Identificarea Sistemelor 2018-2019

Proiect 2

Modelarea unui sistem dinamic folosind un **model ARX NELINIAR polinomial**

Student: Călina Corneliu-Dumitru

CUPRINS

1. Introducere

Modelul NARX polinomial este o generalizare a modelului ARX liniar în care ieșirea este formată din dependențe neliniare între ieșirile și intrările precedente ale sistemului.

Scopul proiectului este de a modela un **sistem dinamic** în care ieșirea poate fi afectată de zgomot folosind modelul de mai sus și de a-i verifica performanțele în funcție de ordinul sistemului (pentru simplitate se va alege ordinul maxim al sistemului = 3).

Proiectul va avea două concluzii:

1) Pentru PREDICȚIE – în care vom găsi o ieșire estimată celei reale folosind procedeul explicat în continuare.
2) Pentru SIMULARE – în care nu vom avea acces la ieșirea reală a sistemului, dar vom avea acces la model.

2. Prezentarea modelului

2.1. Model NARX polinomial. Aspecte generale.

Modelul NARX polinomial se realizează în mare parte la fel ca regresia liniară. Vom avea nevoie de vectorul de regresori care este format din dependențele neliniare dintre ieșiri și intrări și vectorul de parametri pentru a calcula un vector de ieșiri aproximativ egale cu ieșirile sistemului.

$$\begin{bmatrix} y(1) \\ . \\ . \\ . \\ y(n) \end{bmatrix} = \begin{bmatrix} \rho_1(1) & \dots & \rho_k(1) \\ & . & \\ & . & \\ & . & \\ \rho_1(n) & \dots & \rho_k(n) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \\ . \\ . \\ \theta_k \end{bmatrix}$$

Forma matriceală ($y=\Phi\cdot\theta$) de mai sus corespunde structurii modelului NARX polinomial:

$y(k)=g(y(k-1),\dots,y(k-na),u(k-1),\dots,u(k-nb))\cdot\theta + e(k)$ (1)

unde matricea Φ este polinomul g format din dependențele neliniare și fiecare element ρ_i are ca și corespondent un produs.

Având regresorii și ieșirile reale ale sistemului vom calcula vectorul de parametri θ folosind formula: $\theta=\Phi \setminus yreal$.

După ce avem matricile Φ și θ vom calcula $y = \Phi\cdot\theta$ care este o estimare a ieșirii reale.

(part2_getline.m va crea pe linie matricea Φ)

2.2. Crearea regresorilor.

Pentru a avea un model bun, trebuie luate în calcul următoarele probleme:

1) riscul supraantrenării modelului.
2) riscul duplicării unor termeni (în funcție de metoda folosită pentru găsirea regresorilor).

Spre înțelegerea acestor probleme vom aduce un exemplu folosind ordinele sistemului na=1, nb=1, gradul polinomului=3 și vom discuta mai târziu rezolvările acestora. Polinomul „g" folosit în ecuația (1) din pagina precedentă are următoarea formă:

g(y,u)=$y(k-1) + y^2(k-1) + y^3(k-1) + u(k-1) + u^2(k-1) + u^3(k-1) + y(k-1) \cdot u(k-1) + y(k-1) \cdot u^2(k-1) + y^2(k-1) \cdot u(k-1)$ (2)

Am renunțat la scrierea argumentelor din motive de simplitate a citirii.

În matricea Φ, ρ(i) va fi g(y,u) pentru k=i.

2.3. Implementarea metodei.

Pentru generarea tuturor dependențelor vom folosi reprezentarea pe biți. Metoda constă în realizarea unei matrici de forma: (pe baza relației (2))

$$[\ y(k-1)\ u(k-1)\ y^2(k-1)\ u^2(k-1)\ y^3(k-1)\ u^3(k-1)\ ]$$

Fiecare combinare între ieșire și intrare trebuie să aibă gradul mai mic decât gradul polinomului impus (ex.: $y(k-1) \cdot u^3(k-1)$ nu e bun!). Vom elimina atunci ultimii na+nb termeni deoarece ei au gradul maxim și vom simplifica operațiile.

⇨ $[\ y(k-1)\ u(k-1)\ y^2(k-1)\ u^2(k-1)\ ]$

Odată realizată această matrice observăm că are lungimea $(na+nb) \cdot m$ și atunci vom reprezenta toate numerele de la 1 la $2^{(na+nb)\cdot m}$ pe biți în baza 2. Pentru a simplifica din nou unele operații și de a reduce timpul de execuție, vom lua doar acele numere care au suma biților de „1" >= 2 și <= m . Atunci, pentru acele reprezentări cu condiția de mai sus îndeplinită, vom înmulți termenii de pe indexul corespunzător bitului „1".

De ținut cont că ieșirile și intrările la momente negative sau 0 le vom inițializa cu 0. Această inițializare o vom face în MATLAB cu comanda ***zeros(a,b)*** unde a=numărul liniilor și b=numărul coloanelor. În final aceste valori le vom concatena vectorului nostru inițial.

(part2_getvalues.m, part2_getbits.m și part2_multip.m vor face aceste operații – vezi anexa)

2.4. Rezolvarea posibilelor problemelor.

Problemele discutate în subcapitolul (2.2):

1) Să evităm să luăm ordinele sistemului și gradul polinomului foarte mari chiar dacă eroarea pe datele de identificare e foarte mică deoarece riscăm să aproximăm de fapt și zgomotul din date și atunci vom avea un model compromis.

2) Având informațiile de mai sus, vom lua un caz care prezintă problema (pe baza exemplului oferit):

[y(k-1) u(k-1) y^2(k-1) u^2(k-1)]

și reprezentarea pe biți :

[1 0 1 0]

Am spus că metoda constă în înmulțirea valorilor corespunzătoare biților de „1”. În cazul nostru se vor înmulți y(k-1) · y^2(k-1) ceea ce duce la o duplicare a termenilor deoarece noi avem deja un y^3(k-1) și atunci tot ce va trebui să facem este să punem o condiție de verificare: dacă între oricare ar fi 2 biți de „1”, distanța dintre indecșii corespunzători lor este un multiplu de (na+nb) atunci nu va fi luat în considerare produsul.

(part2_bool_check.m verifică această condiție – vezi anexa)

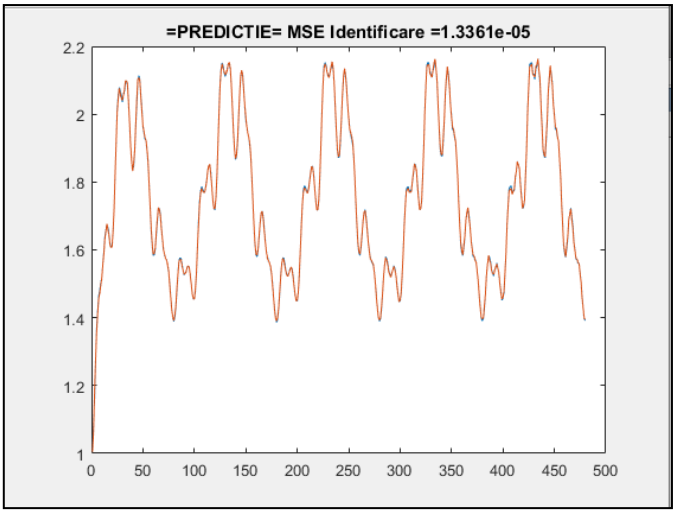
3. Rezultate obținute. Discuții.

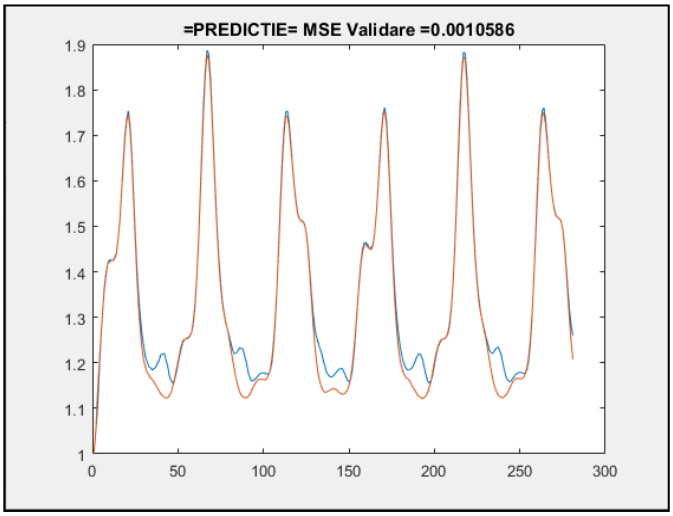
3.1. Predicție.

3.1.1. Aspecte generale.

Predicția este găsirea modelului folosind ieșirea sistemului. Odată cu găsirea acestui model, se calculează o ieșire estimată yestimat = Φ·θ . Pentru a vedea cât de bun este modelul, vom folosi MSE (metoda erorii pătratice) pentru a determina θ optim ținând cont și de problema discutată anterior (supraantrenarea). Vom executa codul MATLAB pentru toate combinările posibile dintre na,nb,nk și vom vedea pentru care valori ale acestor ordine θ pe datele de VALIDARE va fi minim.

3.1.2. Rezultate.





Pentru ordinele na=1, nb=1 și gradul polinomului 3 s-a obținut cea mai mică eroare pe datele de validare egală cu 0.0010586.

3.1.3. Tabel de erori.

na	nb	m	MSEid	MSEval
1	1	1	0.002938	0.004374
1	1	2	1.47E-05	0.002871
1	1	3	1.34E-05	0.001059
1	2	1	0.002932	0.004311
1	2	2	1.01E-05	0.002925
1	2	3	8.45E-06	0.001987
1	3	1	0.002885	0.004127
1	3	2	7.67E-06	0.003116
1	3	3	5.58E-06	0.004971
2	1	1	0.002721	0.004195
2	1	2	1.01E-05	0.002671
2	1	3	8.59E-06	0.082999
2	2	1	0.002689	0.004215
2	2	2	8.64E-06	0.008974
2	2	3	5.98E-06	0.055814
2	3	1	0.002637	0.004383
2	3	2	6.72E-06	0.005353
2	3	3	3.90E-06	1.017735
3	1	1	0.002599	0.004625
3	1	2	7.23E-06	0.002315
3	1	3	5.51E-06	0.040116
3	2	1	0.002597	0.004603
3	2	2	6.55E-06	0.005707
3	2	3	4.08E-06	0.51161
3	3	1	0.002591	0.004652
3	3	2	6.07E-06	0.013119
3	3	3	3.61E-06	11.88114

3.2. Simulare.

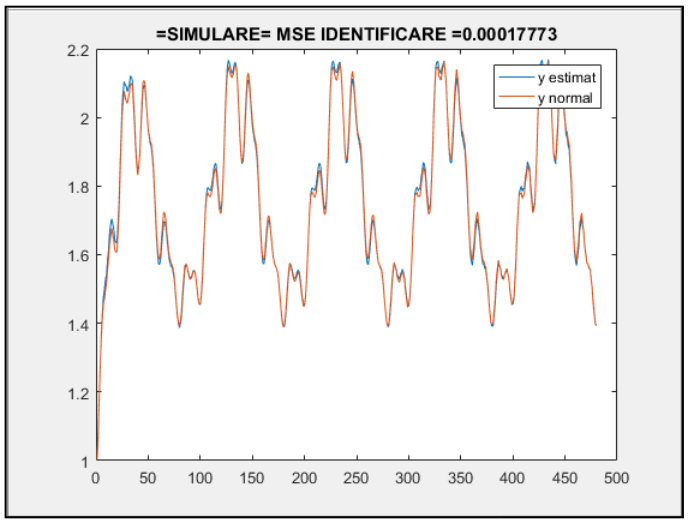
3.2.1. Aspecte generale.

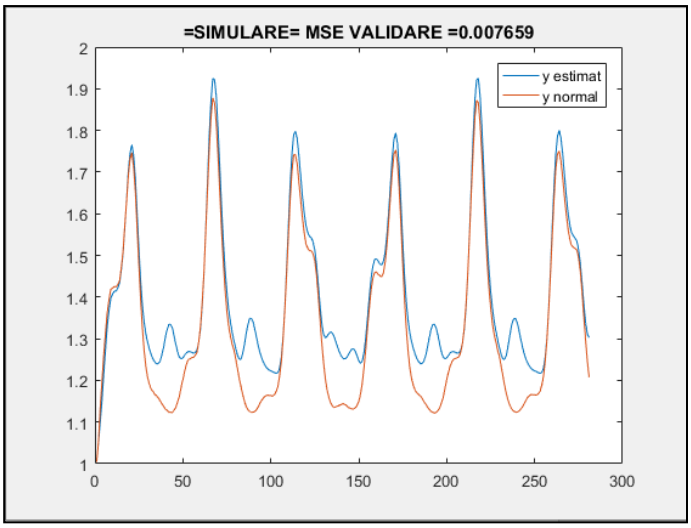
În simulare nu sunt cunoscute ieșirile precedente ale sistemului și atunci din această cauză va trebui să creem ieșirea sistemului ***recursiv*** pornind de la un vector inițializat cu 0 (notație: y_est_simulare pentru MATLAB). Vom lua vectorul de parametri ca fiind cel mai bun găsit pe predicție, iar la fiecare pas când creem linia de regresori, vom înmulți acea linie cu vectorul de parametri pentru a găsi y_est_simulare la pasul curent și vom actualiza întreg vectorul pentru următoarele iterații. Ordinele sistemului se vor lua cele pentru care am găsit cel mai bun θ.

```
y_est_simulare=zeros(length(yid),1);
for i=1:length(yid)
    L=part2_getline(y_est_simulare,uid,na_optim,nb_optim,i,m_optim)
    y_est_simulare(i)=L*theta_optim;
    PSI=[PSI;L];
end
```

În final calculăm MSE-ul dintre y_est_simulare și ieșirea sistemului (din date).

3.2.2. Rezultate.





4. Anexă cod MATLAB

part2_solution.m :

```
clc;
clear all;
close all;
load('iddata-04.mat');
uid=id.u;
yid=id.y;
uval=val.u;
yval=val.y;
na=3;
nb=3;
PSI=[];
m=3;
L=[];
min=9999;
MSEid=[];
MSEval=[];
MSEidentificareEXCEL=[];
MSEvalidareEXCEL=[];

% ==================PREDICTIE===============
% ==================IDENTIFICARE============
for x=1:na
    for j=1:nb
        for k=1:m
            PSI=[];
            PSIval=[];
            for i=1:length(yid)
                L=part2_getline(yid,uid,x,j,i,k);
                PSI=[PSI;L];
            end
            theta=PSI\yid;
            y_est=PSI*theta;
            e=(yid-y_est).^2;
            MSEid=1/length(e)*sum(e);
            MSEidentificareEXCEL=[MSEidentificareEXCEL, MSEid];


            % ==================VALIDARE================
            PSIval=[];
            Lval=[];
            for i=1:length(yval)
                Lval=part2_getline(yval,uval,x,j,i,k);
                PSIval=[PSIval;Lval];
            end
            y_estval=PSIval*theta;
            eval=(yval-y_estval).^2;
```

```
            MSEval=1/length(eval)*sum(eval);
            MSEvalidareEXCEL=[MSEvalidareEXCEL,MSEval];

            if MSEval<min
                na_optim=x;
                nb_optim=j;
                m_optim=k;
                min=MSEval;
                theta_optim=theta;
                yid_estimat=y_est;
                yval_estimat=y_estval;
                MSEvalidare=MSEval;
                MSEidentificare=MSEid;

            end
        end
    end
end
plot([yid_estimat,yid])
title(['=PREDICTIE= MSE Identificare =',num2str(MSEidentificare)]);
figure;
plot([yval_estimat,yval]);
title(['=PREDICTIE= MSE Validare =',num2str(MSEvalidare)]);

%================SIMULARE==================
% ================IDENTIFICARE============
PSI=[];
y_est_simulare=zeros(length(yid),1);
for i=1:length(yid)
    L=part2_getline(y_est_simulare,uid,na_optim,nb_optim,i,m_optim);
    y_est_simulare(i)=L*theta_optim;
    PSI=[PSI;L];
end


e=(yid-y_est_simulare).^2;
MSE=1/length(e)*sum(e);

figure
plot([y_est_simulare,yid]);
legend('y estimat','y normal');
title(['=SIMULARE= MSE IDENTIFICARE =',num2str(MSE)]);

% ==================VALIDARE====================
PSI=[];
y_est_simulare=zeros(length(yval),1);
for i=1:length(yval)
    L=part2_getline(y_est_simulare,uval,na_optim,nb_optim,i,m_optim);
    y_est_simulare(i)=L*theta_optim;
    PSI=[PSI;L];
end
eval=(yval-y_est_simulare).^2;
MSEval=1/length(eval)*sum(eval);
figure;
plot([y_est_simulare,yval]);
legend('y estimat','y normal');
title(['=SIMULARE= MSE VALIDARE =',num2str(MSEval)]);
```

part2_getline.m:

```
function [L] = part2_getline(yid,uid,na,nb,k,m)
    yid=[zeros(1,na),yid'];
    uid=[zeros(1,nb),uid'];
    values=[];

    %Matricea valorilor de tipul:
    %[y u y^2 u^2 ... y^m u^m]
    %unde y=[y(k-1)..y(k-na)]
    %si u=[u(k-1)..u(k-nb)]
    values=[yid(k+na-1:-1:k),uid(k+nb-1:-1:k)];
    auxmatrix=values;
    for i=2:m
        values=[values, auxmatrix.^i];
    end

    %Matricea gradelor de tipul:
    %[1 1 1 2 2 2 ... m m m] -> exemplu pentru na+nb=3;
    %-> 1 corespunde elementelor y,u
    %-> 2 corespunde elementelor y^2, u^2 s.a.m.d
    gr=ones(1,(na+nb)*m);
    for i=1:m-1
        gr((i*(na+nb)+1):(i+1)*(na+nb))=i+1;
    end

    %vom lua doar valorile (m-1)*(na+nb) deoarece ultimele (na+nb) valori
    %vor avea deja gradul m si inmultite cu alti termeni vor avea gradul >m
     L=[1,values,part2_getvalues(values(1:(m-1)*(na+nb)),gr(1:(m-
1)*(na+nb)),na+nb)];
end
```

part2_getvalues.m:

```
function [ output] = part2_getvalues( values,grad,clones_check)
    output_=[];
    for i=1:2^length(values)-1
        %reprezentarea pe biti a fiecarui numar de la 1 la 2^length
        %functia returneaza reprezentarea si suma bitilor de "1"

        [sum,matrix]=part2_getbits(i,length(values));

        %nu are rost sa inmultim termenii cu suma mai mica decat 1
        %sau suma mai mare decat gradul maxim (=m)
        if sum > 1 && sum <= grad(end)+1
            produs=part2_multip(values,matrix,grad,clones_check);
            %conditie din part2_multip (vezi functia)
            if produs ~= 10^40
                output_=[output_, produs];
            end
        end
    end
```

```
     output=output_;
end
```

part2_getbits.m:

```
function [sum,output] = part2_getbits( number,bitsnumber)
    A=zeros(1,bitsnumber);
    sum_=0;
    A(1)=mod(number,2);
    if A(1)==1
        sum_=sum_+1;
    end
    k=1;
    while floor(number/2) ~= 0
        number=floor(number/2);
        A(k+1)=mod(number,2);
        if A(k+1)==1
            sum_=sum_+1;
        end
        k=k+1;
    end
    output=A;
    sum=sum_;

end
```

part2_bool_check.m:

```
function [ output_bool ] = part2_bool_check( vector , clone)
    output_=0;
    for i=1:length(vector)
        for j=1:length(vector)
            if mod(vector(i)-vector(j),clone)==0 && i~=j
                output_=1;
            end
        end
    end
    output_bool=output_;
end
```

part2_multip.m:

```
function [ output ] = part2_multip( values,matrix,grad,clones)
    sum=0;
    produs=1;
    aux=[];
    check=0;
    %matricea returnata anterior in part2_getvalues
    %verificam bitii de "1" si pentru fiecare pozitie
    %in care gasim acesti biti, vom inmulti valorile si adunam gradul
    %corespunzator acestor pozitii
    for i=1:length(values)
        if matrix(i) == 1
            sum=sum+grad(i);
```

```
            produs=produs*values(i);
            aux=[aux, i];
        end
    end
    %verificare sa nu existe dubluri
    % exemplu matricea pe biti:
    %[1 0 1 1 0 0] corespunde la
    %[y(k-1) y(k-2) u(k-1) y^2(k-1) y^2(k-2) u^2(k-1)]
    %algoritmul nostru va inmulti si y(k-1) cu y^2(k-1) dar noi avem
    %deja un y^3(k-1)

    check=part2_bool_check(aux,clones);


    if sum <= (grad(end)+1) && check==0
        output=produs;
    else
        output=10^40;
    end

end
```