Chaîne de Markov : un problème de contrôle stochastique

Bernard Lapeyre
CERMICS, École des Ponts ParisTech

22 mars 2018

Table des matières

Le fichier source en `Scilab` correspondant à ce document est accessible scilab_controle.sci. Il est partiellement mais pas totalement corrigé. Le correction complète sera accessible scilab_controle_corrige.sci à la fin du TP.

Vendeur de journaux : le cas de deux indices de temps

On commence par définir les caractéristiques d'une loi discrète par le vecteur $(w_i, 1 \leq i \leq N)$ des valeurs possibles et la probabilité $(p_i, 1 \leq i \leq N)$ de chaque valeur.

```
// une loi discrète
w_i=[30,50,80];
p_i=[1/2,1/4,1/4];
mu=p_i*w_i';// la moyenne
```

On tire un échantillon de taille N selon cette loi. Pour la simulation on utilise `grand` pour simuler une chaine de Markov dont toutes les lignes sont égales à $(p_i, 1 \leq i \leq N)$. Ce qui fait le travail (exercice sur ... les chaînes de Markov), même si c'est un peu "tordu".

```
c=10,cm=20,cs=5;cf=200;

function y=J(u)
  // La fonction J(u) est le cout moyen de  j(u,w) sous la loi p_i
  //          j(u,w) = c*u + cs*max(u - w,0) + cm*max(w - u,0)
  // Cette espérance se calcule donc par :
  y=c*u+cs*p_i*max((u-w_i'),0)+cm*p_i*max((w_i'-u),0);
endfunction
```

La fonction suivante calcule le minimum de $J(u)$.

```
function [uopt,m]=minimum(J)
  // calcul de J(u) pour u entre -10 et 100.
  U=-10:100;
  Ju=[];for u=U do Ju=[Ju,J(u)];end

  // recherche du minimum de J pour u entre -10 et 100.
  [m,kmin]=min(Ju);
  uopt=U(kmin);
endfunction
```

Pour tracer la fonction $J(u)$ et matérialiser le minimum remplacer `%f` par `%t` dans ce qui suit.

```
if %f then
  xset('window',1);xbasc();

  // Tracé de J(u)
  U=-10:100;
  Ju=[];for u=U do Ju=[Ju,J(u)];end
  plot2d(U,Ju,style = 2);
  xtitle("La fonction J");

  [uopt,m]=minimum(J);

  printf("Nombre optimal de journaux a commander %f\n",uopt);
  printf("Moyenne de la demande %f\n",mu);

  // materialisation du point minimisant
  plot2d(uopt,m,style = -4);
  plot2d3(uopt,m);
  // xclick();// permet d'arrêter le programme
end
```

On regarde maintenant un cas où le vendeur a déjà des journaux et où il paye un coup fixe c_F s'il commande des journaux. On s'attend, en optimisant, à obtenir une une stratégie de type $[s, S]$. On peut le vérifier. On introduit la function $\tilde{J}$.

```
function y=Jtilde(u,x)
  y=cf*(u > 0)+J(u+x)-c*x
endfunction
```

On calcule, pour x (entier) variant de 0 à $2\max(w_i)$, la commande optimale de journaux correspondant à chaque valeur de x.

```
xv=0:1:2*max(w_i);// les valeurs de x prises en compte
U=0:2*max(w_i);// les valeurs de U prises en compte
xuopt=[];
```

```
for x0=xv do
  Ju=[];for u=U do Ju=[Ju,Jtilde(u,x0)];end
  [m,kmin]=min(Ju);
  xuopt=[xuopt,U(kmin)];
end
```

On veut calculer s où s est la valeur maximum à partir de laquelle on passe une commande > 0. `xuopt=0` signifie que l'on commande rien pour le x correspondant. On va chercher toutes les indices correspondant à des valeurs pour lesquelles `xuopt=0`.

```
I=find(xuopt==0);
```

Puis on selectionne la plus petite des valeurs d'indice i.e. `I(1)`, la valeur de x correspondant à l'indice juste avant (`I(1)-1`) va nous donner s par `s=xv(I(1)-1)` ;

```
s=xv(I(1)-1);
```

On peut alors vérifier que la stratégie optimale est bien de la forme $[s, S]$.

```
xset('window',1);
xbasc();
plot2d2(xv,xuopt,style = 2);
plot2d2(xv(1:s),xv(1:s)+xuopt(1:s),style = 1);
xtitle(sprintf("Valeur de s=%d et de S=%d",xv(s),xv(s-1)+xuopt(s-1)));

// Calcul de S le minimiseur de  min J(u) = JS = J(S)
[S,JS]=minimum(J);

// On calcule maitenant s
x=0:2*max(w_i);
Jv=[];for i=1:size(xv,'*') do Jv=[Jv,J(x(i))];end
I=find(Jv <= cf+JS);
if isempty(I) then s=0;else s=x(I(1)-1);end;

printf("On trouve s=%d et S=%d\n",s,S);

// Il faut verifier que I(i+1)=I(i)+1 pour tout i=1..size(I)-1;
// pour prouver que l'ensemble d'exercice est de la forme [s,S]
// On doit trouver T
and(I(2:$)==(I(1:$-1)+1))
```

Vendeur de journaux : le cas de T pas de temps

On commence par décrire le système dynamique contrôlé en se restreignant à un ensemble borné.

```
// on transforme la dynamique pour rester dans un ensemble
// borné [-100,100]
XMIN=-100;XMAX=100;

function y=f(x,u,w)
  y=min(max(x+u-w,XMIN),XMAX)
endfunction
```

La fonction de coût.

```
alpha=1.0;//le coefficient d'actualisation

function y=ct(u,x)
  // coût instantané pour t autre que la date de départ
  y=cf*(u > 0)+c*u+alpha*cs*max(x,0)+alpha*cm*max(-x,0)
endfunction

function y=c0(u)
  // co^ut instantané pour t=0
  y=cf*(u > 0)+c*u
endfunction

function y=K(x)
  // coût final, noté |£h(x)£| dans le texte
  y=cs*max(x,0)+cm*max(-x,0)
endfunction

function res=cout(t,x,u)
  // Noté |£l(n,x,u)£| dans le texte.
  // Vaut c0 pour t=0 et ct sinon
  if t==1 then
    res=c0(u);
  else
    res=ct(u,x);
  end
endfunction

function cost=ecost(t,x,u,cout,Vtp1)
  // Pour t, u et x fixé calcul de
  //        |£l(t,u,x) + \E(v(t+1,f(x,u,W_1))£|
  cost=0;
  for l=1:size(w_i,'*') do
```

```
    // boucle sur les aléas pour calculer
    //          |£\E(v(t+1,f(x,u,W_1))£|
    xtp1=f(x,u,w_i(l));// calcul du nouvel état
    ix=find(xtp1==ensemble_etats);// on retrouve son indice
    // dans le tableau des états
    cost=cost+p_i(l)*Vtp1(ix);
  end;
  cost=cout(t,x,u)+cost;// On rajoute |£l(t,x,u)£|
  cost=alpha^t*cost;// on actualise
endfunction
```

On calcule $v(t,x)$ à l'aide de l'équation de Bellman. On commencera avec $T = 2 : 2$ indices de temps (0 et 1) qui correspond le problème à un pas de temps.

```
T=2;// l'horizon de temps
ensemble_etats=XMIN:XMAX;// les etats possibles (fini)
ensemble_controles=0:XMAX;// les commandes de journaux possibles (fini)
```

On commence par diverses initialisations. On affecte $\alpha^T \times K(.)$ à $v(T,.)$. On va calculer $V(t,.)$ pour $T-1,...,1$ et stocker les résultats du calcul de $V(t,.)$ dans une liste `L` et de la commande (feedback) optimale $U(t,.)$ dans `U`.

```
// La fonction v_T(x) = K(x)
VT=alpha^T*K(ensemble_etats);
L=list(VT);// liste qui permet de stocker (V_T,V_{T-1},.....,V_1)
U=list(0);// liste qui permet de stocker (-- ,U_{T-1},.....,U_1)
```

On exécute la boucle rétrograde en temps pour résoudre l'équation de Bellman.

```
for t=T-1:-1:1 do
  printf("t=%d\n",t);
  Vt=zeros(1,size(ensemble_etats,'*'));
  Ut=%nan*ones(1,size(ensemble_etats,'*'));
  Vtp1=L($);// Vtp1 est la fonction de Bellman au temps t+1

  // On cherche a calculer ici Vt et Ut
  for i=1:size(ensemble_etats,'*') do
    x=ensemble_etats(i);
    mcost=%inf;
    // boucle sur les commandes possibles
    for k=1:size(ensemble_controles,'*') do
      u=ensemble_controles(k);
      // on affecte à cost le coût associé à la commande u.
      cost=ecost(t,x,u,cout,Vtp1);
      // Il faut maintenant comparer cost au meilleur coût (mcost)
      // trouvé jusque là (on minimise) et mettre à jour mcost
      // et kumin (l'indice de la commande qui donne le meilleur coût)
```

```
      if cost < mcost then
        mcost=cost;
        kumin=k;
      end
    end
    // On stocke pour l'état x d'indice i le coût optimal
    Vt(i)=mcost;
    // et la commande optimale
    Ut(i)=ensemble_controles(kumin);
  end
  // On range Vt et Ut a la fin(=£) de la liste
  L($+1)=Vt;
  U($+1)=Ut;
end
```

Dessin de la fonction de Bellman et de la commande optimale au temps $t = 1$.

```
xset('window',1);
xbasc();
plot2d2(ensemble_etats,L($));// la fonction de Bellman au temps t=1;

xset('window',1);
xbasc();
plot2d2(ensemble_etats,U($));// la commande optimale au temps t=1;
```