10. ACORDAREA REGULATOARELOR PID UTILIZÂND ALGORITMI GENETICI

Algoritmii genetici fac parte din clasa tehnicilor de calcul evoluţionist şi sunt algoritmi de căutare inspiraţi din selecţia naturală şi din genetică.

Scopul lucrării este acordarea unui regulator de tip PID folosind algoritmi genetici.

10.1 Breviar teoretic

Metoda de optimizare folosind algoritmi genetici funcţionează după următorul principiu. Dată fiind o populaţie formată din indivizi (cromozomi) care constituie posibile soluţii ale problemei, aceştia sunt puşi în competiţie pentru supravieţuire. După evaluarea fiecărui individ, celor puternici li se dă o şansă mai mare de a participa la procesul de reproducere. Mecanismele prin care iau naştere copiii se numesc recombinare (crossover) şi mutaţie. La pasul următor, aceşti copii rivalizează între ei şi, posibil, chiar şi cu părinţii lor. Astfel, populaţiile vor fi din ce în ce mai aproape de soluţia dorită prin acest mecanism de selecţie a celor mai buni părinţi şi de eliminare a celor slabi.

Structura algoritmului genetic fundamental este prezentată în figura 10.1. Un algoritm genetic realizează o căutare multidirecţională. La fiecare generaţie, populaţia suportă un proces de evoluţie simulată, fiind generată o nouă generaţie, în care soluţiile relativ bune se reproduc, iar cele nesatisfăcătoare dispar.

Operatorii utilizaţi în mod uzual de către algoritmii genetici sunt selecţia, încrucişarea şi mutaţia. În funcţie de problemă, se pot alege şi alţi operatori (inversiune, reordonare, operatori speciali). Un algoritm genetic trebuie să aibă cinci componente de bază pentru a fi capabil să rezolve o problemă de optimizare, şi anume,

- o reprezentare genetică a soluţiei potenţiale a problemei;
- un mod de generarea a unei populaţii de indivizi;
- o funcţie de evaluare (sau funcţie cost) care joacă rolul mediului înconjurător;
- operatori care schimbă componenţa populaţiei;
- valori ale diferiţilor parametri ai algoritmului genetic (mărimea populaţiei, criterii de oprire, probabilităţi de aplicare a operatorilor etc.).

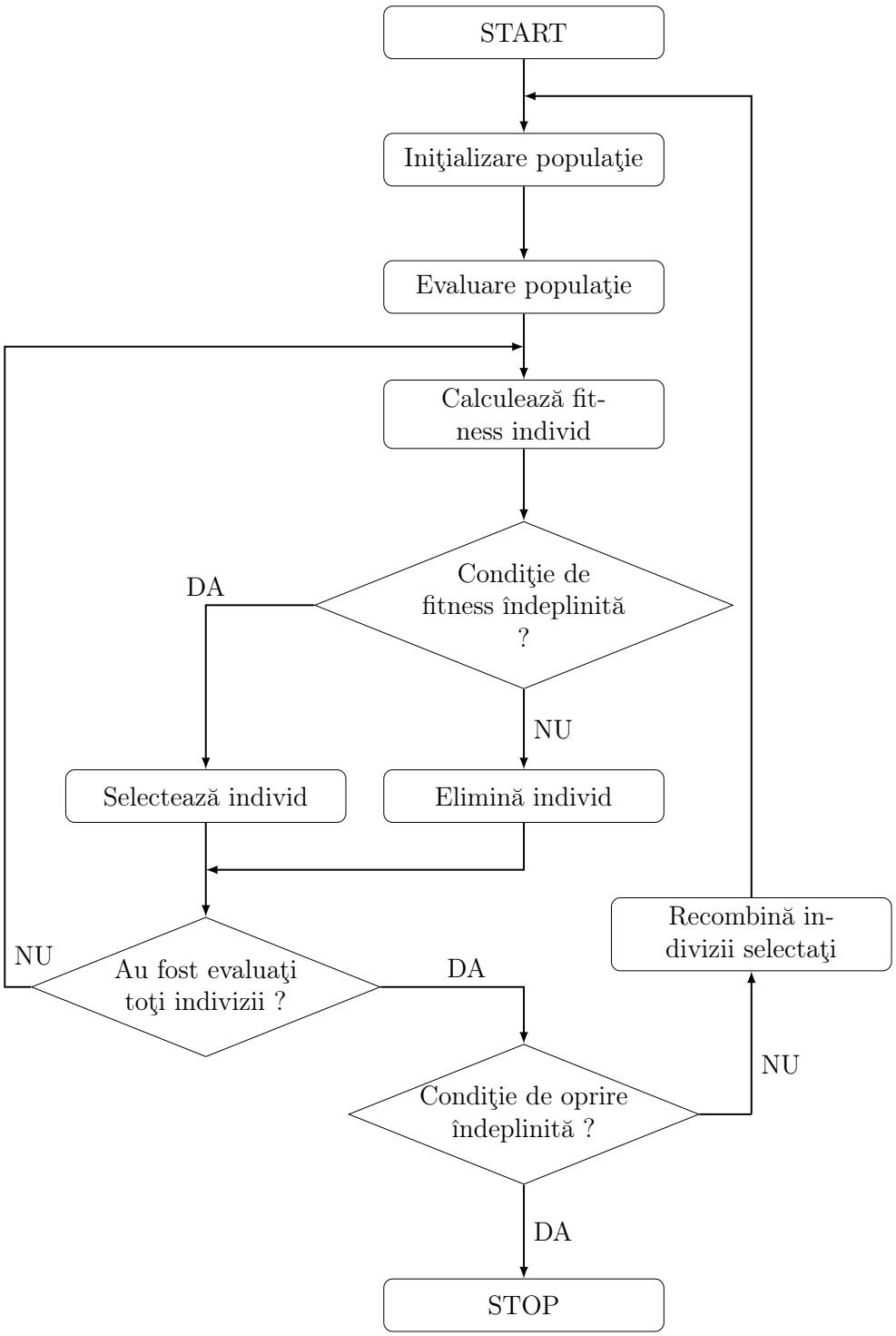

Figura 10.1: Structura unui algoritm genetic

Selecţia este un operator care joacă un rol important în cadrul unui algoritm genetic. Acest operator decide care dintre indivizii unei populaţii pot participa la formarea populaţiei următoare. Scopul selecţiei este de a asigura mai multe şanse de reproducere celor mai performanţi indivizi dintr-o populaţie dată. Prin selecţie se urmăreşte maximizarea performanţei indivizilor.

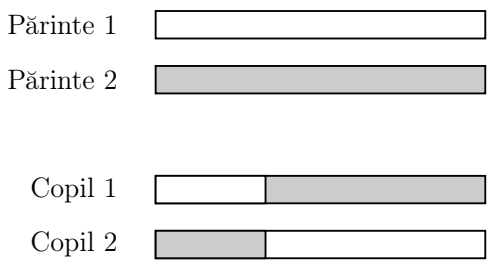

Figura 10.2: Încrucişarea într-un punct

Încrucişarea este aplicată indivizilor din submulţimea selectată a populaţiei. Sunt aleşi aleator indivizi şi anumite porţiuni din aceştia sunt interschimbate. Operatorul imită încrucişarea intercromozomială naturală. De regulă, se utilizează operatori de încrucişare de tipul $(2, 2)$, adică doi părinţi dau naştere la doi descendenţi. Încrucişarea realizează un schimb de informaţie între cei doi părinţi. Descendenţii obţinuţi prin încrucişare vor avea caracteristici ale ambilor părinţi. În figura 10.2 este reprezentată încrucişarea într-un punct.

Mutaţia introduce în populaţie indivizi care nu ar fi putut fi obtinuţi prin alte mecanisme. Operatorul de mutaţie acţionează asupra genelor[1] indiferent de poziţia lor în cromozom şi are rolul de menţinere a diversităţii genetice care asigură evitarea blocării în minime locale.

10.2 Acordarea regulatoarelor PID cu algoritmi genetici

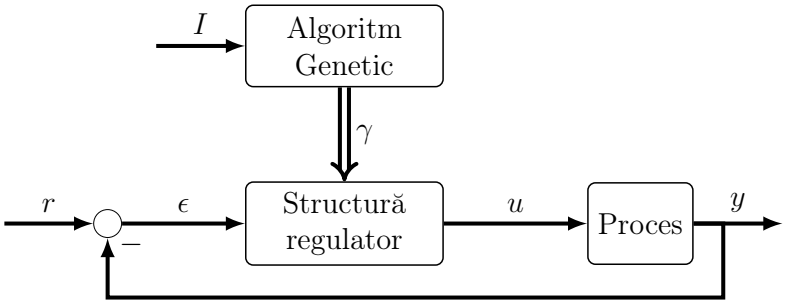

Figura 10.3: Acordarea regulatoarelor cu algoritmi genetici

În problema sintetizării regulatoarelor folosind algorimi genetici, metoda, ilustrată în figura 10.3, foloseşte o structură de regulator prescrisă (în cazul de faţă din clasa

[1] Componentele de bază ale cromozomilor.

regulatoarelor PID) conectată în SRA cu modelul, în cazul sintezei offline, sau direct cu procesul, în sinteză online. Pe baza erorii ϵ (dintre referinţa r şi variabila reglată y), tehnica de optimizare genetică minimizează un criteriu de performanţă I pentru a calcula cel mai potrivit regulator pentru procesul dat (care primeşte semnalul de intrare dat de comanda u). Ieşirea algoritmului genetic este un vector de valori ale parametrilor regulatorului codificat sub formă de cromozom, ca soluţii potenţiale ale problemei reglării. Cu această metodă, parametrii regulatorului PID ales pot fi acordaţi fin într-un mediu de simulare şi, apoi, testat/validat pe procesul real. Validarea se face prin verificarea câtorva indici de performanţă uzuali ai SRA, fiecare prezentând atât avantaje, cât şi dezavantaje.

Criteriul ISE (Integral of Square Error) este un indice general de minimizare a erorii de reglare pentru referinţă treaptă, urmărind reducerea rapidă a erorii de reglare şi, astfel, reducând valoarea timpului tranzitoriu. Principalul dezavantaj este că nu se ţine cont de eventuala apariţie a suprareglajului.

$$I_{ISE} = \int_0^\infty \epsilon^2(t)dt.$$

Criteriul IAE (Integral of Absolute Error) este un indice de minimizare a erorii de reglare, care utilizează valorile absolute ale erorii obţinute pentru referinţă treaptă. Acest criteriu prezintă aceleaşi dezavantaje ca şi ISE, dar necesită mai puţine operaţii la nivel de procesor.

$$I_{IAE} = \int_0^\infty |\epsilon(t)|dt.$$

Criteriul ITAE (Integral of Time multiplied by the Absolute value of Error) este un indice care minimizează timpul tranzitoriu. Principalul dezavantaj al acestuia este sensibilitatea mare la orice variaţie a parametrilor procesului, deşi oferă un suprareglaj mai mic decât ISE şi decât IAE.

$$I_{ITAE} = \int_0^\infty t|\epsilon(t)|dt.$$

Criteriul IEC (Integral of Error and Command) este un indice care se utilizează pentru minimizarea suprareglajului (în acest caz este de aşteptat o creştere a timpului tranzitoriu). Criteriul IEC este dependent de valoarea factorului de penalizare a comenzii ρ (strict pozitiv).

$$I_{IEC} = \int_0^\infty (\epsilon^2(t) + \rho u^2(t))dt.$$

Criterii specifice pot fi de asemenea utilizate prin combinarea criteriilor integrale cu alte criterii potrivite problemei. De exemplu, criteriul ISE poate fi suplimentat cu condiţii asupra suprareglajului.

10.3 Sarcini de lucru

1. Se consideră un motor de curent continuu utilizat pentru acţionarea unui robot mobil. Un model matematic aproximativ pentru caracterizarea motorului este

$$H_P(s) = \frac{K_P}{(T_1 s + 1)(T_2 s + 1)}, \tag{10.1}$$

unde $K_P = 1.35$, $T_1 = 0.87$ şi $T_2 = 0.57$. Se cere

a) o structură SRA a turaţiei motorului,
b) algoritmul de reglare astfel încât răspunsul la referinţă treaptă al SRA să fie aperiodic, cu eroare staţionară nulă şi un timp tranzitoriu $t_t \leq 7$sec.

Soluţie. Se realizează schema Simulink de reglare automată a procesului descris utilizând un regulator PID clasic în forma paralel, obţinându-se modelul *model-SRA.mdl* (figura 10.4). Paşii 2 şi 3 creează cele două funcţii Matlab pentru rularea algoritmilor genetici. □

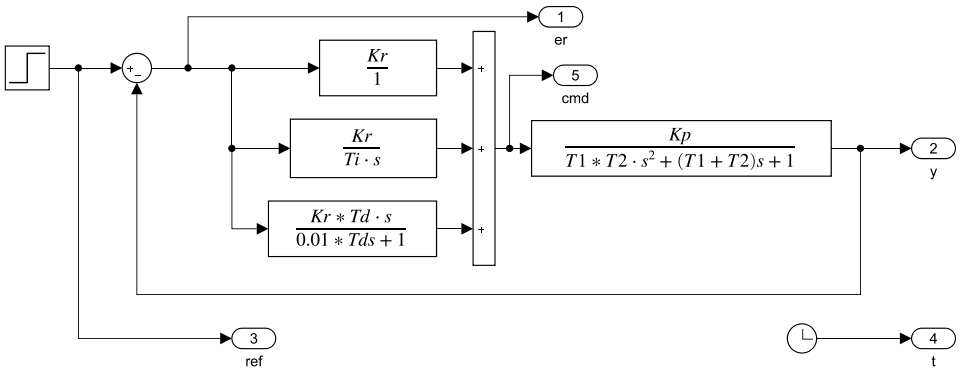

Figura 10.4: Modelul Simulink al SRA

2. Se realizează un program de optimizare utilizând algoritmi genetici care va căuta parametrii de acord optimi pentru regulatorul ales anterior. Pentru aceasta, se va utiliza toolbox-ul Matlab GAOT (Genetic Algorithm Optimization Toolbox). Se creează funcţia *algoritmulmeu.m*.

2.1. Iniţializare

- populationSize: reprezintă numărul de indivizi (cromozomi) în cadrul unei populaţii.

  ```
  populationSize=n; %n=numar natural intre 20 si 100
  ```

- variableBounds: reprezintă o matrice $n \times 2$ în care sunt date limitele inferioare şi cele superioare ale fiecărei caracteristici a individului, unde n reprezintă numărul de caracteristici ale individului. În cazul de faţă, individul are 3 caracteristici: K_r, T_i, T_d.

  ```
  variableBounds=[Krmin Krmax;Timin Timax;Tdmin Tdmax];
  ```

- evalFN: reprezintă evaluarea funcţiei obiectiv (fitness) şi i se va atribui numele funcţiei Matlab în care se evaluează *fitness*-ul fiecărui individ (vezi pasul 3).

  ```
  evalFN='functieobiectiv';
  ```

- evalOps: orice opţiuni care trebuie atribuite funcţiei de evaluare a *fitness*-ului. În cazul de faţă, se va utiliza valoarea implicită.

  ```
  evalOps=[];
  ```

- options: reprezintă o linie cu două elemente, dintre care primul reprezintă precizia parametrilor PID (caracteristicile individului), iar cel de-al doilea este tipul de reprezentare a individului (1- real, 0- binar).

  ```
  options=[1e-6 1];
  ```

- initializega: reprezintă comanda de iniţializare.

  ```
  initPop=initializega(populationSize,variableBounds,evalFN,
                       evalOps,options);
  ```

2.2. Setarea parametrilor de rulare

- bounds: la fel ca variableBounds, însă limitele variabilelor pot să difere de cele folosite la iniţializare, deorece acum trebuie definit întreg domeniul de căutare al algoritmului genetic.

  ```
  bounds=[Krmin Krmax;Timin Timax;Tdmin Tdmax];
  ```

- evalFN şi evalOps: vezi pasul 2.1.
- startPop: reprezintă prima populaţie care se testează şi care este dată de funcţia de iniţializare.

  ```
  startPop=initPop;
  ```

- opts: reprezintă un set de opţiuni, idem options (v. 2.1).

  ```
  opts=[1e-6 1 0];
  ```

- termFN: reprezintă declaraţia funcţiei de oprire a algoritmului genetic. Acest parametru este folosit pentru a opri algoritmul genetic odată au fost îndeplinite condiţiile de stop. În cazul de faţă, se va utiliza funcţia *maxGenTerm* din GAOT toolbox, care opreşte algoritmul după un număr maxim de generaţii.

```
termFN='maxGenTerm';
```

– termOps: reprezintă o opţiune care va fi atribuită funcţiei de oprire a algoritmului. Setarea implicită are valoarea 100, adică algoritmul va dezvolta 100 de generaţii înainte de a se opri. Pentru problema curentă, se începe cu 50 de generaţii.

```
termOps=50;
```

– selectFN: reprezintă declaraţia funcţiei de selecţie. Implicit se selectează *normGeomSelect (Normalised geometric selection)*. Toolboxul GAOT mai are două funcţii pentru selecţie implementate: *Tournament selection* şi *Roulette wheel selection*.

```
selectFN='normGeomSelect';
```

– selectOps: probabilitatea de a fi selectat cel mai potrivit individ.

```
selectOps=0.08;
```

– xOverFNs: reprezintă declaraţia funcţiei de încrucişare.

```
xOverFNs='arithXover';
```

– xOverFNs: reprezintă numărul de puncte de încrucişare.

```
xOverOps=4;
```

– mutFNs: reprezintă declaraţia funcţiei de mutaţie.

```
mutFNs='unifMutation';
```

– mutOps: reprezintă numărul total de mutaţii. Acesta se alege în aşa fel încât numărul total de mutaţii să fie între $8 \div 10\%$ din dimensiunea populaţiei. Pentru 100 de indivizi, se alege

```
mutOps=8;
```

2.3. Rularea procedurii de optimizare

```
[x,endPop,bPop,traceInfo]=ga(bounds,evalFN,evalOps,startPop,
                             opts,termFN,termOps,selectFN,
                             selectOps,xOverFNs,xOverOps,
                             mutFNs,mutOps);
```

2.4. Reprezentarea grafică a răspunsului la treaptă al sistemului în buclă închisă cu parametrii PID optimi

```
Kr = x(1);
Ti = x(2);
Td = x(3);
assignin('base', 'Kr', x(1));
assignin('base', 'Ti', x(2));
assignin('base', 'Td', x(3));
[T,state,outputs] = sim('modelSRA',20); %20[sec.] reprezinta
                                        timpul de simulare
```

```
y=outputs(:,2);
ref=outputs(:,3);
t=outputs(:,4);
figure(1);
hold on;
grid on;
G1=plot(t,y,'b');
G2=plot(t,ref,'m');
legend('Turatia motorului','Turatia de referinta');
axis tight;
hold off;
```

3. Se alege o funcţie obiectiv (*fitness*)

– Se creează funcţia *functieobiectiv.m*.

```
function[x_pop,fx_val]=functieobiectiv(x_pop,options)
```

– Se setează regulatorul PID cu parametrii daţi de individul curent.

```
Kr=x_pop(1);
Ti=x_pop(2);
Td=x_pop(3);
assignin('base', 'Kr', x_pop(1));
assignin('base', 'Ti', x_pop(2));
assignin('base', 'Td', x_pop(3));
```

– Se evaluează răspunsul la o intrare de tip treaptă.

```
[T,state,outputs] = sim('modelSRA',20);
                    %20 reprezinta timpul de simulare
er = outputs(:,1);
y = outputs(:,2);
ref = outputs(:,3);
t=outputs(:,4);
```

Observaţia 10.1. Parametrii y, er şi ref se obţin din modelul *modelSRA.mdl* corect prin numerotarea corespunzătoare a porturilor de ieşire.

– Se evaluează eroarea pătratică medie de urmărire a referinţei (indicele ISE va asigura eroare staţionară zero).

```
IND = (er'*er); %ISE - Integral of Square Error
```

Observaţia 10.2. Indivizii care nu îndeplinesc cerinţele de performanţă (răspuns aperiodic şi $t_t \leq 7$sec.), trebuie penalizaţi prin atribuirea unei valori de IND foarte mare.

```
yt1=find(er>0.05, 1, 'last');
```

```
yt2=find(er<-0.05, 1, 'last'); %pe liniile de mai sus se cauta
                                banda de regim stationar de 5%
SettlingTime = t(max([yt1 yt2]));
Overshoot = abs(min(min(er),0));
if (Overshoot > OvershootCond & t1 > SettlingTime > t2)
                %se inlocuieste OvershootCond cu o valoare
                              specifica problemei de rezolvat
                %se adauga conditii corespunzatoare
                              asupra timpului tranzitoriu
IND = 10e300;
end
```

– Se calculează fitness-ul individului (procedurile din toolbox-ul GAOT maximizează, aşadar inversarea valorii este necesară minimizării în problema de faţă).

```
fx_val=1/IND;
```

4. Se rulează funcţia *algoritmulmeu.m*.

Observaţia 10.3. În cazul în care algoritmul nu găseşte o soluţie convenabilă, se vor ajusta intervalele de căutare, atât la iniţializare, cât şi la rularea procedurii de optimizare. Pentru utilizatorii începători, se recomandă, iniţial, setarea aceloraşi intervale.

5. Se compară rezultatele obţinute prin procedura de optimizare pe bază de algoritmi genetici cu rezultatele obţinute la lucrarea 5.