Rapport de projet sur la Programmation Dynamique
Optimisation et contrôle

Mohammed Amine KHELDOUNI

26 mai 2017

1 Introduction

On se propose de faire une application simple d'un cas de programmation dynamique pour illustrer l'intérêt d'une telle méthode et commenter les résultats numérique que l'on peut observer sur un cas simple d'optimisation.

2 Réponse aux questions

Question 1

Rappelons la dynamique qui régit le système :

$$x_{t+1} = x_t + u_t + w_{t+1}$$

A t fixé, on note $x = x(t)$.

Pour $x \in \mathbb{X}$, on sait que l'état x_{t+1} est généré par la dynamique (1).
Comme w_{t+1} est l'aléatoire du modèle et peut prendre les valeurs $\{-1, 1\}$, nous allons chercher l'ensemble $\mathbb{B}(x)$ qui assure que les états restent dans le même ensemble $\mathbb{X}$. En effet, cela est possible par distinction des cas :

- Si $x = x^{\#}$, on prend $\mathbb{B}(x) = \{-1\}$
- Si $x = x^{\#}$, on peut se permettre d'avoir un contrôle négatif ou nul mais pas positif cas l'aléatoire risque de générer un état hors de l'ensemble $\mathbb{X}$.
  Donc $\mathbb{B}(x) = \{-1, 0\}$
- Si $x = 1$, il suffit de prendre $\mathbb{B}(x) = \{1\}$
- Si $x = 2$, on peut se permettre d'avoir un contrôle nul mais pas négatif.
  Donc $\mathbb{B}(x) = \{0, 1\}$
- Sinon, si x n'est pas dans un "bord" de $\mathbb{X}$, on peut prendre $\mathbb{B}(x) = \{-1, 0, 1\}$.

Ceci est illustrer dans la fonction SCILAB ci-dessous :

```
// Compute B(x) subset :
function [res]=B(x)
    if x == 1 then
        res = [1];
    elseif x == Nx then
        res = [-1];
    elseif x == 2 then
        res = [0,1];
    elseif x == Nx-1 then
        res = [-1,0];
    else
        res = [-1,0,1];
```

```
    end
endfunction
```

Une fois l'ensemble des contrôles admissibles $\mathbb{B}(x)$ pour un état x calculé, on effectue des itérations de boucle sur les états, les contrôles admissibles et l'aléa pour calculer le coût en espérance que l'on prend quadratique et symétrique en x_{ref} :

$$K(x_{t_f}) = (x(t_f) - x_{ref})^2$$

Le problème d'optimisation devient alors :

$$\underset{u(.),\ x(.)}{min} \quad \mathbb{E}(K(x(t_f)))$$

Question 2

Pour résoudre le problème d'optimisation, nous faisons appel à l'équation de Bellman de programmation dynamique stochastique :

$$V(t,x) = min_u \mathbb{E}(V(t+1, x_t + u_t + w_{t+1})$$

avec la condition aux bords $V(t_f, x) = K(x)$.

On retient finalement les valeurs du coût dans une matrice V et les valeurs du contrôle dans une matrice U.
Enfin, on actualise le coût selon l'état x_t en calculant l'espérance des coûts futurs engendrés par les aléas possibles $\mathbb{W} = \{-1, 1\}$.
Ceci est illustrer sur le code SCILAB ci-dessous :

```
for t = Tf-1:-1:Ti do
    for x = 1:Nx do
        cost_x = %inf;
        u_opt = 0;
        Bset = B(x);
        for k = 1:length(Bset) do
           u = Bset(k);
            cost_x_u_w = 0;
            for w = -1:2:1 do
                cost_x_u_w = cost_x_u_w + V(t+1,x+u+w)*p(w);
            end
            cost_x_u = cost_x_u_w;
            if cost_x_u < cost_x then
                u_opt = u;
                cost_x = cost_x_u;
            end
```

```
17          end
18          V(t,x) = cost_x;
19          U(t,x) = u_opt;
20      end
21  end
```

Résultats

On obtient alors une matrice de coûts V dans laquelle on retrouve la symétrie par rapport au minimum de notre fonction K :

$$V = \begin{pmatrix} 1.066 & 1.015 & 1.003 & 1 & 1 & 1 & 1.003 & 1.015 & 1.066 \\ 1.125 & 1.03125 & 1.0078125 & 1 & 1 & 1 & 1.0078125 & 1.03125 & 1.125 \\ 1.234375 & 1.0625 & 1.015625 & 1 & 1 & 1 & 1.015625 & 1.0625 & 1.234375 \\ 1.4375 & 1.125 & 1.03125 & 1 & 1 & 1 & 1.03125 & 1.125 & 1.4375 \\ 1.8125 & 1.25 & 1.0625 & 1 & 1 & 1 & 1.0625 & 1.25 & 1.8125 \\ 2.5 & 1.5 & 1.125 & 1 & 1 & 1 & 1.125 & 1.5 & 2.5 \\ 3.75 & 2. & 1.25 & 1 & 1 & 1 & 1.25 & 2. & 3.75 \\ 6. & 3. & 1.5 & 1 & 1 & 1 & 1.5 & 3. & 6. \\ 10. & 5. & 2. & 1 & 1 & 1 & 2. & 5. & 10. \\ 16. & 9. & 4. & 1 & 0. & 1 & 4. & 9. & 16. \end{pmatrix}$$

Enfin, on trace la matrice des coûts V en fonction des états et on obtient la courbe suivante :

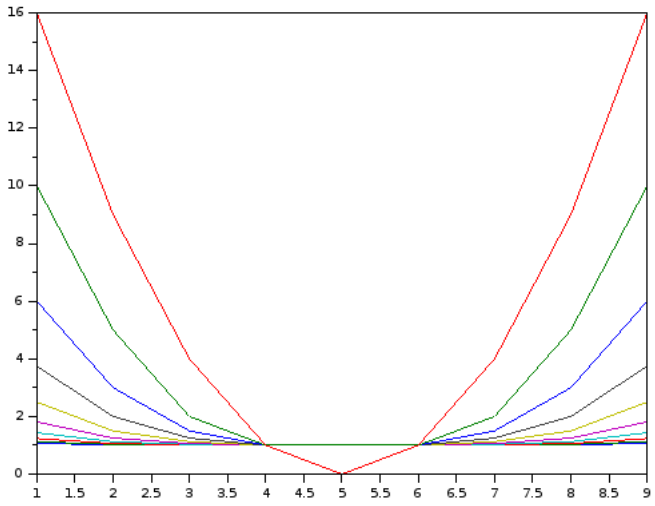

FIGURE 1 – Tracé de la matrice des coûts de Bellman par rapport aux états possibles

On remarque que la courbe rouge ($t = T_f$) fournit en $x = x_{ref}$ un coût nul mais engendre des coûts très importants sur les extrémités. On retrouve donc le caractère convexe de la fonction de coût.
De plus, nous pouvons expliquer cette augmentation du coût sur un temps plus long par rapport à un horizon fini plus proche par l'aléa qui génére forcément plus de risque et donc en espérant un plus grand coût pour un état très éloigné de l'état optimal.
Ce phénomène d'aplatissement des courbes pour des horizons de moins en moins loin revient donc au fait que le calcul de l'espérance du coût est bien moins importante lorsqu'on itére sur moins d'éléments $t \in \mathbb{T} = [T_i, T_f - 1]$.

Question 3

En reprenant le code SCILAB, on trace la matrice des contrôles U qui apparaît comme suit :

$$U = \begin{pmatrix} 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 0. & -1. & -1 & -1 & -1 \end{pmatrix}$$

Cette matrice de contrôle est bien symétrique par rapport à la colonne $x_{ref} = 5$, et montre que si l'on s'aventure dans $\mathbb{X}$ pour des états x à droite, elle vaut -1 et a tendance à renvoyer l'état à son optimum x_{ref}, et si on s'aventure à gauche de l'état minimisant les coûts, on obtient un contrôle de 1 pour parcourir des états à droite dans l'ensemble $\mathbb{X}$.

Question 4

Nous avons dans cette question, reformulé le problème en programmant une fonction SCILAB calculant sur demande le contrôle optimal à appliquer pour un instant t et un état x fixé.
Ainsi, nous produisons une réponse au problème qui permettra de fournir l'action optimale à enclencher pour générer un coût minimal en partant d'un état du problème x à l'instant t.

```

function [ud]=optControl(t,x)
    cost_x = %inf;
    ud = 0;
    Bset = B(x);
    for k = 1:length(Bset) do
```

```
        u = Bset(k);
         cost_x_u_w = 0;
         for w = -1:2:1 do
             cost_x_u_w = cost_x_u_w + V(t+1,x+u+w)*p(w);
         end
         cost_x_u = cost_x_u_w;
         if cost_x_u < cost_x then
             ud = u;
             cost_x = cost_x_u;
         end
     end
endfunction
```

Question 5

Dans cette dernière sous-section, nous avons utilisé les différentes macros fournies pour évaluer l'espérance du *payoff*. Dans une première méthode, nous avons utilisé la loi forte des grands nombres (LFGN) pour approcher cette espérance par Monte-Carlo. Dans un second temps, nous avons effectué un calcul exact de cette espérance par génération de la loi des w aléatoires. Enfin, nous allons réutilisé la programmation dynamique pour faire ce calcul d'espérance et renvoyer le vecteur des *payoff* pour chaque état.

Pour la méthode Monte-Carlo, nous avons donc fait tourner l'algorithme pour $N = 10000$ pour avoir une bonne approximation de la vraie valeur du *payoff* :

On obtient alors le vecteur de *Payoff* approchés suivant :

$$payoff_{MC} = (1.0705, 1.0136, 1.0031, 1, 1, 1, 1.0047, 1.0128, 1.0724)$$

Pour la deuxième méthode de calcul, nous générons tous les aléas possibles W dans une matrice pour effectuer le calcul exact de l'espérance :

On obtient alors :

$$payoff_{EX} = (1.0664062, 1.015625, 1.0039062, 1, 1, 1, 1.0039062, 1.015625, 1.0664062)$$

Enfin, pour le calcul de l'espérance par la programmation dynamique, nous avons lancé l'algorithme fourni dans l'énoncé, pour un ensemble $\mathbb{W} = \{-1, 1\}$. C'est en fait, une manière récursive de calculer le coût optimal.

On obtient alors le vecteur des *payoff* suivant

$$payoff_{PD} = (1.0664062, 1.015625, 1.0039062, 1, 1, 1, 1.0039062, 1.015625, 1.0664062)$$

On retrouve donc bien le même résultat pour les deux dernières méthodes de calcul exact du *payoff*.

La convergence de la méthode de Monte-Carlo vers les bonnes valeurs est également correcte et approche bien les valeurs optimales comme le montre la figure ci-dessous

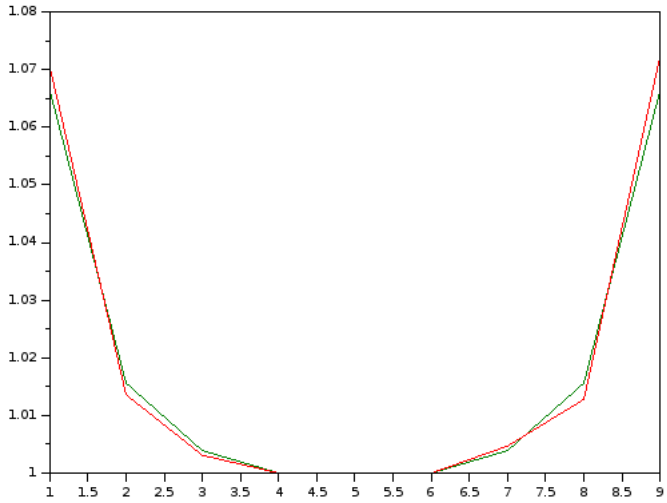

FIGURE 2 – Tracé des *payoff* en fonction des états pour les méthodes exactes (courbe verte) et la méthode de Monte-Carlo (courbe rouge)

Puis pour comparer ces méthodes avec deux types de Monte-Carlo avec forte convergence ($N = 1000$) et avec plus faible convergence ($N = 1000$) (courbe en cyan) on obtient les courbes ci dessous (Figure [3])

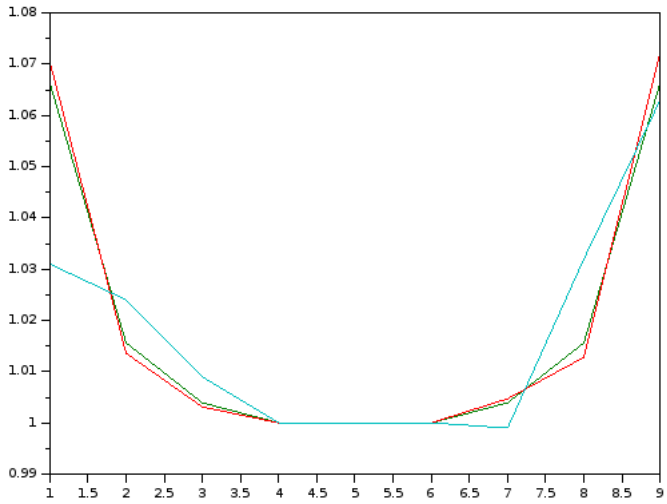

FIGURE 3 – Tracé des coûts en fonction de l'état x_0 pour 4 méthodes

Enfin, nous comparons les temps passés par le CPU de la machine pour calculer des trois méthodes différentes le *payoff* :

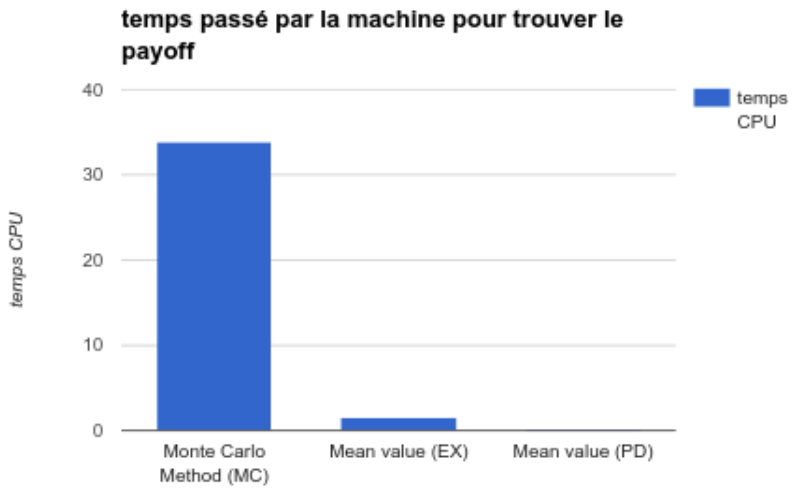

FIGURE 4 – Tracé du temps passer sur chaque méthode de calcul

Nous remarquons donc que pour Monte-Carlo avec un nombre d'itération de 10000, le temps de calcul est très grand par rapport aux autres méthodes.
Néanmoins, c'est la méthode la plus utilisée pratiquement car pour un plus petit nombre d'itérations, elle donne une assez bonne approximation du *payoff* en un temps raisonnable.
Dans ce cas pratique, nous avions d'assez faibles dimensions temporelles et spatiales, mais dans d'autres études d'optimisation par programmation dynamique, la méthode de Monte-Carlo pourrait s'avérer très utile et être utilisée comme une heuristique du problème.