke stažení jsou dostupné na adrese: http://www.scilab.org. Ve standardní instalaci obsahuje stovky základních matematických funkcí, datových struktur a vizualizačních nástrojů, které jsou strukturovány do jednotlivých modulů. Nejvýznamněji jsou zahrnuty a využívány funkcionality z oblastí:

- **Matematika a simulace**

  Základní sady funkcí pro matematiku i aplikované obory zahrnující matematické operace i analýzu dat.

- **2-D & 3-D Vizualizace**

  Grafické nástroje pro vizualizaci, komentování a export dat prostřednictvím různých typů grafů.

- **Optimalizace**

  Algoritmy pro řešení spojitých i diskrétních problémů v oblasti optimalizace.

- **Statistika**

  Nástroje pro analýzu a modelování statistických dat.

- **Analýza a simulace systémů**

  Standardní nástroje a algoritmy pro řešení problémů z oblasti automatizace a teorie systémů.

- **Zpracování signálů**

  Vizualizace, analýza a filtrování signálů.

- **Vývoj aplikací**

  Nástroje pro vytváření uživatelských aplikací a modulů, nástroje pro výměnu a export dat s jinými programy.

- **Xcos – Hybridní dynamický simulátor systémů**

  Nástroj umožňuje grafické modelování a simulaci dynamických systémů.

Kromě těchto vestavěných základních funkcionalit, které jsou součástí standardního instalačního balíku, jsou k dispozici různé nadstavby. Scilab se od verze 5.2 snaží upustit od používání termínu toolbox a tyto nadstavby distribuuje pod pojmem moduly buď jako moduly vyvinuté týmem Scilab Enterprises na adrese: http://www.scilab.org/scilab/modules nebo jako externí moduly dostupné na portálu ATOMS: http://atoms.scilab.org. Dalším zdrojem rozšíření je portál pro výměnu souborů (skriptů, funkčních bloků atd.) file exchange na adrese: https://fileexchange.scilab.org.

Instalace

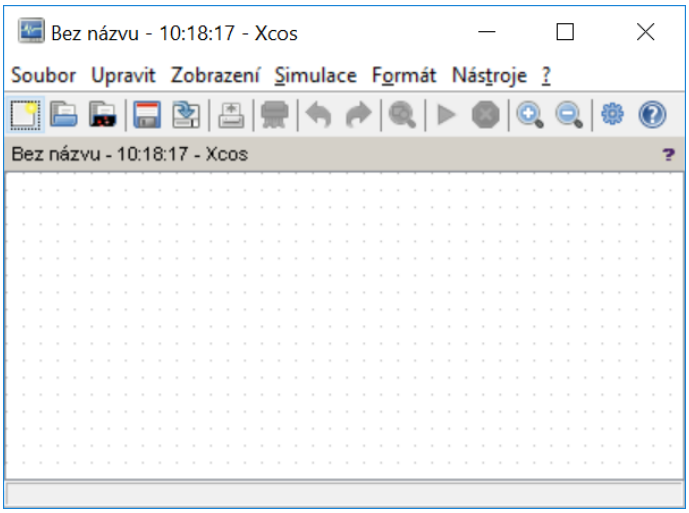
Systém lze stáhnout z webových stránek https://www.scilab.org. Kromě standardní instalace softwaru je vhodné doinstalovat některé moduly pomocí správce modulů ATOMS. V případě vývoje nových bloků apod. je nezbytný modul MinGW toolbox ze složky Windows Tools, který obsahuje externí MinGW kompilátor pro instalovanou verzi operačního systému [11].

4.1 Xcos

Jak už bylo zmíněno nadstavba Xcos funguje nejen jako nástroj grafického modelování, ale obsahuje i vlastní řešitel diferenciálních rovnic charakteristických pro dynamické procesy. V dřívějších verzích byl distribuován pod názvem Scicos [2], [13].

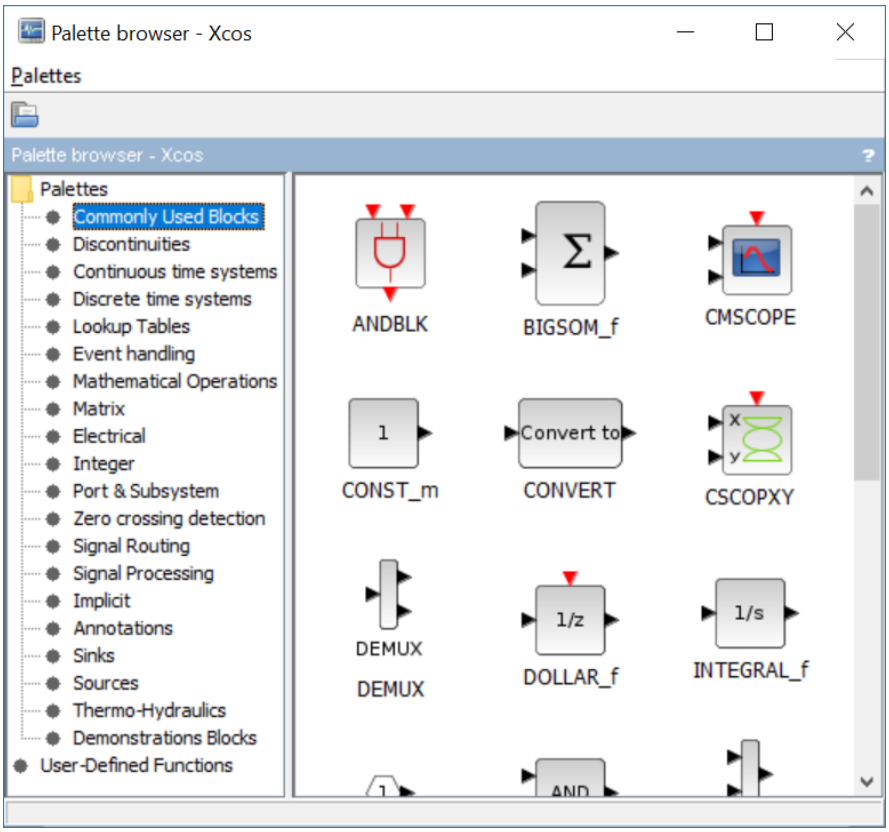
Celý systém funguje na základě sestavování modelů z jednotlivých bloků, tyto modely jsou potom zkompilovány a simulovány interním řešitelem. Základní prostředí se skládá ze tří částí:

- **Editační okno** – umožňuje základní sestavování modelu z jednotlivých bloků



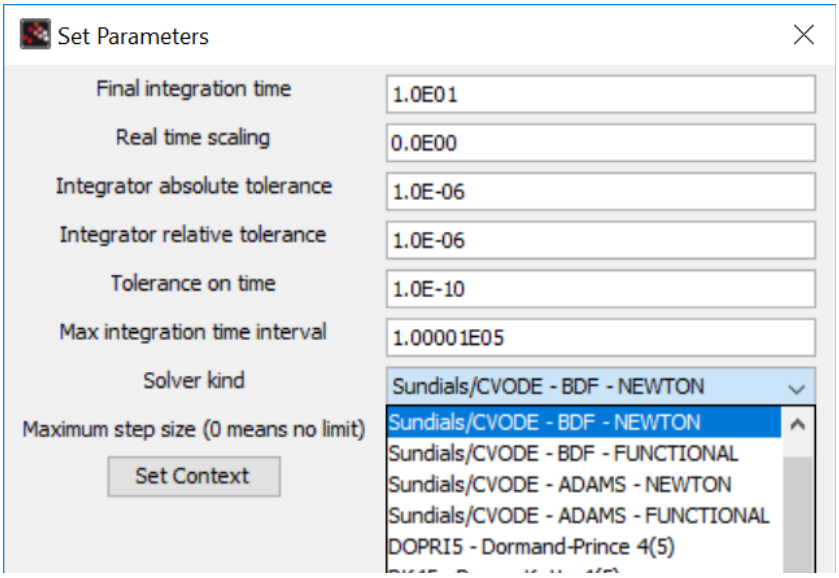
Obr. 9: Editační okno Xcos

- **Průzkumník palet** – zde se nachází knihovna jednotlivých standardně definovaných bloků se kterými lze přetažením pracovat v editačním okně. K dispozici jsou zejména bloky z dílčích knihoven:
  - *Continuous time systems* – spojité systémy
  - *Discrete time systems* – diskrétní systémy
  - *Mathematical Operations* – základní matematické operace
  - *Matrix* – knihovna maticových operací
  - *Sources* – zdroje signálů
  - *Sinks* – zobrazení výsledků



Obr. 10: Průzkumník palet Xcos

- **Kompilátor a simulátor** – Xcos umožňuje v záložce editačního okna *Simulation* provádět základní nastavení kompilace a ladění, pro simulaci je k dispozici nastavení délky simulace, řešitele a dalších možností (Obr. 11).



Obr. 11: Nastavení parametrů simulace pro Xcos

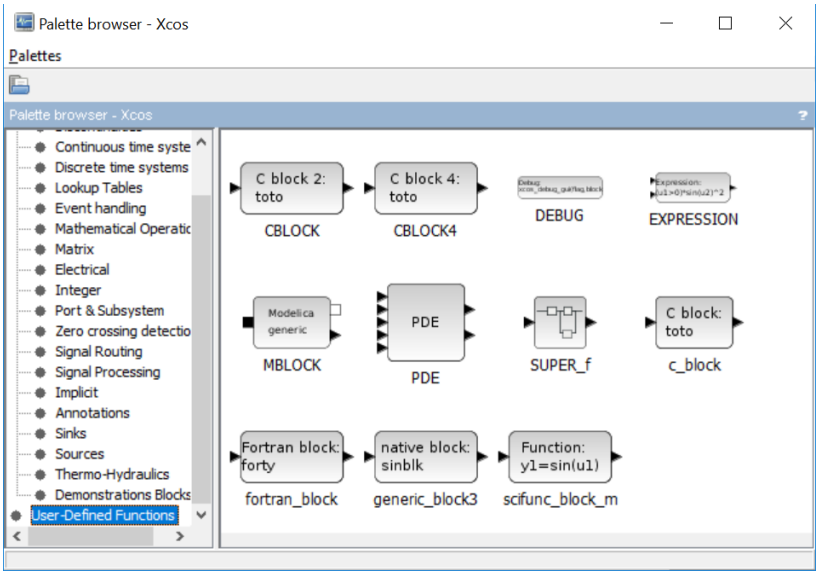
4.2 Realizace nového bloku v Xcosu

Paleta uživatelsky definovaných funkcí

Kromě jednotlivých knihoven standardních bloků umožňuje Xcos vytvoření vlastních uživatelsky definovaných bloků úpravou bloků v paletě *User defined functions*.

Pro definovaní vlastních funkcí lze nejčastěji využít bloků:

- *EXPRESSION* – výpočet jednoduchých matematických výrazů ze skalárních vstupů
- *scifunc_block_m* – definice funkce pomocí skriptovacího jazyka Scilabu
- *fortran_block* - definice funkce pomocí programovacího jazyka FORTRAN
- *PDE* – grafický řešitel parciálních diferenciálních rovnic
- *CBLOCK4* - definice funkce pomocí programovacího jazyka ANSI C (dále označovaná jako C blok)
- *SUPER_f* – složení vybrané skupiny bloků do nového superbloku



Obr. 12: Paleta uživatelsky definovaných funkcí Xcos

Datová struktura bloků Xcos

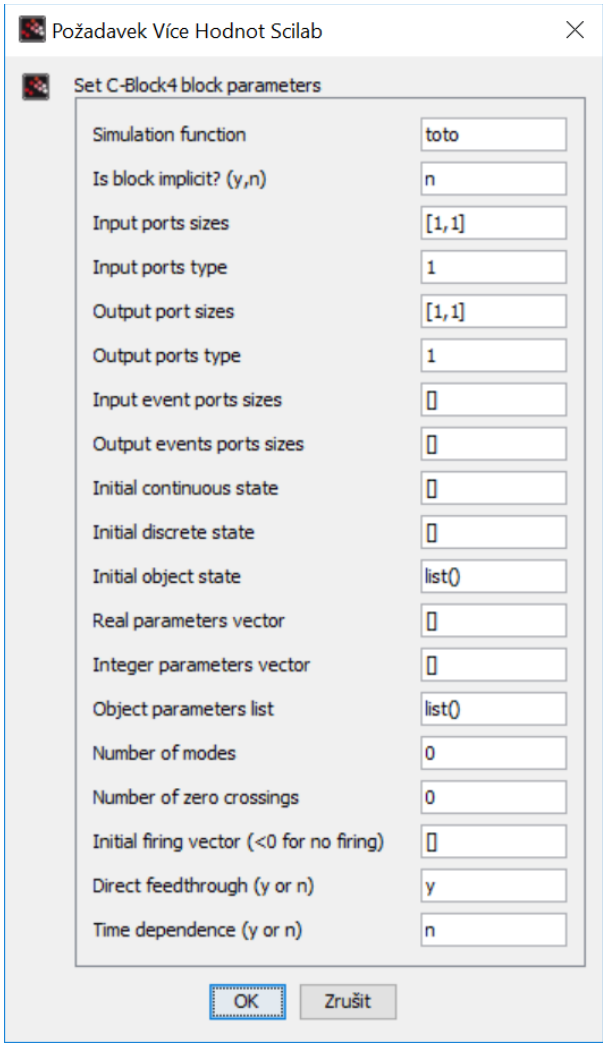
Bloky Xcos pracují s daty ve formě struktury s jednotlivými proměnnými dle obrázku (Obr. 13). Kromě pole názvu výpočetní funkce (*sim*) jsou důležité pole pro práci s vektory vstupů a výstupů (*in, in2, intyp, out, out2, outtyp*). Dále pak předávané parametry celých čísel (*ipar*), reálných čísel (r*par*) a objektů např. ve formě vektorů (*opar*).

Fields	Description	Type/size	Example
sim	Name/type of the computational function.	list of size 2	model.sim=list('tows_c',4)
in	First dimensions of regular input ports.	vector of size nin	model.in=[1;2]
in2	Second dimensions of regular input ports.	vector of size nin	model.in2=[3;1]
intyp	Data type of regular input ports.	vector of size nin	model.intyp=[1;8]
out	First dimensions of regular output ports.	vector of size nout	model.out=[1;2]
out2	Second dimensions of regular output ports.	vector of size nout	model.out2=[3;1]
outtyp	Data type of regular output ports.	vector of size nout	model.outtyp=[1;8]
evtin	Size of event input ports.	vector of size nevin	model.evtin=[1;1]
evtout	Size of event output ports.	vector of size nevout	model.evtout=[1;1]
state	Initial condition of continuous state.	vector of size nx	model.state=[0;0.1;-5.1]
dstate	Initial condition of discrete state.	vector of size nz	model.dstate=[0;0;-1]
odstate	Initial condition of object discrete state.	list of size noz	model.odstate=list([0;0;-1],... int32(3))
ipar	Integer parameters.	vector of size nipar	model.ipar=[1;2;-6]
rpar	Real parameters.	vector of size nrpar	model.rpar=[0.8;2.1;-6.55]
opar	Object parameters.	list of size nopar	model.opar=list([0.8;2.1],,... 1+2*%i)
blocktype	Type of the block.	character	model.blocktype='d'
firing	Initial date of output events.	vector of size nevout	model.firing=[-1;0.1]
dep_ut	Scheduling properties.	boolean vector of size 2	model.dep_ut=[%t;%f]
label	Label of the block.	string	model.label=["My label"]
nzcross	Number of zero crossing.	integer	model.nzcross=1
nmode	Number of modes.	integer	model.nmode=0
equations	Modelica block definition.	list of size 4	model.equations=modelica(); model.equations.model=... 'Capacitor' model.equations.inputs='p' model.equations.outputs='n' model.equations.parameters=... list(['C','v'],list(C,v),[0,1])

Obr. 13: Datová struktura obecného bloku

Výpočetní funkce C bloku

Výpočetní funkce pro bloky v programovacím jazyce C je realizována jako standardní funkce jazyka C s návratovým typem void a parametry volání ukazatele na blok pro který je volána a celočíselného příznaku *flag*. Kromě jiných použitých hlavičkových souborů musí být připojen soubor *scicos/scicos_block4.h* , který daná funkce rozšiřuje. Typy proměnných, vstupy a výstupy a parametry jsou funkci předávány pomocí masky bloku (Obr. 14).



Obr. 14: Maska bloku C funkce

Typy proměnných

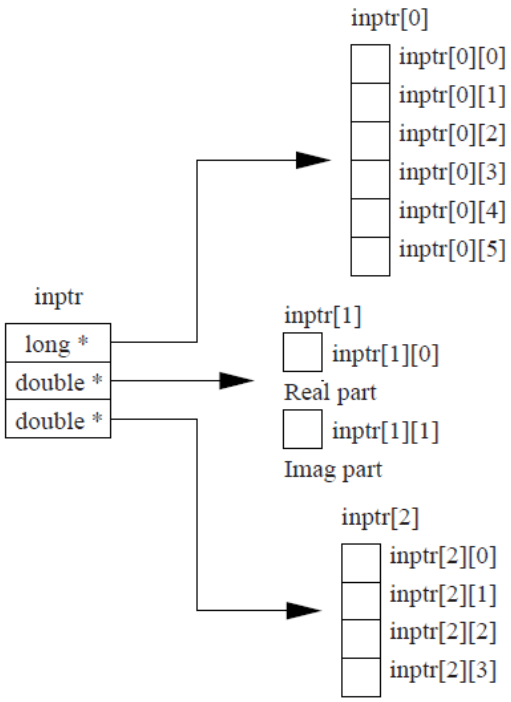
Xcos pracuje s typy proměnných jazyka C dle obrázku (Obr. 15). Na vstupu i výstupu z funkce tyto typy konvertuje na svůj interní formát.

C		
Number	Type	Macros
10	double	SCSREAL_COP
11	double	SCSCOMPLEX_COP
84	long int	SCSINT32_COP
82	short	SCSINT16_COP
81	char	SCSINT8_COP
814	unsigned long int	SCSUINT32_COP
812	unsigned short	SCSUNINT16_COP
811	unsigned char	SCSUINT8_COP
-1	double	SCSUNKNOWN_COP

Obr. 15: Typy proměnných C bloku

Vstupy

Vstupy definované funkce jsou načítány pomocí proměnné struktury bloku *block→inptr*, která zpřístupňuje data na vstupu do bloku ve formě pole ukazatelů na jednotlivé typy proměnných vstupů. Pokud budeme mít aktivní tři vstupy (matice typu long 3x2, komplexní číslo typu double a vektor typu double 4x1), pak tyto vstupy načteme pomocí *block→inptr* s indexováním polí dle obrázku (Obr. 16).



Obr. 16: Načtení vstupů funkce

Pokud však nechceme načítat vstupy přímo, pak je lepší využít pro načtení funkce hlavičkového souboru *scicos/scicos_block4.h*, pro načtení jednotlivých typů proměnných:

- *GetRealInPortPtrs(block,x)*
- *GetImagInPortPtrs(block,x)*
- *Getint8InPortPtrs(block,x)*
- *Getint16InPortPtrs(block,x)*
- *Getint32InPortPtrs(block,x)*

- *Getuint8InPortPtrs(block,x)*
- *Getuint16InPortPtrs(block,x)*
- *Getuint32InPortPtrs(block,x)*

Jako parametr předáváme funkci instanci bloku a číslo vstupu, návratová hodnota je ukazatel na pole odpovídající typu proměnné (Obr. 15).

Výstupy

Výstupy funkce jsou definovány analogicky jako vstupy, tzn. lze k nim přistupovat přímo pomocí ukazatelů na jednotlivé výstupy a typy proměnných nebo pomocí funkcí hlavičkového souboru *scicos/scicos_block4.h*:

- *GetRealOutPortPtrs(block,x)*
- *GetImagOutPortPtrs(block,x)*
- *Getint8OutPortPtrs(block,x)*
- *Getint16OutPortPtrs(block,x)*
- *Getint32OutPortPtrs(block,x)*
- *Getuint8OutPortPtrs(block,x)*
- *Getuint16OutPortPtrs(block,x)*
- *Getuint32OutPortPtrs(block,x)*

Kde jako parametr předáváme funkci opět instanci bloku a číslo výstupu a návratová hodnota je ukazatel na pole odpovídající typu proměnné (Obr. 15).

Parametry

Funkci lze předávat tři typy parametrů - celočíselné, reálné a objekty (ve formě vektorů apod.). Lze k nim přistupovat přímo ve struktuře bloku přes volání jednotlivých indexovaných polí *block→ipar [], block→rpar [], block→opar []*. Lepší je ovšem využít

přístup pomocí funkcí hlavičkového souboru *scicos/scicos_block4.h*:

- *GetRparPtrs(block)*
- *GetIparPtrs(block)*

Kde jako parametr předáváme funkci opět instanci bloku a návratová hodnota je ukazatel na pole odpovídající typu proměnné parametru.

- *GetRealOparPtrs(block,x)*

Pro objekty předáváme funkci kromě instance bloku i index objektu v rámci vstupního pole, návratová hodnota je pak ukazatel na pole parametru typu double.

Příznak flag

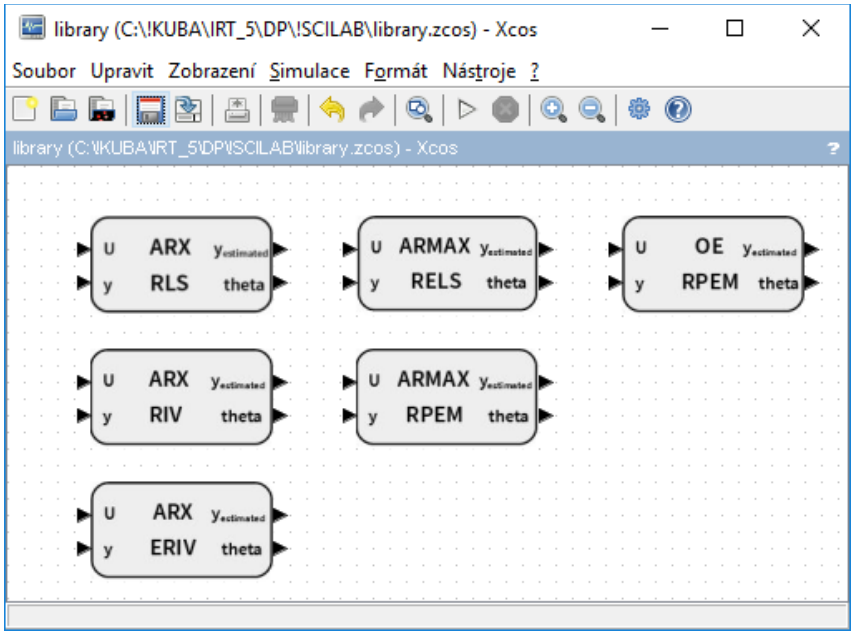
Prostředí Xcos spouští každou funkci v různých fázích simulace s různými příznaky, které vnitřně fungují jako přepínač typu switch. Jednotlivé fáze volání funkce jsou:

- *Flag 4 - inicializace*. Slouží k jednorázovému importu parametrů, počátečnímu nastavení vstupů a výstupů.
- *Flag 6 - inicializace, pevně dané výpočty*. Volání výpočetních funkcí během simulace.
- *Flag 1 - výpočet výstupů*. Volání pro nastavení výstupů v každém kroku simulace.
- *Flag 5 – Ukončení funkce*. Poslední volání funkce před ukončením simulace, Může být využito pro uvolnění paměti, ošetření chyb apod.

Kromě těchto základních příznaků obsahuje Xcos řadu dalších zejména pro speciální výpočetní funkce. Např. výpočty diskrétních přenosů ve stavovém prostoru (flag 2), výpočty spojitých přenosů ve stavovém prostoru (flag 0), ošetření pro dělení nulou (flag 9), výpočet Jacobiánu (flaf 10), nastavení spojitých stavových proměnných (flag 7) apod.

4.3 Realizace knihovny rekurzivních identifikačních algoritmů v prostředí Scilab Xcos

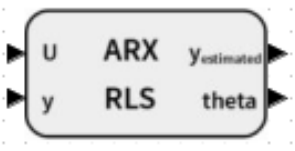
Knihovna rekurzivních identifikačních algoritmů je realizována pro modely a metody zmíněné v teoretické části práce. Konkrétně pro modely ARX, ARMAX a OE a pro rekurzivní metody nejmenších čtverců (RLS), rozšířenou metodu nejmenších čtverců (RELS), metodu instrumentální proměnné (RIV), rozšířenou metodu instrumentální proměnné (ERIV) a pro metodu predikčních chyb (RPEM) pro model ARMAX i OE. Celkem tedy knihovna obsahuje šest uživatelských bloků (Obr. 17).



Obr. 17: Knihovna rekurzivních identifikačních algoritmů v prostředí Scilab Xcos

Blok knihovny

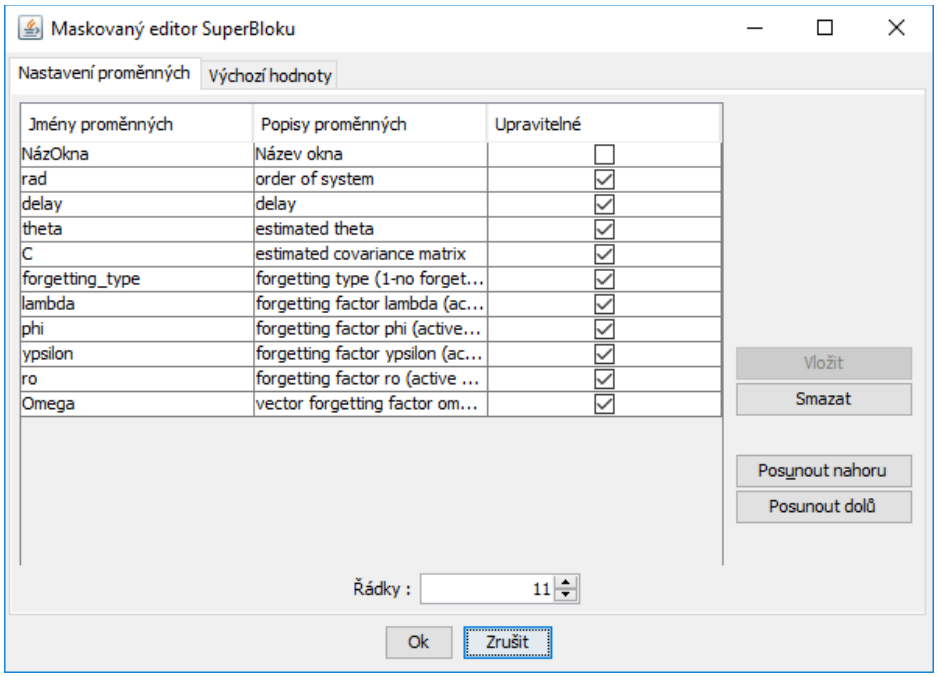
Jednotlivé bloky obsahují vždy dva vstupy a dva výstupy. Vstupem je logicky vstup do systému, který chceme identifikovat a výstup z takto identifikovaného systému. Perioda vzorkování je přejata z identifikovaného systému zprostředkovaně. Výstupem bloku je odhadovaný výstup a vektor identifikovaných parametrů systému (Obr. 18).



Obr. 18: Blok knihovny

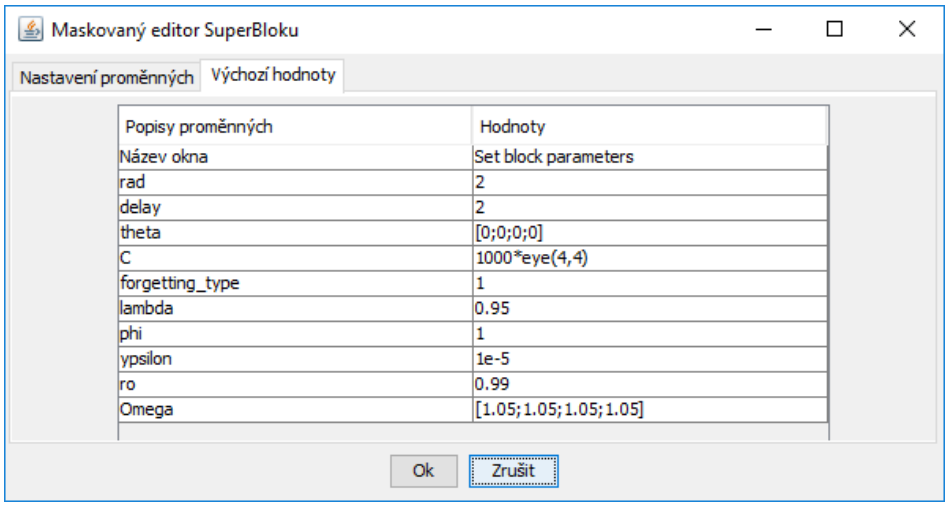
Maska bloku

Bloky knihovny jsou zamaskovány uživatelskou maskou pomocí uživatelsky definovaného bloku *SUPER_f*. Tato maska umožňuje zadání proměnných nezbytných pro identifikaci systému a volbu způsobu a parametrů zapomínání. Při tvorbě masky lze zadat zobrazované popisy proměnných, jejich vnitřní názvy a atribut editovatelnosti pole (Obr. 19)



Obr. 19: Tvorba masky bloku

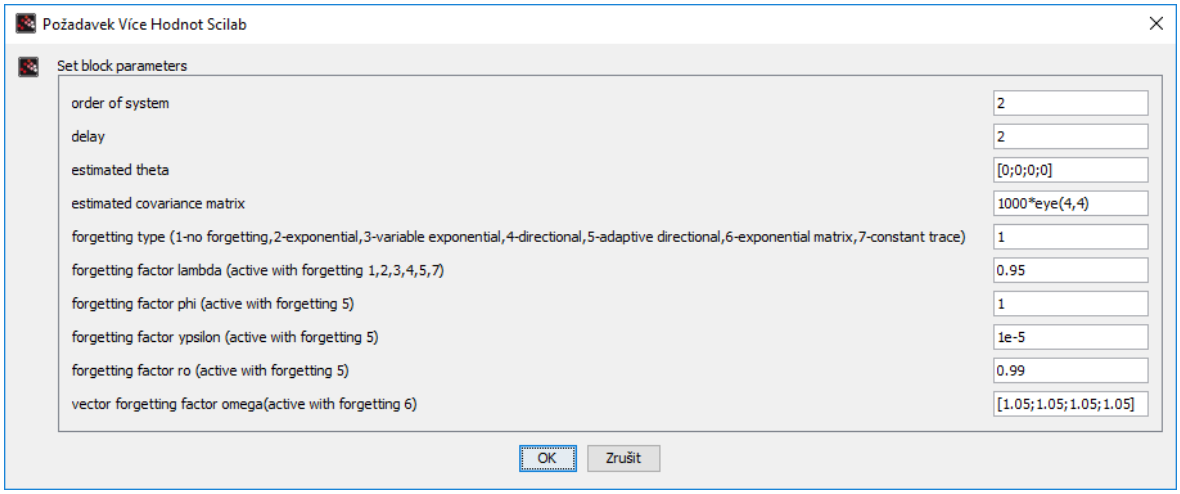
Pro jednotlivé proměnné lze zadat počáteční zobrazované hodnoty (Obr. 20).



Obr. 20: Zadání počátečních hodnot masky bloku

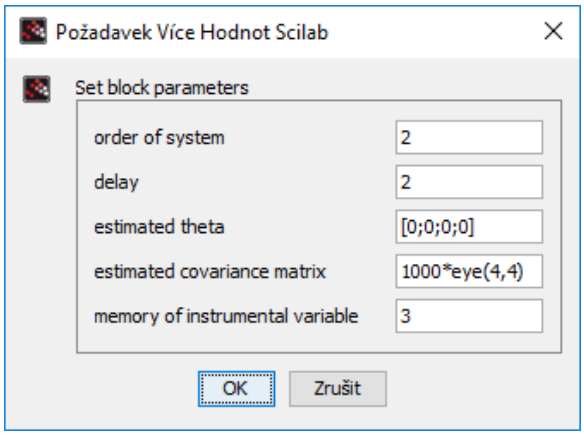
Výsledná maska má podobu podle obrázku (Obr. 21). Jednotlivými parametry jsou:

- řád systému (*order of system*)
- zpoždění (*delay*)
- odhad parametrů vektoru theta (*estimated theta*)
- odhad parametrů kovarianční matice (*estimated covariance matrix*)
- typ zapomínání (*forgetting type*)
- jednotlivé faktory zapomínání, které jsou aktivní pro definované typy zapomínání



Obr. 21: Maska bloku

Výjimku tvoří maska pro blok metody rozšířené instrumentální proměnné (ERIV), pro kterou není definováno zapomínání, ale pouze hloubka paměti rozšířeného vektoru instrumentální proměnné (Obr. 22).



Obr. 22: Maska bloku ERIV

Výpočetní C funkce bloku

Samotný výpočet rekurzivní identifikace je realizován ve vnitřním bloku v jazyce ANSI C. Zdrojový kód pro základní metodu nejmenších čtverců (RLS) je přílohou této diplomové práce.

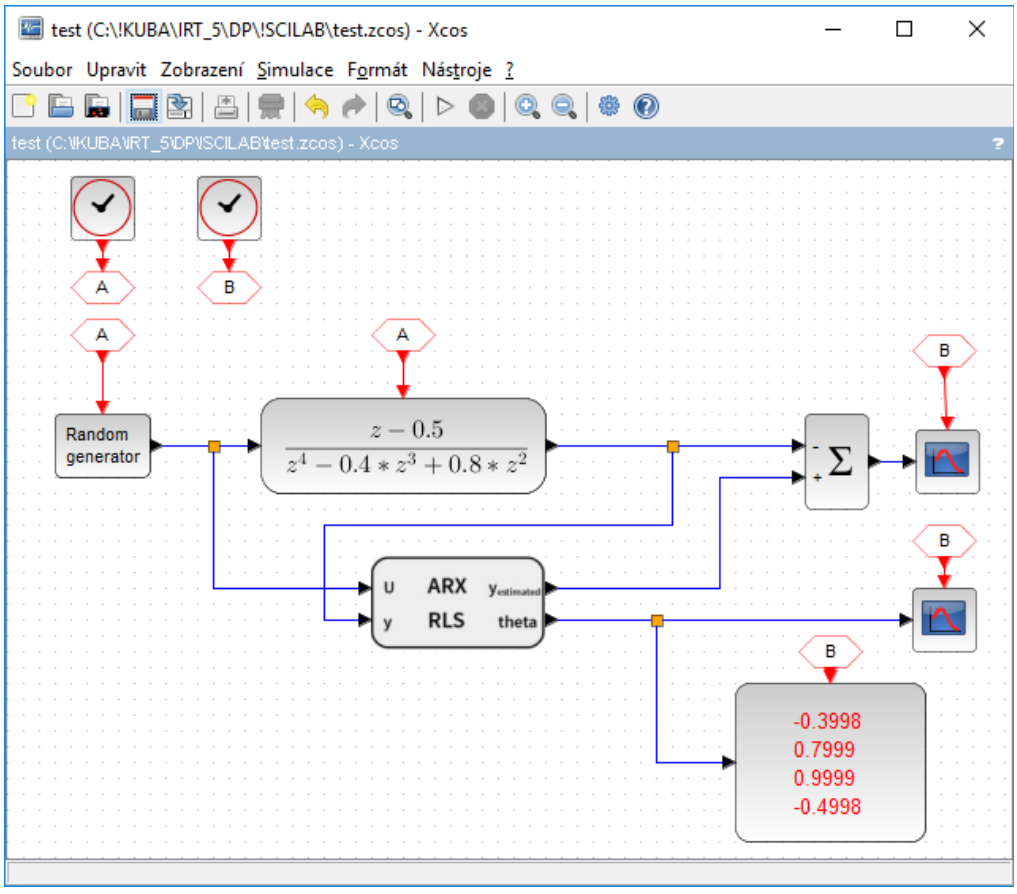
4.4 Příklady použití bloku knihovny

Identifikace systému pomocí rekurzivní metody nejmenších čtverců (RLS)

Identifikujeme následující systém druhého řádu s dopravním zpožděním definovaný přenosem:

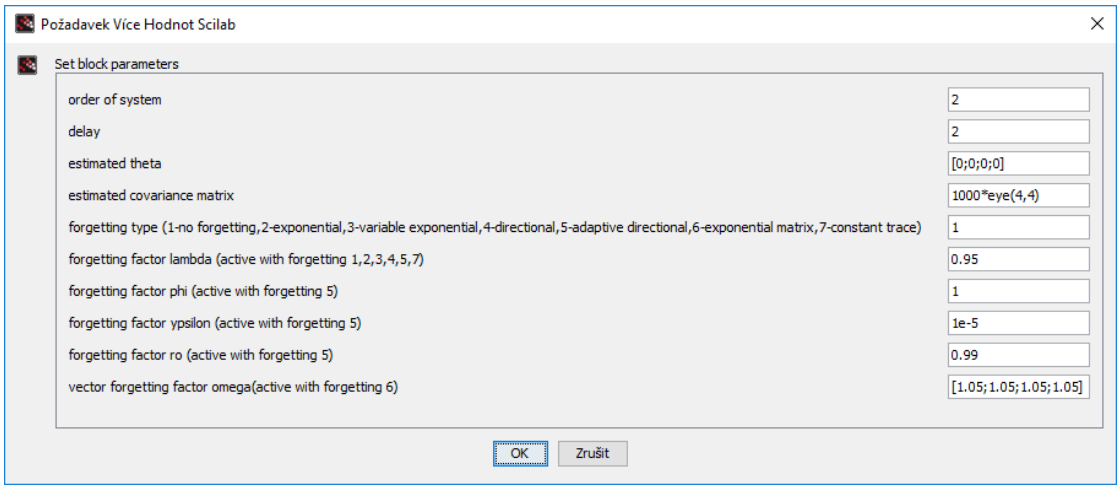
$$G(z)=\frac{z-0{,}5}{z^4-0{,}4\,z^3+0{,}8\,z^2}$$

Zadaný přenos zapojíme s blokem knihovny podle následujícího schematu (Obr. 23).



Obr. 23: Příklad zapojení bloku knihovny (RLS)

Pro blok nastavíme uživatelské parametry v masce (Obr. 24).



Obr. 24: Nastavení masky bloku

Po výpočtu simulace dostaneme vykreslený průběh identifikace (Obr. 25).

Obr. 25: Průběh identifikace systému (RLS)

Identifikace systému pomocí rekurzivní metody nejmenších čtverců (RLS) s využitím zapomínání pomocí konstantní stopy (constant trace)

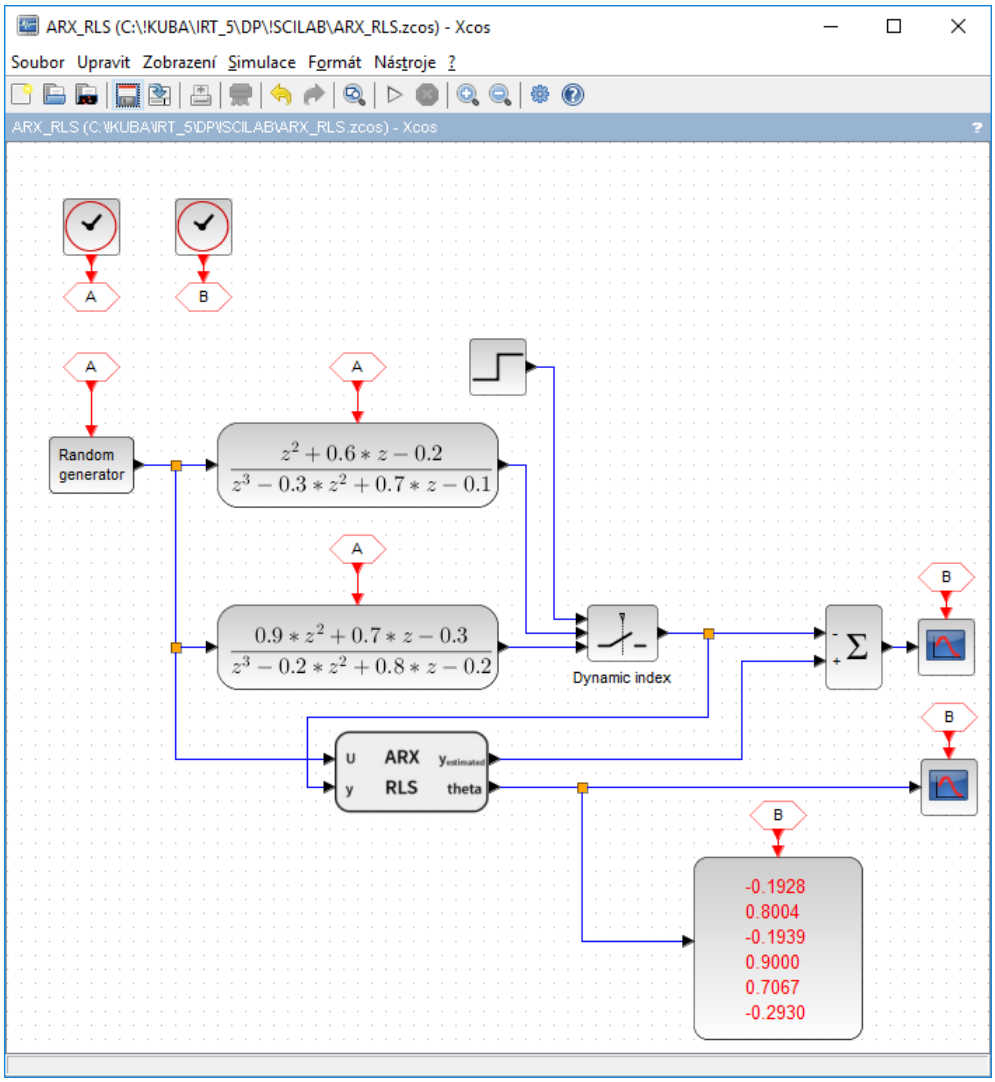
Identifikujeme systém třetího řádu s proměnlivými parametry:

$$G(z)=\frac{z^2+0{,}6\,z-0{,}2}{z^3-0{,}3\,z^2+0{,}7\,z-0{,}1}$$

, který v čase simulace t=10 změní parametry na:

$$G(z)=\frac{0{,}9\,z^2+0{,}7\,z-0{,}3}{z^3-0{,}2\,z^2+0{,}8\,z-0{,}2}$$

Pro identifikaci využijeme modifikaci se zapomínáním pomocím konstantní stopy. Identifikovaný systém zapojíme s blokem knihovny podle schematu (Obr. 26).



Obr. 26: Příklad zapojení bloku knihovny (RLS-constant trace)

Po výpočtu simulace dostaneme vykreslený průběh identifikace (Obr. 27).

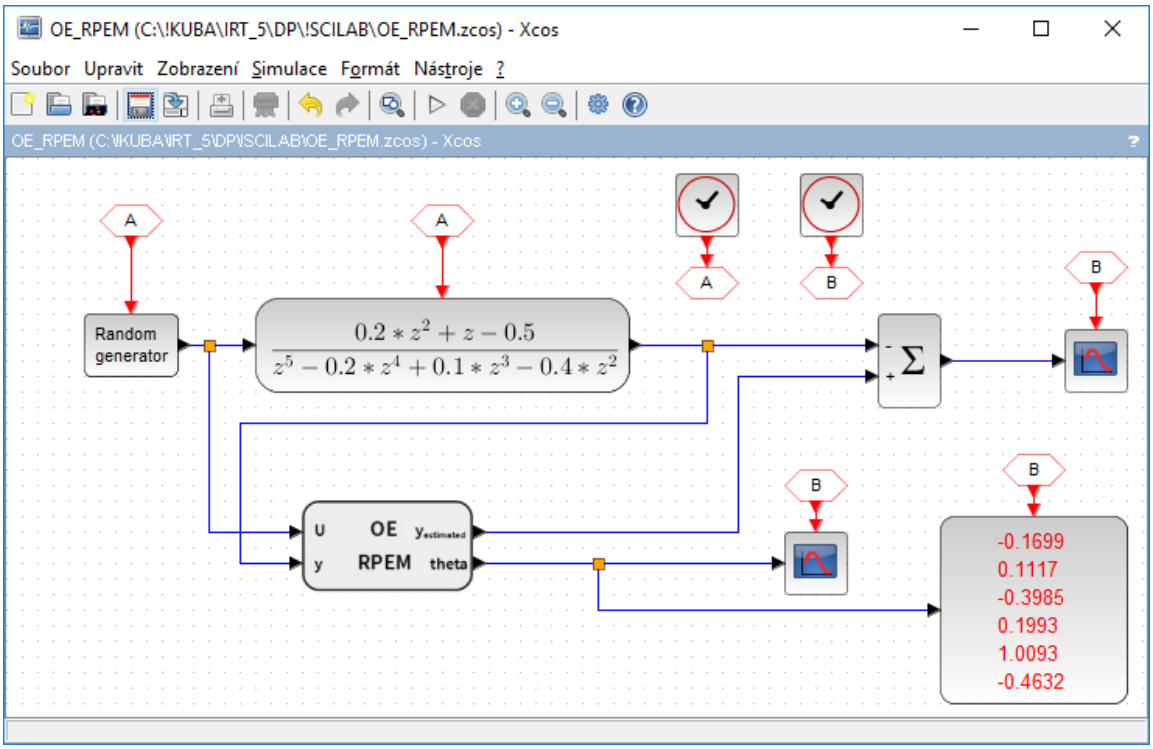
Obr. 27: Průběh identifikace systému (RLS-constant trace)

Identifikace systému pomocí metody predikčních chyb pro model OE (prediktor)

Identifikujeme následující systém třetího řádu s dopravním zpožděním definovaný přenosem:

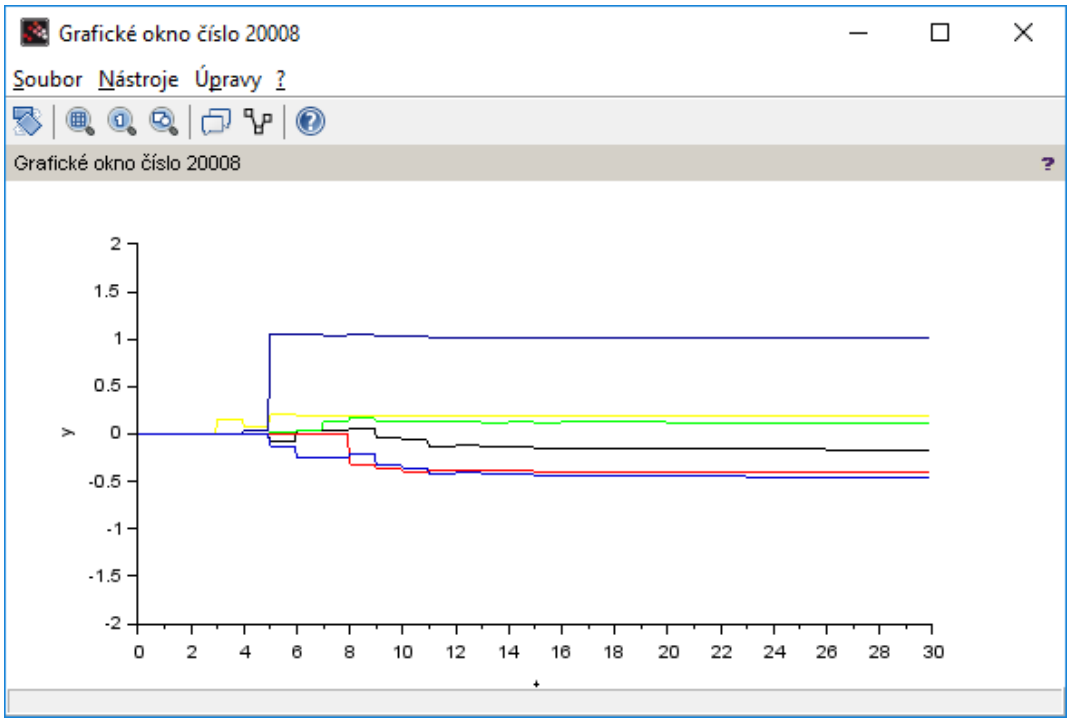
$$G(z)=\frac{0{,}2\,z^2+z-0{,}5}{z^5-0{,}2\,z^4+0{,}1\,z^3-0{,}4\,z^2}$$

Zadaný přenos zapojíme s blokem knihovny podle následujícího schematu (Obr. 28).



Obr. 28: Příklad zapojení bloku knihovny (RPEM)

Po výpočtu simulace dostaneme vykreslený průběh identifikace (Obr. 29).



Obr. 29: Průběh identifikace systému (RPEM)

ZÁVĚR

Tato diplomová práce si klade za cíl vytvoření knihovny rekurzivních identifikačních algoritmů v prostředí programu Scilab Xcos. Tvorba této knihovny je popsána v praktické části práce, která navazuje na poznatky z teoretické části.

V teoretické části práce jsou shrnuty základní přístupy pro modelování dynamických systémů včetně kriterií kvality takto vytvořených modelů. Dále jsou popsány jednotlivé metody rekurzivní identifikace systémů včetně rozšíření těchto metod o jednotlivé typy zapomínání.

V praktické části je popsán způsob tvorby vlastních bloků knihovny, které jsou realizovány pomocí uživatelsky definovaného bloku. Vlastní knihovna se skládá ze šesti bloků pro jednotlivé typy modelů a metody identifikace. Vstupem bloku je vždy vstup a výstup systému, výstupem bloku je odhadovaný výstup a odhadované parametry systému. Parametry bloku včetně jednotlivých způsobů zapomínání lze zadávat v uživatelsky definované masce.

Pro knihovnu je zpracována jednoduchá nápověda ve formě html stránek. Kromě základního popisu jednotlivých modelů a identifikačních metod jsou zde uvedeny schemata jednotlivých zapojení a jejich použití.

Takto vytvořená a popsaná knihovna může najít využití v samočinně se nastavujících regulátorech, při sledování průběhu dynamicky se měnících procesů, případně pro studijní účely pro studenty v oborech zabývajících se popisem a řízením systémů. V neposlední řadě rozšiřuje tato knihovna komunitu kolem programu Scilab, který získává právě díky grafické nadstavbě Xcos a bezplatné licenční politice čím dál více na oblibě. I proto podobné projekty značně rozšiřují řady uživatelů.

Knihovna je včetně html nápovědy v angličitně sdílena přes oficiální distribuční kanál programu Scilab na: https://fileexchange.scilab.org/.

SEZNAM POUŽITÉ LITERATURY

[1] Bobál, Vladimír. Identifikace systémů. Zlín: Univerzita Tomáše Bati ve Zlíně, 2009. ISBN 9788073188887

[2] CAMPBELL, S. L., CHANCELIER Jean-Philippe a NIKOUKHAH Ramine. Modeling and simulation in Scilab/Scicos. New York: Springer Science+Business Media, 2006. ISBN 9780387278025

[3] FIKAR, Miroslav a MIKLEŠ, Ján. Identifikácia systemov. Bratislava: STU, 1999. ISBN 8071947768

[4] Friedlander, B. The overdetermined recursive instrumental variable method. Systems Control Technology. 1984, s. 353-356

[5] KULHAVÝ, Rudolf. Restricted exponential forgettingin real time identification. Automatica. 1987, s. 589-600

[6] LJUNG, Lennart. System identification: theory for the user. New York: Prentice-Hall, 1987. ISBN 0138816409

[7] MOORE, J.B. a BOEL, R.K. Adaptation tracking in system identication. Automatica. 1986, s. 237-240

[8] NELLES, Oliver. Nonlinear system identification: from classical approaches to neural networks and fuzzy models. New York: Springer, 2001. ISBN 3540673695

[9] Noskievič, Petr. Modelování a identifikace systémů. Ostrava: MONTANEX, 1999. ISBN 8072250302

[10] SALGADO, M. E., GOODWIN, G. C. a MIDDLETON, R.H. Exponential Forgetting and Resetting. International Journal of Control. 1988, s. 477

[11] SANDEEP, Nagar. Introduction to Scilab: For Engineers and Scientists. New York: Apress, 2017. ISBN 9781484231913

[12] SÖDERSTRÖM, Torsten. a Petre. STOICA. System identification. New York: Prentice Hall, 1989. ISBN 0138812365

[13] TEJAS, B. Sheth. Scilab: A Practical Introduction to Programming and Problem Solving. Gujarat: CreateSpace Independent Publishing Platform, 2016. ISBN 9781539027843

[14] WELLSTEAD, P. E. a ZARROP Martin. Self-tuning systems: control and signal processing. New York: Wiley, 1991. ISBN 0471928836

SEZNAM OBRÁZKŮ

Obr. 1: Blokové schéma modelu obecného stochastického procesu....12
Obr. 2: Blokové schéma ARX modelu....15
Obr. 3: Blokové schéma ARMAX modelu....15
Obr. 4: Blokové schéma OE modelu....16
Obr. 5: Blokové schema-chyba vstupu....17
Obr. 6: Blokové schema-chyba výstupu....18
Obr. 7: Blokové schema-chyba rovnice....18
Obr. 8: Blokové schema ARMAX modelu pro metodu RELS....28
Obr. 9: Editační okno Xcos....41
Obr. 10: Průzkumník palet Xcos....42
Obr. 11: Nastavení parametrů simulace pro Xcos....42
Obr. 12: Paleta uživatelsky definovaných funkcí Xcos....43
Obr. 13: Datová struktura obecného bloku....44
Obr. 14: Maska bloku C funkce....45
Obr. 15: Typy proměnných C bloku....45
Obr. 16: Načtení vstupů funkce....46
Obr. 17: Knihovna rekurzivních identifikačních algoritmů v prostředí Scilab Xcos....49
Obr. 18: Blok knihovny....49
Obr. 19: Tvorba masky bloku....50
Obr. 20: Zadání počátečních hodnot masky bloku....50
Obr. 21: Maska bloku....51
Obr. 22: Maska bloku ERIV....51
Obr. 23: Příklad zapojení bloku knihovny (RLS)....52
Obr. 24: Nastavení masky bloku....53
Obr. 25: Průběh identifikace systému (RLS)....53
Obr. 26: Příklad zapojení bloku knihovny (RLS-constant trace)....54
Obr. 27: Průběh identifikace systému (RLS-constant trace)....55
Obr. 28: Příklad zapojení bloku knihovny (RPEM)....56
Obr. 29: Průběh identifikace systému (RPEM)....56

PŘÍLOHA P 1: ZDROJOVÝ KÓD BLOKU RLS

```
#include "scicos_block4.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

struct matrix {
    double data[20][20];
    int rows;
    int columns;
};

struct matrix build_matrix(double matrix_in[],int rows_in,int columns_in){
         struct matrix m;
         m.rows=rows_in;
         m.columns=columns_in;
         for(int i=0;i<columns_in;i++){
                  for(int j=0;j<rows_in;j++){
                                   m.data [j] [i]=matrix_in [j+(i*rows_in)];
                           }
         }
         return m;
}

struct matrix build_diagonal_matrix(double matrix_in[],int rows_in,int columns_in){
         struct matrix m;
         m.rows=rows_in;
         m.columns=columns_in;
         for(int i=0;i<columns_in;i++){
                  for(int j=0;j<rows_in;j++){
                                   if(j==i){
                                            m.data [j] [i]=1/sqrt(matrix_in [i]);
                                   }else{
                                            m.data [j] [i]=0;
                                   }

                           }
         }
         return m;
}

void send_matrix(struct matrix m,double matrix_out[]){
         for(int i=0;i<m.columns;i++){
                  for(int j=0;j<m.rows;j++){
                            matrix_out [j+(i*m.rows)]=m.data [j] [i];
                  }
         }
}

struct matrix sum_matrix(struct matrix m_1,struct matrix m_2){
         struct matrix m;
         m.rows=m_1.rows;
         m.columns=m_2.columns;
         for(int j=0;j<m.rows;j++){
                           for(int i=0;i<m.columns;i++){
                                   m.data [j] [i]=m_1.data [j] [i] + m_2.data [j] [i];
                           }
         }
         return m;
}

struct matrix difference_matrix(struct matrix m_1,struct matrix m_2){
         struct matrix m;
         m.rows=m_1.rows;
         m.columns=m_2.columns;
         for(int j=0;j<m.rows;j++){
                           for(int i=0;i<m.columns;i++){
                                   m.data [j] [i]=m_1.data [j] [i] - m_2.data [j] [i];
                           }
         }
         return m;
```

```
}

struct matrix multiply_matrix(struct matrix m_1,struct matrix m_2){
         struct matrix m;
         m.rows=m_1.rows;
         m.columns=m_2.columns;
         for(int i=0; i<m.rows; i++){
        for(int j=0; j<m.columns; j++){
                 m.data[i][j]=0;
        }
         }

         for(int i=0; i<m.rows; i++){
                for(int j=0; j<m.columns; j++){
                    for(int k=0; k<m_1.columns; k++)
                    {
                        m.data[i][j]=m.data[i][j]+m_1.data[i][k]*m_2.data[k][j];
                    }
                         }
         }
         return m;
}

struct matrix transpose_matrix(struct matrix m_1){
         struct matrix m;
         m.rows=m_1.columns;
         m.columns=m_1.rows;
         for(int j=0;j<m.rows;j++){
                          for(int i=0;i<m.columns;i++){
                                  m.data [j] [i]=m_1.data [i] [j] ;
                          }
         }
         return m;
}

struct matrix div_const_matrix(struct matrix m,double c){
         for(int j=0;j<m.rows;j++){
                          for(int i=0;i<m.columns;i++){
                                  m.data [j] [i]=(m.data [j] [i]/c);
                          }
         }
         return m;
}

struct matrix build_eye_diagonal_matrix(int rows_in,int columns_in){
         struct matrix m;
         m.rows=rows_in;
         m.columns=columns_in;
         for(int i=0;i<columns_in;i++){
                  for(int j=0;j<rows_in;j++){
                                  if(j==i){
                                          m.data [j] [i]=1;
                                  }else{
                                          m.data [j] [i]=0;
                                  }

                          }
         }
         return m;
}

double trace_matrix(struct matrix m){
         double c=0;
         for(int j=0;j<m.rows;j++){
                  for(int i=0;i<m.columns;i++){
                          if(j==i){
                                  c=c+m.data [j] [i];
                          }
                  }
         }
         return c;
}

double return_first(struct matrix m){
```

```
        double c;
        c=m.data[0][0];
        return c;
}

SCSREAL_COP *in1;
SCSREAL_COP *in2;
SCSREAL_COP *opar1;
SCSREAL_COP *opar2;
SCSREAL_COP *opar3;
SCSREAL_COP *out1;
SCSREAL_COP *out2;
SCSINT32_COP *ipar;
SCSREAL_COP *rpar;
int rad;
int delay;
int forgetting_type;
struct matrix fi;
struct matrix C;
struct matrix theta;
struct matrix Omega;
struct matrix Lambda;
struct matrix e_k;
struct matrix y_k;
struct matrix y_est;
double y [10];
double u [30];
double fi_array [20];
/* forgetting parameters */
double lambda;
double lambda_0=0.95;
double epsilon;
double ksi=0;
double phi=1;
double eta;
double ypsilon=0.00001;
double ro=0.99;
double e_2;

void ARX_RLS(scicos_block *block,int flag)
{
  /* init */
  if (flag==4) {
        ipar=GetIparPtrs(block);
        rpar=GetRparPtrs(block);
        opar1=GetRealOparPtrs(block,1);
        opar2=GetRealOparPtrs(block,2);
        opar3=GetRealOparPtrs(block,3);
        rad=ipar[0];
        delay=ipar[1];
        forgetting_type=ipar[2];
        lambda=rpar[0];
        ro=rpar[1];
        ypsilon=rpar[2];
        phi=rpar[3];
        C=build_matrix(opar1,rad*2,rad*2);
        theta=build_matrix(opar2,rad*2,1);
        Omega=build_diagonal_matrix(opar3,rad*2,rad*2);
        /* alloc u vector */
        for(int i=0;i<(rad+delay);i++){
                u[i]=0;
        }
        /* alloc y vector */
        for(int i=0;i<rad;i++){
                y[i]=0;
        }

  /* output computation */
  } else if(flag==1) {
/* each-step init */
        in1=GetRealInPortPtrs(block,1);
        in2=GetRealInPortPtrs(block,2);
        y_k=build_matrix(in2,1,1);
```

```
/* join u,y to fi vector */
        for(int i=0;i<2*rad;i++){
                if(i<rad){
                        fi_array [i]=-y[i];
                }else{
                        fi_array [i]=u[i-rad+delay];
                }
        }
        fi=build_matrix(fi_array,rad*2,1);

/* computing function */
        y_est=multiply_matrix(transpose_matrix(fi),theta);
        e_k=difference_matrix( y_k,y_est);

        if(forgetting_type==1){           //no forgetting
                C=difference_matrix(C,
(div_const_matrix(multiply_matrix(C,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),C))),1+retur
n_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi)))));
                theta=sum_matrix(theta,multiply_matrix(C,multiply_matrix(fi,e_k)));

        }else if(forgetting_type==2){     //exponential forgetting
                C=div_const_matrix(difference_matrix(C,
(div_const_matrix(multiply_matrix(C,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),C))),lambda+
return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi)))),lambda);
                theta=sum_matrix(theta,multiply_matrix(C,multiply_matrix(fi,e_k)));

        }else if(forgetting_type==3){     //variable exponential forgetting
                lambda=(lambda*lambda_0+1-lambda_0);
                C=div_const_matrix(difference_matrix(C,
(div_const_matrix(multiply_matrix(C,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),C))),lambda+
return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi)))),lambda);
                theta=sum_matrix(theta,multiply_matrix(C,multiply_matrix(fi,e_k)));

        }else if(forgetting_type==4){     //fixed directional forgetting
                if(return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi))==0){
                        epsilon=lambda;
                }else{
                        epsilon=lambda-((1-lambda)/
(return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi))));
                }
                C=difference_matrix(C,
(div_const_matrix(multiply_matrix(C,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),C))),
(1/epsilon)+return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi)))));
                theta=sum_matrix(theta,multiply_matrix(C,multiply_matrix(fi,e_k)));

        }else if(forgetting_type==5){     //adaptive directional forgetting
                ksi=return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi));
                if(return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi))==0){
                        epsilon=phi;
                }else{
                        epsilon=phi-((1-phi)/ksi);
                }
                theta=sum_matrix(theta,div_const_matrix(multiply_matrix(C,multiply_matrix(fi,e_k)),
(1+ksi)));
                C=difference_matrix(C,
(div_const_matrix(multiply_matrix(C,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),C))),
(1/epsilon)+return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi)))));
                e_2=return_first(e_k)*return_first(e_k);
                eta=e_2/lambda;
                if((1+ksi)>0){
                        phi=1/(1+((1+ro)*log(1+ksi))+((((ypsilon+1)*eta)/(1+ksi+eta))-1)*(ksi/
(1+ksi)));
                }
                lambda=phi*(lambda+((e_2)/(1+ksi)));
                ypsilon=phi*(ypsilon+1);
        }else if(forgetting_type==6){     //exponential forgetting matrix
                Lambda=multiply_matrix(multiply_matrix(Omega,C),Omega);
                C=difference_matrix(Lambda,
(div_const_matrix(multiply_matrix(Lambda,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),Lambda)
)),1+return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),Lambda),fi)))));
                theta=sum_matrix(theta,multiply_matrix(C,multiply_matrix(fi,e_k)));
        }else if(forgetting_type==7){     //constant trace
                ksi=return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi));
                theta=sum_matrix(theta,div_const_matrix(multiply_matrix(C,multiply_matrix(fi,e_k)),
(1+ksi)));
                C=div_const_matrix(difference_matrix(C,
(div_const_matrix(multiply_matrix(C,multiply_matrix(fi,multiply_matrix(transpose_matrix(fi),C))),lambda+
```

```
return_first(multiply_matrix(multiply_matrix(transpose_matrix(fi),C),fi))))),lambda);
                C=sum_matrix(div_const_matrix(C,(trace_matrix(C)/10000)),
build_eye_diagonal_matrix(rad*2,rad*2));
        }

/* output */
        out1=GetRealOutPortPtrs(block,1);
        send_matrix(y_est,out1);
        out2=GetRealOutPortPtrs(block,2);
        send_matrix(theta,out2);

/* shift u vector */
        for(int i=(rad+delay)-1;i>-1;i--){
                if (i!=0){
                        u[i]=u[i-1];
                }else if (i==0){
                        u[i]=(in1[0]);
                }
        }

/* shift y vector */
        for(int i=rad-1;i>-1;i--){
                if (i!=0){
                        y[i]=y[i-1];
                }else if (i==0){
                        y[i]=(in2[0]);
                }
        }

  /* ending */
  } else  if (flag == 5) {
  } else if(flag == 6) {
  /* ending */
  }
}
```