Laborator 3

Probleme de optim în automatică

1. Obiectivele laboratorului

- Însușirea unei metode de calcul a mărimii de control pentru obținerea unei traiectorii date pentru ieșirea unui sistem, folosind algoritmi genetici
- Însușirea unei metode de identificare a sistemelor, folosind algoritmi genetici
- Exerciții

2. Studiu de caz I: controlul unui sistem pentru a obține traiectoria dată

În teoria controlului se abordează de multe ori problema calculului mărimii de control a unui sistem astfel încât ieșirea să fie cât mai aproape de o referință dată. O problemă mai complexă și dificil de rezolvat pe căi clasice este aceea de a calcula mărimea de control care produce o traiectorie a ieșirii cât mai aproape de o traiectorie referință. Lungimea intervalului în care traiectoria referință trebuie urmărită în mod optim se numește *orizont de optimizare*.

2.1. Formularea problemei

Se dă un sistemul de gradul II în spațiul stărilor, cu o intrare, două stări și o ieșire. Valorile inițiale ale stărilor sunt zero, iar mărimea de intrare $u(k)$ este limitată în intervalul [-2, 2].

$$\begin{cases} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.8 & 0.1 \\ 0.2 & 0.7 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 5 \\ 3 \end{bmatrix} \cdot u(k) \\ y(k) = \begin{bmatrix} 0.3 & -0.7 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \end{cases}$$

Se consideră orizontul de optimizare [0, 13]. Se cere să se calculeze mărimea de intrare $u(k)$ pentru care mărimea de ieșire se apropie cât mai mult de referința indicată cu roșu în figura de mai jos.

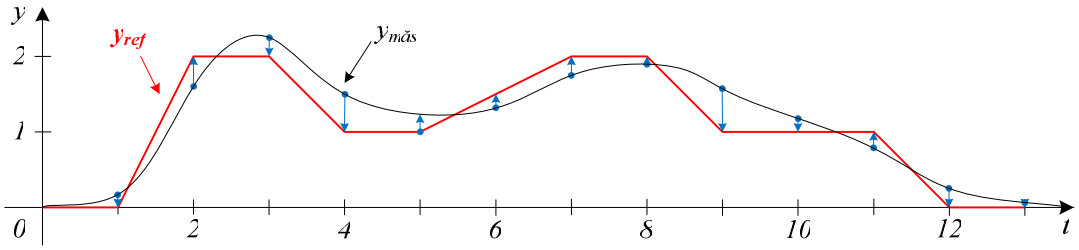
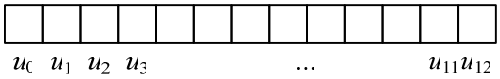


Figura 5. Controlul unui sistem pentru a obține traiectoria dată

2.2. Rezolvarea problemei

Soluția problemei (individul) este un vector $U = [u_0, u_1, ..., u_{12}]$ de valori reale, care conține fiecare mărime de intrare pentru intervalul [0, 12]. Întradevar, orizontul de optimizare este [0, 13], dar se observă că valoarea u_{13} nu își produce efectele în intervalul de optimizare, deci este irelevantă din punctul de vedere al problemei.

Pentru această problemă, vom considera un cromozom compus din 13 gene, așa cum este indicat mai jos. Valoarea pentru fiecare alelă este un număr real (`DoubleGene`) în intervalul dat, deci $u_k \in G = [-2, 2]$. Funcția de mapping va fi deci funcția identică.



Funcția de performanță vizează, ca și în capitolul 3 din laboratorul precedent, diferența între valoarea *y* măsurată și referința dată. Pentru că sistemul este discret, se vor considera numai valorile care corespund momentelor de eșantionare $t = 0, 1, 2, \ldots, 13$, deci segmentele indicate cu albastru în figura anterioară.

2.3. Implementare

Oferim mai jos codul relevant pentru funcția de performanță. Programul principal rămâne similar cel prezentat în laboratorul anterior.

În secvența de cod de mai jos, metoda `Mapping(chr)` extrage din obiectul de tip `IChromosome` valorile alelelor și returnează un array de 13 valori double. Metoda `GetY(u)` este practic simulatorul sistemului: pe baza a 13 valori de intrare, *u*, calculează iterativ mai întâi stările x_1, x_2 și apoi valorile de ieșire *y*. Aceste două metode trebuie implementate.

Secvența de cod 1. Funcția de performanță

```
import org.jgap.*;
public class FitnessFunctionTrajectory extends FitnessFunction {
   private static final double[] yref = { 0, 0, 2, 2, 1, 1, 1.5, 2, 2,
1, 1, 1, 0, 0 };
   private static final int optimizationHorizon = 13;
   public double evaluate(IChromosome chr) {
      double[] u = Mapping(chr);
      double[] y = GetY(u);
      double errorSum = 0;
      for (int i=0; i<optimizationHorizon; i++)
        errorSum += Math.abs(y[i]-yref[i]);
      return errorSum;
   }

   private double[] Mapping(Ichromosome chr) {
      //add Mapping(chr) method
   }

   private double[] GetY(double[] u) {
      //add GetY(u) method
   }
}
```

2.4. Testare și concluzii

Observații privind structura cromozomului și operatorii genetici

La problemele din laboratorul precedent am considerat cromozomul ca având gene de tip boolean; practic, am distribuit fiecare necunoscută care trebuie aflată pe parcursul a mai multe gene (vezi laborator 2, capitolul 3). Numărul de gene devine astfel mai mare (deci implică mai multe calcule), dar operatorul de încrucișare este mai eficient. Nu există o regulă care dictează în ce cazuri se abordează o strategie sau alta, decizia ține mai mult de experiența programatorului.

În general, operatorul de încrucișare este cel care transformă cea mai mare parte a cromozomilor, ajutând genele bune să se perpetueze de la o generație la alta. Operatorul de mutație este responsabil pentru introducerea, din când în când, a unor noi caracteristici în bazinul de cromozomi, dar și pentru scoaterea sistemului intr-o eventuală stare de optim local, dar nu global.

În cazul folosirii genelor de tip număr real, nu se poate ajusta precizia folosită, ea fiind implicit precizia (variabilă!) a numerelor reale în limbajul de programare ales.

Parametrii de testare

În cazul acestei probleme, vom stabili mărimea populației mai mare decât în alte cazuri pentru a introduce un grad mare de hazard (engl. *randomness*) de la început, pentru a compensa ineficiența relativă a operatorului de încrucișare. Vom stabili populația ca având 1000 de indivizi și vom lăsa algoritmul să ruleze 200 de generații.

Limitările abordării

Pentru a garanta convergența și comportamentul consistent al algoritmului, este necesar ca, ori de câte ori se evaluează același cromozom (în aceeași generație sau în generații diferite) să obținem aceeași valoare a performanței. Dacă sistemul are o componentă stohastică sau există perturbații aleatoare care nu pot fi controlate (chiar dacă ele pot fi măsurate), este evident că evaluarea nu poate fi făcută în mod consistent. În acest caz, putem recurge de exemplu, la simularea repetată a sistemului (cu diferite perturbații) și returnarea unui indice de performanță mediu.

O altă limitare o constituie sistemele continue, pentru care mărimea de control nu poate fi mapată pe un cromozom cu componente discrete. În acest caz, putem recurge la aproximarea sistemului prin discretizare.

3. Studiu de caz II: identificarea sistemelor folosind algoritmi genetici

Metodele performante de identificare a sistemelor sunt extrem de laborioase și fac uz de multe detalii care sunt de obicei greu de reținut, mai ales dacă nu se lucrează cu ele sistematic. Prezentăm în continuare o variantă de identificare a sistemelor cu algoritmi genetici, care primează prin ușurința abordării. Ca în orice problemă, parametrii necunoscuți trebuie codificați în cromozom și evaluarea se face prin simularea sistemului.

3.1. Formularea problemei

Se dă un sistem de gradul II ca o cutie neagră (*blackbox*), deci nu se cunoaște decât comportamentul exterior (mărimea măsurată *y*, în funcție de intrarea *u*). Se cere să se calculeze parametrii unui sistem de gradul II în spațiul stărilor care aproximează cât

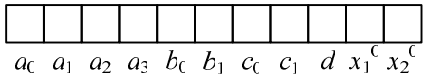
mai bine sistemul dat. Există o limitare pe mărimea de intrare $u \in [0, 5]$, iar orizontul de optimizare este $[0, \tau]$

3.2. Rezolvarea problemei

Considerăm modelul unui sistem de gradul II în spațiul stărilor în formă generală. Necunoscutele sunt coeficienții $a_0, \ldots, a_3, b_0, b_1, c_0, c_1, d$, numere reale:

$$\begin{cases} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ a_2 & a_3 \end{bmatrix} \cdot \begin{bmatrix} x_2(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \cdot u(k) \\ y(k) = \begin{bmatrix} c_0 & c_1 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + d \cdot u(k) \end{cases}$$

Ca și în exemplul anterior, vom considera cromozomul ca fiind format din gene cu alele de tip număr real (`DoubleGene`), și vom limita intervalul genotip G = [-10, 10]. Este nevoie deci de 9 gene pentru parametri și încă 2 gene pentru valorile stărilor inițiale, x_1^0, x_2^0, cu același genotip G. Individul va fi modelul sistemului în spațiul stărilor.



Aranjamentul pentru evaluarea sistemului este arătat în figura de mai jos. Pentru a calcula performanța unui model găsit vom lua suma integrată a erorii între $y_{măsurat}$ și $y_{estimat}$ pe întreg orizontul de optimizare τ. Vom considera în schimb pătratul erorii, pentru a penaliza progresiv diferențiat cazurile în care comportamentul modelului se abate puternic de la cel al procesului pe care îl identificăm. Avem deci, în varianta discretă: $J = \sum_{k=0}^{\tau} \left(y_m(k) - y_{est}(k)\right)^2$. Valorile ieșirii $y(k)$, cu $k \in [0, \tau]$ vor fi calculate ca și în exemplul precedent, folosind o metodă căreia îi transmitem ca parametrii modelul și valorile mărimilor de intrare u, `double[] GetY(a, b, c, d, u)`.

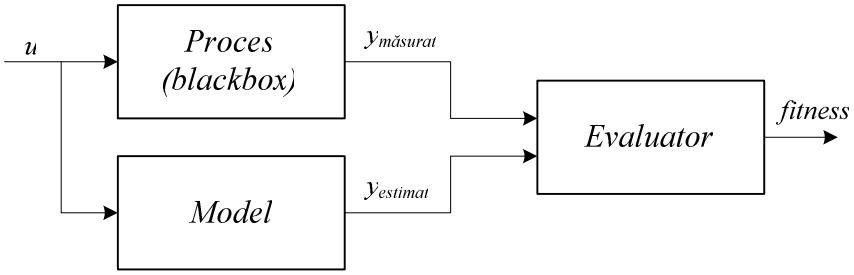


Figura 6. Aranjament pentru identificarea sistemului

Problema de optimizare este aceea de a *minimiza* funcția de performanță J pentru *toate valorile posibile* ale intrării u. Dacă folosim o singură valoare a intrării, de exemplu semnalul treaptă, vom obține un model al sistemului care funcționează bine numai pentru acea valoare a intrării u. Pentru simplitate, vom considera totuși doar 2 cazuri: treaptă, și rampă. Vom avea deci:

$$J = \sum_{k=0}^{\tau} \left(y_m^{step}(k) - y_{est}^{step}(k) \right)^2 + \sum_{k=0}^{\tau} \left(y_m^{ramp}(k) - y_{est}^{ramp}(k) \right)^2$$

Vom alege τ = 20s, iar pentru semnalul treaptă: $u(k) = 0$ pentru $k \leq 5$ și $u(k) = 1$ pentru $k > 5$. Sistemul trebuie să păstreze ieșirea și în lipsa unui semnal de intrare.

Pentru semnalul rampă: $u(k) = 0.25 * k$, factorul de scalare având rolul de a păstra valoarea intrării în intervalul dat.

Nu este întotdeauna evident ce influență au semnalele de intrare stabilite asupra performanței modelului obținut (sau a soluției problemei, pe caz general). Ca o regulă generală, o soluție generată va fi performantă doar pentru cazurile în care ea a fost evaluată (testată) în funcția de performanță. Ține de experiența programatorului găsirea acelor scenarii de testare a soluției potențiale care să acopere toate cazurile necesare în problemă.

4. Exerciții

1. Pentru exemplul din capitolul 2, implementați un program Java folosind pachetul JGAP, care calculează mărimea de intrare $u(k)$, k=0,1,...,12 cu algoritmi genetici. Pentru aceasta, completați metodele `Mapping(chr)` și `GetY(u)`, oferite în secvența de cod dată.
2. Cum ar trebui modificată funcția de evaluare a programului anterior dacă sistemul primește o perturbație aleatoare în intervalul [-0.2, 0.2] care acționează direct pe starea x_1 ? Realizați modificarea și evaluați convergența și consistența algoritmului.
3. Înlocuiți numărul fix de generații cu următoarea condiție de oprire: dacă pe parcursul a 10 generații nu se înregistrează o creștere a performanței mai mare decat 5%, algoritmul se oprește. Câte generații rulează algoritmul, în medie ?
4. Pentru exemplul din capitolul 3, implementați un program Java care să rezolve identificarea sistemului folosind algoritmi genetici. Pentru sistemul necunoscut (*blackbox*) se va folosi sistemul din capitolul 2, cu valorile inițiale ale stărilor: $x_1^0 = 2$, $x_2^0 = -2$.
5. Testați sistemul identificat cu un semnal periodic, apoi unul aleator. Cum răspunde sistemul ? Cum se poate modifica funcția de evaluare pentru a cuprinde și acest caz ?